SANDIA REPORT
SAND2024-04049
Printed April 2024

Sandia National Laboratories

SCEPTRE 2.7 User's Guide

Shawn D. Pautz and Harley Hanes

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico
87185 and Livermore,
California 94550

Issued by Sandia National Laboratories, operated for the United States Department of Energy by National Technology & Engineering Solutions of Sandia, LLC.

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

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@osti.gov
Online ordering: http://www.osti.gov/scitech

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Rd
Alexandria, VA 22312

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.gov
Online order: https://classic.ntis.gov/help/order-methods/

ABSTRACT

Sandia's Computational Engine for Particle Transport for Radiation Effects (SCEPTRE) is a computer code that solves the linear Boltzmann transport equation, particularly targeting coupled photon-electron problems. It uses unstructured finite element meshes in space, multigroup in energy, and discrete ordinates (S_n) or other methods in angle. SCEPTRE uses an xml-based input file to specify the problem. This report documents the options and syntax of that input file.

ACKNOWLEDGEMENTS

We recognize the many years of effort of several past SCEPTRE developers, whose contributions enabled many of its capabilities.

CONTENTS

LIST OF FIGURES

ACRONYMS AND TERMS

Acronym/Term	Definition
BXSLIB	Binary Cross-Section Library
CEPXS	Coupled Electron-Photon Cross Section
GMRES	Generalized Minimal Residual Method
P_n	Spherical harmonics
SCEPTRE	Sandia's Computational Engine for Particle Transport for Radiation Effects
S_n	Discrete ordinates
XSLIB	Cross-Section Library

1. INTRODUCTION

SCEPTRE is a "general-purpose" linear Boltzmann transport solver. It is a massively parallel code with multiple solvers and various options to control its execution. Some general characteristics of SCEPTRE are:

- Multigroup or finite element in the energy variable
- Discrete ordinates (S_n) or spherical harmonics (P_n) in the angular variable
- Finite elements on unstructured meshes in the spatial variable

More details of the discretizations and solvers may be found in [1].

SCEPTRE accepts an input file in the xml format to define the problem and how it is to be solved. Other files (e.g., meshes) are defined within that input file. The purpose of this document is to describe the format of the xml input file for *some* of the available options in version 2.7 of the code. We document the use of the first-order sweeps-based solver and the first-order GMRES solver. We do not document other solvers, or the first-collision and last-collision capabilities. It is intended that future versions of this manual will provide more comprehensive documentation of all the options in the code.

2. XML INPUT SECTIONS

In this section we describe the input format for SCEPTRE. The input file uses the xml format to describe the problem and how it is to be solved. There is no expectation that a user has a thorough understanding of the xml format, as we will define how it is used for SCEPTRE.

An example input file is provided in Figure 1 below, which illustrates many of the available options in SCEPTRE. A fuller description of the syntax and allowable values is contained in succeeding subsections.

We note that a SCEPTRE input file must begin with the following line:

```
<?xml version="1.0" encoding="utf-8"?>
```

After that line must follow the main section heading:

```
<SCEPTRE_Input>
```

which may be optionally closed with the end of that section:

```
</SCEPTRE_Input>
```

All other content must be contained within the `SCEPTRE_Input` section. Indentation of each section or subsection may help provide clarity but is not required. We also note that each section may occur in any order but will be parsed by SCEPTRE in an order constrained by dependencies of the input.

```
<?xml version="1.0" encoding="utf-8"?>
<SCEPTRE_Input>

  <Output_Options>
    <Write_Restart>true</Write_Restart>
    <Cache_Results>true</Cache_Results>
    <Verbosity>high</Verbosity>
  </Output_Options>

  <Transport_Mode>Forward</Transport_Mode>

  <Mesh_File>mesh/rg402Tri3.par</Mesh_File>

  <XS_File>xsec/rg402_10g.xslib</XS_File>

  <Output_Prefix>rg402Tri3</Output_Prefix>
  <Output_Format>Exodus</Output_Format>

  <Sn_Options>
    <Sn_Order>4</Sn_Order>

<Angular_Quadrature_Type>Level_Symmetric</Angular_Quadrature_Typ
e>
  </Sn_Options>
```

```
  <Scattering_Options>
    <Scattering_Order>1</Scattering_Order>

<Delta_Function_Scattering_Correction>true</Delta_Function_Scatt
ering_Correction>

<Scattering_Moments_Method>Standard</Scattering_Moments_Method>
  </Scattering_Options>

  <Outer_Iteration_Options>
    <Maximum_Number_Iterations>10</Maximum_Number_Iterations>
    <Convergence_Tolerance>1.e-3</Convergence_Tolerance>
  </Outer_Iteration_Options>


<Enable_User_Defined_Solvers>true</Enable_User_Defined_Solvers>
  <Solvers>
    <Solver name="1stOrder">
      <Solver_Form>First_Order</Solver_Form>
      <Element_Set_Size>1</Element_Set_Size>
      <Coarse_Sn_Order>4</Coarse_Sn_Order>
      <Error_Control_Options>

<Maximum_Number_Iterations>100</Maximum_Number_Iterations>
        <Convergence_Tolerance>1.e-4</Convergence_Tolerance>
        <Error_Metrics>
          <Error_Metric>
            <Metric_Type>whole</Metric_Type>
            <Integration_Policy>discrete</Integration_Policy>
            <Error_Norm>L</Error_Norm>
            <Error_Order>2</Error_Order>
            <Sign_Policy>absolute</Sign_Policy>
          </Error_Metric>
        </Error_Metrics>
      </Error_Control_Options>
    </Solver>
  </Solvers>

  <Solver_Assignment explicit="true">
    <Solver_By_Group_Range>1stOrder 1 8</Solver_By_Group_Range>
    <Solver_By_Group>
      <Group index="9">1stOrder</Group>
      <Group index="10">1stOrder</Group>
    </Solver_By_Group>
  </Solver_Assignment>
```

```
<Materials enable="true">
  <Material name="copper">
    <Density>7.874</Density>
    <Conductor>true</Conductor>
  </Material>
  <Material name="iron">
    <Density>8.96</Density>
    <Conductor>true</Conductor>
  </Material>
  <Material name="silver">
    <Density>10.5</Density>
    <Conductor>true</Conductor>
  </Material>
  <Material name="teflon">
    <Density>2.2</Density>
    <Conductor>false</Conductor>
  </Material>
</Materials>

<Material_Assignment enable="true">
  <ElementBlock name="block1">copper</ElementBlock>
  <ElementBlock name="block2">iron</ElementBlock>
  <ElementBlock name="block3">silver</ElementBlock>
  <ElementBlock name="block4">teflon</ElementBlock>
  <ElementBlock name="block5">copper</ElementBlock>
</Material_Assignment>

<Fixed_Source_Options enable="true">
  <Source>
  </Source>
  <Source>
    <Scale_Factor>3.</Scale_Factor>
  </Source>
  <Source>
    <Scale_Factor>2.</Scale_Factor>
    <Energy_Expansion enable="true">
      <Group index="1">0.5</Group>
      <Group index="2">0.2</Group>
      <Group index="3">0.1</Group>
      <Group_Range>0.3 4 6</Group_Range>
    </Energy_Expansion>
  </Source>
  <Source>
    <Scale_Factor>8.</Scale_Factor>
    <Energy_Expansion enable="true">
      <Group index="1">2.5</Group>
      <Group index="6">3.5</Group>
```

```
      </Energy_Expansion>
      <Angle_Expansion enable="true">
        <Angle index="1">3.2</Angle>
        <Angle_Range>3.3 2 3</Angle_Range>
      </Angle_Expansion>
      <Block_Expansion enable="true">
        <Block name="block4">5.</Block>
      </Block_Expansion>
    </Source>
  </Fixed_Source_Options>

  <Boundary_Source_Options enable="true">
    <Source>
    </Source>
    <Source>
      <Scale_Factor>3.</Scale_Factor>
    </Source>
    <Source>
      <Scale_Factor>2.</Scale_Factor>
      <Energy_Expansion enable="true">
        <Group index="1">0.5</Group>
        <Group index="2">0.2</Group>
        <Group index="3">0.1</Group>
        <Group_Range>0.3 4 6</Group_Range>
      </Energy_Expansion>
    </Source>
    <Source>
      <Scale_Factor>8.</Scale_Factor>
      <Energy_Expansion enable="true">
        <Group index="1">2.5</Group>
        <Group index="6">3.5</Group>
      </Energy_Expansion>
      <Angle_Expansion enable="true">
        <Angle index="1">3.2</Angle>
        <Angle_Range>3.3 2 3</Angle_Range>
      </Angle_Expansion>
      <SideSet_Expansion enable="true">
        <SideSet name="sideset5">5.</SideSet>
      </SideSet_Expansion>
    </Source>
  </Boundary_Source_Options>

</SCEPTRE_Input>
```

Figure 1: SCEPTRE input file

2.1. Mesh_File

All SCEPTRE input files must contain the `Mesh_File` section, as illustrated in Figure 1 and shown below:

`<Mesh_File>`*`mesh file name`*`</Mesh_File>`

with the italicized text replaced by the name of the mesh file. SCEPTRE requires a file in the Genesis or Exodus format. For serial runs this will be a single file. For parallel runs these meshes must be decomposed via various tools (e.g. nem_slice/nem_spread) into multiple files in the Nemesis format. Decomposed files must have the following naming convention:

`meshfilename.ncores.0, meshfilename.ncores.1, …`

where *`meshfilename`* will be indicated in the `<Mesh_File>` section, *`ncores`* is the number of MPI ranks in the calculation, and the suffix of each file corresponds to the processor ID. Note that the latter two parts of the file name are *not* indicated in the `<Mesh_File>` section, but instead are deduced in the code from the MPI context.

2.2. Output_Prefix

All SCEPTRE input files must contain the `Output_Prefix` section, as illustrated in Figure 1 and shown below:

`<Output_Prefix>`*`output prefix`*`</Output_Prefix>`

with the italicized text replaced by the name of the output mesh. The same considerations for serial versus parallel filenames as with the input mesh apply. SCEPTRE will always create an output Exodus mesh with the same geometrical data as the input mesh. Depending on other settings the output mesh may have additional contents, in particular the solution.

2.3. Output_Format

The amount of data that SCEPTRE generates can be quite large. This data generally is written out during the code run, which in some cases can be time-consuming. To help manage that I/O time there are several output format options available, which is indicated in the (optional) `Output_Format` section, as indicated below:

`<Output_Format>`*`format`*`</Output_Format>`

Valid values for *`format`* are `Exodus` (the default), `netcdf`, `binary`, and `simple`. If the format specified is `Exodus` then the data will be written to the output Exodus file (indicated in the previous section). If the format is `netcdf` or `binary` then a separate file will be created in one of those formats to which angular fluxes will be written. If the format specified is `simple` then no output data will be written, which may be useful for some testing purposes.

2.4. Dump_File_Prefix

If the output format specified above is either simple or binary, then it will be necessary for SCEPTRE to create a file in that format in addition to the usual Exodus file. The default name for

that file is `DumpFile`. The user may optionally change that name with the `Dump_File_Prefix` section, as shown below:

`<Dump_File_Prefix>`*`dump file prefix`*`</Dump_File_Prefix>`

2.5. Use_Existing_Dumpfiles

When running SCEPTRE multiple times, the user may elect to use the same name for the output files. If the netcdf or binary formats are used, the user may decide to reuse an existing file rather than recreating it. This is done with the optional `Use_Existing_Dumpfiles` section, as indicated below:

`<Use_Existing_Dumpfiles>[true | false]</Use_Existing_Dumpfiles>`

The default value is `false`, which will result in (re)creating a dump file.

2.6. Output_Options

The `Output_Options` section will control some of the behavior of SCEPTRE output. It has multiple options indicated in the subsections below. Its syntax is:

```
<Output_Options>
  options
</Output_Options>
```

where *`options`* are specified below.

2.6.1. Write_Restart

Unless specified otherwise, SCEPTRE will write the angular flux solution to output files. The user can control how often this occurs with the optional `Write_Restart` subsection within the `Output_Options` section, as follows:

`<Write_Restart>[true | false]</Write_Restart>`

with a default of `true`. If `true` is indicated, SCEPTRE will write out the angular fluxes for each energy group as each group is converged. Otherwise, SCEPTRE will wait until the end of the calculation to write out the solution. More frequent writes can be helpful if the calculation needs to proceed in stages (e.g., if time limits prevent the entire calculation from completing in one job submission), since then the most recent results are available when restarting the calculation. This will come at the expense of more I/O.

2.6.2. Cache_Results

SCEPTRE has the option of either keeping all angular fluxes stored in memory (caching), or of keeping only one group's worth of data. The latter can be useful if on a system with limited

memory, but it does require much more file I/O. This is controlled with the optional `Cache_Results` subsection within the `Output_Options` section, as follows:

```
<Cache_Results>[true | false]</Cache_Results>
```

with a default of `true`.

2.6.3. Verbosity

SCEPTRE will output some information concerning the calculation to `stdout` as it proceeds. The amount of information can be controlled with the optional `Verbosity` subsection within the `Output_Options` section, as follows:

```
<Verbosity>[high | medium | low]</Verbosity>
```

with a default of `medium`.

2.7. Transport_Mode

SCEPTRE can be run in either forward or adjoint mode. This is controlled by the optional `Transport_Mode` section, as follows:

```
<Transport_Mode>[forward | adjoint]</Transport_Mode>
```

with a default of `forward`.

2.8. Boundary_Condition_Assignment

In SCEPTRE a "boundary condition" refers to the geometrical nature of mesh surfaces, e.g., if there are reflective conditions (source terms are a separate topic). Specification of such terms are contained within an optional `Boundary_Condition_Assignment` section, which will enclose one or more assignments, with a syntax as follows:

```
<Boundary_Condition_Assignment>
  assignment types
</Boundary_Condition_Assignment>
```

where *`assignment types`* are specified below.

2.8.1. Assignment_By_ID

Assignments of boundary conditions may be done by specifying the "sideset" ID of the mesh file. To indicate that this is how assignments will be made, an optional `Assignment_By_ID` subsection may be included within the `Boundary_Condition_Assignment` section, with a syntax as follows:

```
<Assignment_By_ID>
  assignments
```

</Assignment_By_ID>

where *assignments* are specified below.

2.8.1.1. Boundary_Condition

The actual assignment of a boundary condition occurs in an optional Boundary_Condition subsection within the Assignment_By_ID section. There are three types of Boundary Conditions:

1. Vacuum

   "Vacuum" conditions are intended to indicate that the relevant surface does not face any other material; radiation that crosses that surface from the interior "escapes into the void" from that exterior surface. It is not meant to indicate that there are zero sources there. Vacuum boundary conditions are indicated with the following syntax:

   <Boundary_Condition>vacuum *ID*</Boundary_Condition>

   where *ID* is the sideset ID defined within the mesh file. All sidesets along the outer surface of the mesh are vacuum by default. Note that SCEPTRE will always create an externalBoundary sideset that includes all sides on the exterior of the mesh in every run, which assists in this default assignment.

2. Reflective

   Reflective conditions indicate that radiation streaming from the interior through an external surface will undergo specular reflection back into the problem. SCEPTRE can only handle reflective conditions when the surface is aligned with one of the Cartesian planes, i.e., the surface normal is along one of the coordinate axes. Reflective boundary conditions are indicated with the following syntax:

   <Boundary_Condition>reflective *ID*</Boundary_Condition>

3. Periodic

   Periodic conditions indicate that radiation streaming out from a surface will reenter on the opposite surface, e.g., radiation "escaping" the right side will reenter along the left side. Periodic boundary conditions are indicated with the following syntax:

   <Boundary_Condition>periodic *ID1 ID2*</Boundary_Condition>

   where *ID1* and *ID2* refer to the opposing sidesets. Note that for periodic conditions to work there must be an exact 1:1 correspondence between the nodes and faces of the two sidesets. If that condition is met SCEPTRE should be able to determine how the sides pair with each other.

2.8.2. ("Assignment_By_Name")

Assignments of Boundary Conditions also may be done by sideset name. This is the default way of specifying them; there is not a special keyword to indicate that, as is the case with

`Assignment_By_ID`. Instead, one will provide `Boundary_Condition` subsections directly within the `Boundary_Condition_Assignment` section, as described below.

2.8.2.1. Boundary_Condition

Specification of a Boundary Condition with sideset names looks very similar to that done with sideset IDs. The following syntax is used:

1. Vacuum

   `<Boundary_Condition>vacuum` *`name`*`</Boundary_Condition>`

   where *`name`* is the sideset name defined within the mesh file.

2. Reflective

   `<Boundary_Condition>reflective` *`name`*`</Boundary_Condition>`

   where *`name`* is the sideset name defined within the mesh file.

3. Periodic

   `<Boundary_Condition>periodic` *`name1 name2`*
   `</Boundary_Condition>`

   where *`name1`* and *`name2`* refer to the opposing sideset names.

2.9. Boundary_Source_Options

For those surfaces with vacuum boundary conditions, we may specify boundary sources. This may be done in one of two ways:

1. Implicitly

   Input files, generally the Exodus input file, may have boundary values specified in them. This is particularly useful if the boundary source is complicated. Such sources are generally created by special preprocessing tools prior to a SCEPTRE run; we will not describe them here. Such sources are implicitly read in without user intervention (e.g. when reading in an input Exodus file) and are zero by default.

2. Explicitly

   The optional `Boundary_Source_Options` section may be used in the SCEPTRE input file to directly provide boundary sources. If this section is provided it overrides any implicit boundary source definition.

The syntax for the `Boundary_Source_Options` section is:

```
<Boundary_Source_Options enable="true">
  options
  sources
</Boundary_Source_Options>
```

where *options* and *sources* are specified below.

2.9.1. Input_From_Separate_File

Boundary sources may either be specified directly in the SCEPTRE input file, or they may be defined in a separate file. In the latter case the `Input_From_Separate_File` option may be specified within the `Boundary_Source_Options` section. The syntax for that is:

`<Input_From_Separate_File enable="true">`*source file name*`</Input_From_Separate_File>`

where *source file name* is the file that contains the boundary source specification. That file must begin with the same preamble as the main SCEPTRE input file:

```
<?xml version="1.0" encoding="utf-8"?>
```

It also must contain a `Boundary_Source_Options` section.

2.9.2. Source

Boundary sources are specified by composition of one or more sources. Each source consists of the product of a scaling factor, an energy expansion, an angular expansion, and a spatial expansion. In other words:

$Q(\mathbf{r},E,\Omega) = c_1 f_1(\mathbf{r}) g_1(E) h_1(\Omega) + c_2 f_2(\mathbf{r}) g_2(E) h_2(\Omega) + \ldots$

Each source is specified by the `Source` subsection, with the following syntax:

```
<Source>
  expansions
</Source>
```

where *expansions* are specified below.

2.9.2.1. Scale_Factor

A constant multiplication factor may be provided for a source in the optional `Scale_Factor` subsection. The syntax for this is

`<Scale_Factor>`*factor*`</Scale_Factor>`

where *factor* is the real constant. If `Scale_Factor` is omitted from a `Source` section, then the multiplication factor is 1.0 by default.

2.9.2.2. Energy_Expansion

The energy dependence of a source may be specified by the `Energy_Expansion` subsection. The syntax for this is:

```
<Energy_Expansion enable="true">
  energy expansions
</Energy_Expansion>
```

where *energy expansions* are specified below. If `Energy_Expansion` is omitted from a `Source` section, then the default behavior is to assume a flat spectrum, i.e. g(E) = 1.

2.9.2.2.1. Group

An energy expansion for a single energy group may be specified by the `Group` subsection of the `Energy_Expansion` section. The syntax is:

```
<Group index="index">factor</Group>
```

where *index* is the group index and *factor* is the real constant. Note that the convention in radiation transport is for energy groups to be numbered starting at 1, from highest energy groups down to lowest energy groups. Multiple `Group` subsections may be used in the `Energy_Expansion` section, one per group. Any groups omitted will default to a zero source.

2.9.2.2.2. Group_Range

Instead of listing one group at a time, a range of groups may have an expansion specified by means of the `Group_Range` subsection of the `Energy_Expansion` section. The syntax is

```
<Group_Range>factor index1 index2</Group_Range>
```

Where *factor* is the real constant, *index1* is the lowest-numbered group in the range, and *index2* is the highest-numbered group in the range.

2.9.2.3. Angle_Expansion

The angular dependence of a source may be specified by the `Angle_Expansion` subsection. The syntax for this is:

```
<Angle_Expansion enable="true">
  angle expansions
</Angle_Expansion>
```

where *angle expansions* are specified below. If `Angle_Expansion` is omitted from a `Source` section, then the default behavior is to assume an isotropic source, i.e., h(Ω) = 1.

2.9.2.3.1. Angle

An angle expansion for a single angle may be specified by the `Angle` subsection of the `Angle_Expansion` section. The syntax is:

```
<Angle index="index">factor</Angle>
```

where *index* is the angle index and *factor* is the real constant. Note that, unfortunately, there is not a single convention for numbering angles in various angular quadrature sets. However, for quadratures that include a normally-incident angle, an index of 1 is assigned to the direction in the +x direction. Multiple `Angle` subsections may be used in the `Angle_Expansion` section, one per angle. Any angles omitted will default to a zero source.

2.9.2.3.2. Angle_Range

Instead of listing one angle at a time, a range of angles may have an expansion specified by means of the `Angle_Range` subsection of the `Angle_Expansion` section. The syntax is

`<Angle_Range>`*factor index1 index2*`</Angle_Range>`

where *factor* is the real constant, *index1* is the lowest-numbered angle in the range, and *index2* is the highest-numbered angle in the range.

2.9.2.4. SideSet_Expansion

The spatial dependence of a boundary source may be specified by the `SideSet_Expansion` subsection. The syntax for this is:

```
<SideSet_Expansion enable="true">
  sideset expansions
</SideSet_Expansion>
```

where *sideset expansions* are specified below. If `SideSet_Expansion` is omitted from a `Source` section, then the default behavior is to assume a spatially uniform source, i.e., f(**r**) = 1. Note that boundary sources can only be applied to the actual boundary of the problem, not to interior sidesets. Also, any default values will be applied to the entire boundary of the problem, whether or not the user has included the entire boundary in one or more sidesets.

2.9.2.4.1. SideSet

A sideset expansion for a single sideset may be specified by the `SideSet` subsection of the `SideSet_Expansion` section. The syntax is:

`<SideSet name="`*sideset name*`">`*factor*`</SideSet>`

where *sideset name* is the name of the sideset and *factor* is the real constant. Multiple `SideSet` subsections may be used in the `SideSet_Expansion` section, one per sideset. Any sidesets omitted (including sides not defined in a sideset) will default to a zero source.

2.10. Fixed_Source_Options

Fixed, internal (volumetric) sources may be specified for the problem. We refer to them as "fixed" since these are distinct from scattering sources, which evolve as the problem is iteratively solved. This may be done in one of two ways:

1. Implicitly

   Input files, generally the Exodus input file, may have fixed source values specified in them. This is particularly useful if the fixed source is complicated. Such sources are generally created by special preprocessing tools prior to a SCEPTRE run; we will not describe them here. Such sources are implicitly read in without user intervention (except for the `Binary_Fixed_Source_Input` option, described later) and are zero by default.

2. Explicitly

   The optional `Fixed_Source_Options` section may be used in the SCEPTRE input file to directly provide fixed sources. If this section is provided it overrides any implicit fixed source definition.

The syntax for the `Fixed_Source_Options` section is:

```
<Fixed_Source_Options enable="true">
  options
  sources
</Fixed_Source_Options>
```

where *`options`* and *`sources`* are specified below.

2.10.1. Input_From_Separate_File

Fixed sources may either be specified directly in the SCEPTRE input file, or they may be defined in a separate file. In the latter case the `Input_From_Separate_File` option may be specified within the `Fixed_Source_Options` section. The syntax for that is:

```
<Input_From_Separate_File enable="true">source file name</Input_From_Separate_File>
```

where *`source file name`* is the file that contains the fixed source specification. That file must begin with the same preamble as the main SCEPTRE input file:

```
<?xml version="1.0" encoding="utf-8"?>
```

It also must contain a `Fixed_Source_Options` section.

2.10.2. Source

Fixed sources are specified by composition of one or more sources. Each source consists of the product of a scaling factor, an energy expansion, an angular expansion, and a spatial expansion. In other words:

$Q(\mathbf{r},E,\Omega) = c_1 f_1(\mathbf{r}) g_1(E) h_1(\Omega) + c_2 f_2(\mathbf{r}) g_2(E) h_2(\Omega) + \ldots$

Each source is specified by the `Source` subsection, with the following syntax:

```
<Source>
  expansions
```

</Source>

where *expansions* are specified below.

2.10.2.1. Scale_Factor

A constant multiplication factor may be provided for a source in the optional Scale_Factor subsection. The syntax for this is

<Scale_Factor>*factor*</Scale_Factor>

where *factor* is the real constant. If Scale_Factor is omitted from a Source section, then the multiplication factor is 1.0 by default.

2.10.2.2. Energy_Expansion

The energy dependence of a source may be specified by the Energy_Expansion subsection. The syntax for this is:

<Energy_Expansion enable="true">

energy expansions

</Energy_Expansion>

where *energy expansions* are specified below. If Energy_Expansion is omitted from a Source section, then the default behavior is to assume a flat spectrum, i.e. g(E) = 1.

2.10.2.2.1. Group

An energy expansion for a single energy group may be specified by the Group subsection of the Energy_Expansion section. The syntax is:

<Group index="*index*">*factor*</Group>

where *index* is the group index and *factor* is the real constant. Note that the convention in radiation transport is for energy groups to be numbered starting at 1, from highest energy groups down to lowest energy groups. Multiple Group subsections may be used in the Energy_Expansion section, one per group. Any groups omitted will default to a zero source.

2.10.2.2.2. Group_Range

Instead of listing one group at a time, a range of groups may have an expansion specified by means of the Group_Range subsection of the Energy_Expansion section. The syntax is

<Group_Range>*factor index1 index2*</Group_Range>

Where *factor* is the real constant, *index1* is the lowest-numbered group in the range, and *index2* is the highest-numbered group in the range.

2.10.2.3. Angle_Expansion

The angular dependence of a source may be specified by the `Angle_Expansion` subsection. The syntax for this is:

```
<Angle_Expansion enable="true">
  angle expansions
</Angle_Expansion>
```

where *`angle expansions`* are specified below. If `Angle_Expansion` is omitted from a `Source` section, then the default behavior is to assume an isotropic source, i.e., $h(\Omega) = 1$.

2.10.2.3.1. Angle

An angle expansion for a single angle may be specified by the `Angle` subsection of the `Angle_Expansion` section. The syntax is:

```
<Angle index="index">factor</Angle>
```

where *`index`* is the angle index and *`factor`* is the real constant. Note that, unfortunately, there is not a single convention for numbering angles in various angular quadrature sets. However, for quadratures that include a normally-incident angle, an index of 1 is assigned to the direction in the +x direction. Multiple `Angle` subsections may be used in the `Angle_Expansion` section, one per angle. Any angles omitted will default to a zero source.

2.10.2.3.2. Angle_Range

Instead of listing one angle at a time, a range of angles may have an expansion specified by means of the `Angle_Range` subsection of the `Angle_Expansion` section. The syntax is

```
<Angle_Range>factor index1 index2</Angle_Range>
```

where *`factor`* is the real constant, *`index1`* is the lowest-numbered angle in the range, and *`index2`* is the highest-numbered angle in the range.

2.10.2.4. Block_Expansion

The spatial dependence of a fixed source may be specified by the `Block_Expansion` subsection. The syntax for this is:

```
<Block_Expansion enable="true">
  block expansions
</Block_Expansion>
```

where *`block expansions`* are specified below. If `Block_Expansion` is omitted from a `Source` section, then the default behavior is to assume a spatially uniform source, i.e., $f(\mathbf{r}) = 1$.

2.10.2.4.1. Block

A block expansion for a single Exodus element block may be specified by the `Block` subsection of the `Block_Expansion` section. The syntax is:

`<Block name="`*`block name`*`">`*`factor`*`</Block>`

where *`block name`* is the name of the block and *`factor`* is the real constant. Multiple `Block` subsections may be used in the `Block_Expansion` section, one per block. Any blocks omitted will default to a zero source.

2.11. Binary_Fixed_Source_Input

If a fixed source is available in a binary format, it is indicated via the optional `Binary_Fixed_Source_Input` section. The syntax for this is:

```
<Binary_Fixed_Source_Input enable="true">
  <File_Prefix>file name</File_Prefix>
</Binary_Fixed_Source_Input>
```

where *`file name`* is the name of the binary file.

2.12. Source_Options

The optional `Source_Options` section will control some of the behavior of SCEPTRE sources. It has the single option of `Cache_Sources`. Its syntax is:

```
<Source_Options>
  <Cache_Sources>[true | false]</Cache_Sources>
</Source_Options>
```

with a default of "`false`". If "`true`" is indicated, SCEPTRE will save the fixed sources in memory for all groups. Otherwise, SCEPTRE will read in each group's source from file as it is needed. This allows a tradeoff between memory and I/O.

2.13. XS_File

All SCEPTRE input files must contain the `XS_File` section. The syntax is:

`<XS_File type=[netcdf | bxslib]>`*`file name`*`</XS_File>`

where *`file name`* is the name of the cross-section file and `type` refers to the format of the cross-section file. The default type is `netcdf`.

2.14. Materials

Information about materials in the problem are specified in a Materials section. (Technically this is an optional section, but the default behavior is to define a single material named "mat1" with simple properties, which in real problems is generally not correct.) The main purpose of this section is to map the material data contained in the cross section file to additional metadata (e.g. material name) needed by SCEPTRE. Thus, it is vital to list material data *in the same order* as it is defined in the CEPXS cross section file; there will be a 1-1 mapping imposed by that ordering. In the future it is hoped that a new cross section file format will contain such metadata, but the current BXSLIB format (and its netcdf equivalent) does not allow for it.

The syntax for the `Materials` section is:

```
<Materials enable="true">
  options
materials
</Materials>
```

where *`options`* and *`materials`* are specified below.

2.14.1. Input_From_Separate_File

Materials may either be specified directly in the SCEPTRE input file, or they may be defined in a separate file. In the latter case the `Input_From_Separate_File` option may be specified within the `Materials` section. The syntax for that is:

```
<Input_From_Separate_File enable="true">materials file name</Input_From_Separate_File>
```

where *`materials file name`* is the file that contains the materials specifications. That file must begin with the same preamble as the main SCEPTRE input file:

```
<?xml version="1.0" encoding="utf-8"?>
```

It also must contain a `Materials` section.

2.14.2. Material

Each material in the CEPXS cross section file must have a corresponding `Material` subsection within the `Materials` section. The syntax for each `Material` subsection is:

```
<Material name="material name">
material data
</Material>
```

where *`material name`* is the user-defined name for that material and *`material data`* is described in the following subsections. Note that *`material name`* need not match any name listed in the CEPXS input file, as such names are not encoded in the resulting CEPXS cross section

file. However, *material name* may be used in other SCEPTRE input sections, so consistency within the SCEPTRE input file is required.

2.14.2.1. Density

An optional `Density` subsection may be used in the `Material` section. Note that this will not scale the cross sections if it differs from what was originally used in the CEPXS generation (i.e. it will not affect the transport calculation), but it may be used in some postprocessing. The syntax for the `Density` subsection is:

`<Density>`*value*`</Density>`

where *value* is the real constant for the density, with a default of 1.

2.14.2.2. Conductor

An optional `Conductor` subsection may be used in the `Material` section, with a syntax of:

`<Conductor>[`*true* | *false*`]</Conductor>`

with a default of `false`.

2.14.2.3. DielectricConstant

An optional `DielectricConstant` subsection may be used in the `Material` section, with a syntax of:

`<DielectricConstant>`*value*`</DielectricConstant>`

where *value* is the real constant for the dielectric constant, with a default of 1.

2.14.2.4. ElectricalConductivity

An optional `ElectricalConductivity` subsection may be used in the `Material` section, with a syntax of:

`<ElectricalConductivity>`*value*`</ElectricalConductivity>`

where *value* is the real constant for the conductivity, with a default of 0.

2.15. Mixtures

SCEPTRE provides the ability to create mixtures from existing materials. This can be from the original CEPXS materials or from other mixtures. This is done in an optional `Mixtures` section, with the following syntax:

```
<Mixtures enable="true">
  options
```

mixtures

</Mixtures>

where *options* and *mixtures* are specified below.

2.15.1. Input_From_Separate_File

Mixtures may either be specified directly in the SCEPTRE input file, or they may be defined in a separate file. In the latter case the Input_From_Separate_File option may be specified within the Mixtures section. The syntax for that is:

<Input_From_Separate_File enable="true">*mixtures file name*</Input_From_Separate_File>

where *mixtures file name* is the file that contains the mixtures specifications. That file must begin with the same preamble as the main SCEPTRE input file:

<?xml version="1.0" encoding="utf-8"?>

It also must contain a Mixtures section.

2.15.2. Mixture

Each mixture is defined in a Mixture subsection of the Mixtures section. The syntax for each Mixture subsection is:

<Mixture name="*mixture name*">

  <Composition>

    material fraction

  </Composition>

</Mixture>

where *mixture name* is the user-defined name for that mixture and *material fraction* is described in the following subsection.

2.15.2.1. Material_Fraction

A mixture is composed of one or more materials or previously defined mixtures, each of which is listed in a Material_Fraction subsection of the Mixture section. The syntax is:

<Material_Fraction>*material fraction*</Material_Fraction>

where *material* is the material name and *fraction* is the "volume fraction" of that material in the mixture. Note that the "fractions" of all the materials in a mixture do not need to sum to unity; this is done in lieu of specifying a density. For example, one may define a mixture that is a single material at half nominal density (as defined in the CEPXS cross section file) by providing a single Material_Fraction subsection that uses a fraction of 0.5. Another use case is to define a "void" material that consists of any available material with a fraction of 0. Another use case is to

define molecules this way: instead of defining all possible materials in the CEPXS cross section file, one can simply provide the needed elements in the CEPXS file and then arbitrarily mix them in a `Mixture` section.

2.16. Material_Assignment

The assignment of materials and/or mixtures to element blocks is performed in the `Material_Assignment` section. (Technically this is an optional section, but the default behavior is to assign the first material from the `Materials` section to all element blocks, which in real problems is generally not correct.) Not all available materials need to be assigned to an element block, but all element blocks should be assigned a material. The `Material_Assignment` section has the following syntax:

```
<Material_Assignment enable="true">
  assignments
</Material_Assignment>
```

where *`assignments`* are specified below.

2.16.1. ElementBlock

One method of assigning materials to element blocks is by element block name. This is done with an `ElementBlock` subsection of the `Material_Assignment` section, with the following syntax:

```
<ElementBlock name="block name">material name</ElementBlock>
```

where *`block name`* is the name of the Exodus element block and *`material name`* is the name of the material or mixture. This will only work if the Exodus file has a name for that element block; the name in `ElementBlock` must be identical.

2.16.2. Material

The other method of assigning materials to element blocks is by element block ID. This is done with a `Material` subsection of the `Material_Assignment` section, with the following syntax:

```
<Material block="block ID">material name</Material>
```

where *`block ID`* is the ID of the Exodus element block and *`material name`* is the name of the material or mixture.

2.17. Sn_Options

The solvers in SCEPTRE either use or interact with the discrete ordinates ("S_n") method. This method discretizes the direction of travel of radiation by defining a set of directions on the unit

sphere as well as an associated quadrature rule for computing spherical harmonic moments. Options for this method are specified in the `Sn_Options` section with the following syntax:

```
<Sn_Options>
  options
</Sn_Options>
```

where *options* are specified below.

2.17.1. Sn_Order

Each angular quadrature has an order; higher orders include more angles. The order is specified in the `Sn_Order` subsection of the `Sn_Options` section with the following syntax:

```
<Sn_Order>order</Sn_Order>
```

where *order* must be an even positive integer. Some quadrature types have further restrictions on available order.

2.17.2. Angular_Quadrature_Type

The type of angular quadrature (i.e. the rule that governs its generation) is given in the `Angular_Quadrature_Type` subsection of the optional `Sn_Options` section with the following syntax:

```
<Angular_Quadrature_Type>type</Angular_Quadrature_Type>
```

where *type* is a string that indicates the quadrature rule. Valid values for *type* are `Gauss_Legendre` (the default) and `Gauss_Lobatto` in 1D and `Level_Symmetric` (the default), `Fully_Symmetric`, `Lebedev`, and `Chebyshev` in multiD.

2.18. Scattering_Options

Options governing how the scattering of particles is treated are specified in the `Scattering_Options` section with the following syntax:

```
<Scattering_Options>
  options
</Scattering_Options>
```

where *options* are specified below.

2.18.1. Scattering_Order

The computation of particle scattering involves converting the discrete angular fluxes into spherical harmonic moments, then multiplying them by the Legendre expansion of the cross section given in

the CEPXS file. The order of scattering to be used is specified in the `Scattering_Order` subsection of the `Scattering_Options` section with the following syntax:

`<Scattering_Order>`*`order`*`</Scattering_Order>`

where *`order`* must be a non-negative integer that is less than or equal to the Legendre order contained in the CEPXS cross section file. `Scattering_Order` is a mandatory section with no default value.

2.18.2. Scattering_Moments_Method

Discrete ordinates angular flux values are converted into spherical harmonics moments according to the scattering order indicated above and by means of a rule (method). The method to be used is indicated with the optional `Scattering_Moments_Method` subsection of the `Scattering_Options` section with the following syntax:

`<Scattering_Moments_Method>`*`method`*`</Scattering_Moments_Method>`

where *`method`* is a string that indicates the scattering method. Valid values are `Standard` (the default) and `Galerkin`.

2.18.3. Delta_Function_Scattering_Correction

The delta function scattering correction is a method that can improve the accuracy of the scattering calculation for forward-peaked scattering when the scattering order is insufficient to resolve the physics. It is indicated in the `Delta_Function_Scattering_Correction` subsection of the `Scattering_Options` section with the following syntax:

`<Delta_Function_Scattering_Correction>[`*`true`* `|`
`false``]</Delta_Function_Scattering_Correction>`

If this correction is enabled, then the scattering order selected previously must be less than the Legendre order contained in the CEPXS cross section file.
`Delta_Function_Scattering_Correction` is a mandatory section with no default value.

2.18.4. Within_Group_Scattering

Physically, particles within an energy group may undergo scattering events that leave them in the same group. In some cases, it can be useful to disable that physics (e.g. to obtain uncollided fluxes). This scattering is enabled with the optional `Within_Group_Scattering` subsection of the `Scattering_Options` section with the following syntax:

`<Within_Group_Scattering>[`*`true`* `|`
`false``]</Within_Group_Scattering>`

with a default of `true`.

2.18.5. Group_To_Group_Scattering

Physically, particles within an energy group may undergo scattering events that leave them in a different group. In some cases, it can be useful to disable that physics (e.g. to obtain uncollided fluxes). This scattering is enabled with the optional `Group_To_Group_Scattering` subsection of the `Scattering_Options` section with the following syntax:

```
<Group_To_Group_Scattering>[true |
false]</Group_To_Group_Scattering>
```

with a default of `true`.

2.18.6. Adjust_Total_Cross_Section

If either of the above scattering physics is disabled, the user may wish to decrease the total cross section by a corresponding amount (e.g. in order to preserve particles). This is done in the optional `Adjust_Total_Cross_Section` subsection of the `Scattering_Options` section with the following syntax:

```
<Adjust_Total_Cross_Section>[true |
false]</Adjust_Total_Cross_Section>
```

with a default of `false`.

2.19. First_Collision_Source

(This capability will be documented in future versions of this guide.)

2.20. Last_Collision_Flux

(This capability will be documented in future versions of this guide.)

2.21. Outer_Iteration_Options

During a calculation SCEPTRE will need to converge the group-to-group scattering computation. This is done by cycling through each of the groups, (approximately) solving each one with fixed group-to-group inscattering; a single pass through each group is one outer iteration. Options to control that process are given in the `Outer_Iteration_Options` section with the following syntax:

```
<Outer_Iteration_Options>
  options
</Outer_Iteration_Options>
```

where *options* are specified below.

2.21.1. Maximum_Number_Iterations

The maximum number of outer iterations is controlled with the `Maximum_Number_Iterations` subsection of the `Outer_Iteration_Options` section with the following syntax:

`<Maximum_Number_Iterations>`*`maxIters`*`</Maximum_Number_Iterations>`

where *`maxIters`* must be a positive integer. `Maximum_Number_Iterations` is a mandatory section with no default value.

2.21.2. Convergence_Tolerance

During each outer iteration SCEPTRE will estimate the relative error of the process. The outer iterations will complete when either the maximum number of outer iterations is exceeded or when a convergence tolerance is achieved. That tolerance is specified in the `Convergence_Tolerance` subsection of the `Outer_Iteration_Options` section with the following syntax:

`<Convergence_Tolerance>`*`value`*`</Convergence_Tolerance>`

where *`value`* is a real number between 0 and 1. `Convergence_Tolerance` is a mandatory section with no default value.

2.22. Enable_User_Defined_Solvers

A SCEPTRE run will make use of one or more solvers to complete a simulation. Each energy group will be assigned a solver; each solver has a common interface for arbitrary group assignments. The intention is to allow the user to mix and match solvers based on numerical (e.g., convergence) considerations. The motivating case for this is coupled photon-electron transport, in which different solvers may be more suitable to one particle type than another. A "solver" consists not just of a solver type (e.g., a 1st-order sweeps-based solver), but also options to tailor its behavior.

In the future SCEPTRE may provide defaults for creating solvers. For now, the user must explicitly do so. This is indicated by the `Enable_User_Defined_Solvers` section with the following syntax:

`<Enable_User_Defined_Solvers>true</Enable_User_Defined_Solvers>`

Technically, the default for the above is `false`, but there is undefined behavior if the user does not indicate `true`.

2.23. Solvers

In conjunction with the above section, the definition of solvers to be constructed and used in a SCEPTRE run are provided in the `Solvers` section, with the following syntax:

```
<Solvers>
  solvers
</Solvers>
```

where *solvers* are specified below.

2.23.1. Solver

At least one solver must be specified in a simulation. This is indicated in a Solver subsection in the Solvers section, with the following syntax:

<Solver name="*solver name*">

  solver options

</Solver>

where *solver name* is a unique user-defined name for a solver and *solver options* are specified below.

2.23.1.1. Solver_Form

One mandatory subsection of the Solver section is Solver_Form, which specifies a pre-defined form, or type, of solver to be defined. This is indicated with the following syntax:

<Solver_Form>*solver form*</Solver_Form>

where *solver form* is one of several keywords. For now, the only one we will document is First_Order.

2.23.1.1.1. First_Order

The First_Order solver form is used to select a discontinuous-Galerkin discretization of the 1st-order form of the Boltzmann transport equation that is solved via a sweeps-based solver in conjunction with source iteration. The behavior of this solver is controlled by several options, which are indicated within the Solver section as indicated below.

2.23.1.1.1.1. Work_Per_Message

During the process of solving the first-order form with the sweeps process, data is continuously generated that must be communicated to other processors. In general, those processors may not be able to proceed due to dependencies until some data is received, so there is a cost to delaying sending data. On the other hand, processing and sending many small messages incurs more communication costs than sending fewer larger messages. The optimal message size and delay in sending will balance those two effects, but it may be difficult to determine the appropriate balance a priori. The user may use the optional Work_Per_Message subsection of the Solver section to specify how many computational tasks a processor should perform (if tasks are available) prior to sending and receiving any data, with the following syntax:

<Work_Per_Message>*work*</Work_Per_Message>

where *work* must be a positive integer, with a default of 1.

2.23.1.1.1.2. Element_Set_Size

The sweeps process involves traversing a directed graph. Larger graphs require more processing and memory than smaller ones. The size of the sweep graph is determined in part by how many spatial elements are in the mesh. The effective size of the graph may be reduced by grouping nearby elements into sets (by means of a partitioning tool), which will be processed simultaneously. One drawback is that larger sets delay communication, thus incurring costs similar to that described in the previous section. In addition, the grouping of elements can degrade the iterative effectiveness of source iteration. The user may use the optional `Element_Set_Size` subsection of the `Solver` section to specify the (approximate) number of elements per set in the sweep graph, with the following syntax:

`<Element_Set_Size>`*`size`*`</Element_Set_Size>`

where *`size`* must be a positive integer, with a default of 1.

2.23.1.1.1.3. Coarse_Sn_Order

As indicated in the previous section, tradeoffs may be made with respect to the size of the sweep graph. The effective size of the graph may be reduced by grouping nearby angles into sets, which will be processed simultaneously, with effects similar to grouping elements. This grouping is performed by mapping the angles in the angular quadrature set to those of a lower order (of the same quadrature rule). The user may use the optional `Coarse_Sn_Order` subsection of the `Solver` section to specify the angular quadrature order for the mapping, with the following syntax:

`<Coarse_Sn_Order>`*`order`*`</Coarse_Sn_Order>`

where *`order`* must be a valid quadrature order, with a default of the same order as the transport calculation.

2.23.1.1.1.4. Aggressive_Cycle_Breaking

The sweep graph may not, in general, be acyclic. This may be inherent to the properties of an unstructured mesh, which can induce cycles ("strongly connected components") in the graph for one or more angles. This also can happen if elements and/or angles are grouped into sets. In order for the sweep process to work it must use an acyclic graph. This may be created from a cyclic graph by "removing" edges from the graph, which has the effect of allowing tasks to be performed by using data generated in a previous iteration. The algorithm used to detect and remove such cycles can either remove one edge at a time from each cycle and then reanalyze the graph (which may take many passes), or it can more aggressively break those cycles by removing a fraction of the edges in a cycle (which may remove more than strictly necessary, thus potentially degrading the iterative effectiveness of the sweep process). The user may use the optional `Aggressive_Cycle_Breaking` subsection of the `Solver` section to indicate whether the aggressive algorithm should be employed, with the following syntax:

`<Aggressive_Cycle_Breaking>[`*`true`* `|` *`false`*`]</Aggressive_Cycle_Breaking>`

with a default of `false`.

2.23.1.1.1.5. Cycle_Fraction

If aggressive cycle breaking is employed, the user may use the `Cycle_Fraction` subsection of the `Solver` section to indicate how aggressive the algorithm should be, with the following syntax:

`<Cycle_Fraction>`*`fraction`*`</Cycle_Fraction>`

where *`fraction`* is a real number between 0 and 1. With higher fractions the cycle detection and breaking algorithm will complete in fewer passes, but the iterative sweep process may require more iterations to converge.

2.23.1.1.1.6. Error_Control_Options

The sweeps-based solver is an iterative solver which needs stopping criteria to reduce the iterative error to an acceptable level. These criteria are provided in the `Error_Control_Options` subsection of the `Solver` section with the following syntax:

```
<Error_Control_Options>
  options
</Error_Control_Options>
```

where *`options`* are specified below.

2.23.1.1.1.6.1. Maximum_Number_Iterations

The user must indicate the maximum number of source iterations that are allowed during the iterative solution of an energy group. The solver may return to a given energy group multiple times during the outer iteration, with the maximum iterations applying each time. This maximum number is indicated in the `Maximum_Number_Iterations` subsection of the `Error_Control_Options` section, with the following syntax:

`<Maximum_Number_Iterations>`*`maxIters`*`</Maximum_Number_Iterations>`

where *`maxIters`* must be a positive integer.

2.23.1.1.1.6.2. Number_Aggregated_Iterations

During source iteration the sweeps-based solver will estimate the iterative error to determine if convergence has been achieved. This estimate may be expensive, and it also may be sensitive to the asynchronous nature of the sweeps process, which may yield an iterative operator that is somewhat different at each iteration. For this reason, it may make sense to perform multiple source iterations before computing the error estimate, both to amortize its expense and to smooth out differences between iterations. This is done with the optional `Number_Aggregated_Iterations` subsection of the `Error_Control_Options` section, with the following syntax:

`<Number_Aggregated_Iterations>`*`iters`*`</Number_Aggregated_Iterations>`

where *iters* must be a positive integer, with a default of 1.

2.23.1.1.1.6.3. Convergence_Tolerance

Source iterations will stop when the estimated error is below a convergence tolerance. This tolerance is specified in the `Convergence_Tolerance` subsection of the `Error_Control_Options` section, with the following syntax:

`<Convergence_Tolerance>`*tolerance*`</Convergence_Tolerance>`

where *tolerance* is a real number between 0 and 1.

2.23.1.1.1.6.4. Error_Metrics

The user may specify one or more error metrics that must all be satisfied for source iteration to terminate prior to exceeding the maximum number of iterations. These are specified in the optional `Error_Metrics` subsection of the `Error_Control_Options` section, with the following syntax:

```
<Error_Metrics>
  metrics
</Error_Metrics>
```

where *metrics* and default settings are specified below.

2.23.1.1.1.6.4.1 Error_Metric

At least one error metric must be specified for the sweeps-based solver unless the default is used (default settings are described in each of the following subsections). This is indicated in an `Error_Metric` subsection in the `Error_Metrics` section, with the following syntax:

```
<Error_Metric>
  options
</Error_Metric>
```

where *options* are specified below.

2.23.1.1.1.6.4.1.1 Metric_Type

There are several types of error metrics (norms) that are available that are applied to angular flux field data. By this we mean, in part, the spatial portion of the problem to which the metric will be applied. Technically speaking, a true mathematical norm would need to be applied to the entire phase space, but we allow for additional metrics that focus on parts of the problem to the exclusion of others to help ensure that individual quantities of interest are converged. This is indicated in an optional `Metric_Type` subsection in the `Error_Metric` section, with the following syntax:

`<Metric_Type>[`*`null | whole | region | surface | leakage`*`]</Metric_Type>`

with a default of `whole`.

The types of metrics available are the following:

1. `Null`

   The `null` metric type simply returns 0 as the measure of field data. This can be useful to exercise the solver once without desiring to compute an error.

2. `Whole`

   The `whole` metric type is applied over the entire spatial domain of the problem.

3. `Region`

   The `region` metric type is applied over just a named Exodus element block. This requires an additional `Region_Name` subsection in the `Error_Metric` section, with the following syntax:

   `<Region_Name>`*`region name`*`</Region_Name>`

   where *`region name`* is the name of an Exodus element block.

4. `Surface`

   The `surface` metric type is applied by integration over a named surface (sideset) of the Exodus mesh, with equal weight given to all angles (in contrast to the leakage metric described below). This requires an additional `Surface_Name` subsection in the `Error_Metric` section, with the following syntax:

   `<Surface_Name>`*`surface name`*`</Surface_Name>`

   where *`surface name`* is the name of an Exodus sideset.

5. `Leakage`

   The `leakage` metric type is applied by integration over a named surface of the Exodus mesh, including only outgoing angles, and applying the inner product of each angle with the outward surface normal of each face. This requires an additional `Surface_Name` subsection in the `Error_Metric` section, with the following syntax:

   `<Surface_Name>`*`surface name`*`</Surface_Name>`

   where *`surface name`* is the name of an Exodus sideset.

2.23.1.1.1.6.4.1.2 Integration_Policy

When applying a metric to angular flux field data, there are two options regarding how to sum the results. In the "discrete" approach, the field data is treated as a vector, with equally weighted sums of the entries. In the "continuous" approach, the actual expansions that the field data represent are used in a numerical integration over space and angle. The latter is more expensive, but it also is the method used to determine physical quantities of interest (e.g., average dose), which is particularly

important with irregular meshes. This choice is indicated in an optional `Integration_Policy` subsection in the `Error_Metric` section, with the following syntax:

`<Integration_Policy>[`*`discrete | continuous`*`]</Integration_Policy>`

with a default of `discrete`.

2.23.1.1.1.6.4.1.3 Error_Norm

There are three types of error norms available: the familiar L-norm, the H-norm (or gradient norm), and the "streaming" norm. The streaming norm is $\Omega \cdot \nabla f$, where Ω is the angle and f is a field value, and is related to certain transport properties. The choice of norm is indicated in an optional `Error_Norm` subsection in the `Error_Metric` section, with the following syntax:

`<Error_Norm>[`*`L | H | S`*`]</Error_Norm>`

with a default of `L`.

2.23.1.1.1.6.4.1.4 Error_Order

There are three orders of error norms that are available: the 1-norm, 2-norm, or infinity-norm. The choice of order is indicated in an optional `Error_Order` subsection in the `Error_Metric` section, with the following syntax:

`<Error_Order>[`*`1 | 2 | I`*`]</Error_Order>`

with a default of `2`.

2.23.1.1.1.6.4.1.5 Sign_Policy

When applying the above norms, we may either take the absolute values of each discrete quantity before summation (which a true norm will do), or we may allow the signs to be preserved, which allows for cancellation of errors but is more directly related to many quantities of interest. This choice is indicated in an optional `Sign_Policy` subsection in the `Error_Metric` section, with the following syntax:

`<Sign_Policy>[`*`absolute | signed`*`]</Sign_Policy>`

with a default of `absolute`.

2.23.1.1.2. First_Order_GMRES

The `First_Order_GMRES` solver form is related to the `First_Order` solver. It too uses a discontinuous-Galerkin discretization of the 1st-order form of the Boltzmann transport equation that is solved via a sweeps-based solver. However, rather than using source iteration, it wraps the sweeps-based solver with a GMRES iteration. Consequently, its solver options are somewhat different than those for the `First_Order` solver. It too uses the syntax for `Work_Per_Message`, `Element_Set_Size`, `Coarse_Sn_Order`, `Aggressive_Cycle_Breaking`, and `Cycle_Fraction`. However, it does not use the

`Error_Control_Options` subsection, which is specific to source iteration. Instead, the GMRES iteration is controlled by several other options, which are indicated within the Solver section as indicated below.

2.23.1.1.2.1. Max_Restart

GMRES is an iterative Krylov algorithm, whose computational cost and memory grows with the number of iterations applied. One may "restart" the algorithm after some number of iterations by using the current iterative solution vector as the starting guess for a new, or restarted, GMRES process. The user may wish to specify the maximum number of restarts allowed. This is done with the optional `Max_Restart` subsection of the Solver section, with the following syntax:

`<Max_Restart>`*`maxRestarts`*`</Max_Restart>`

where *`maxRestarts`* must be a positive integer, with a default of 1.

2.23.1.1.2.2. Subspace_Parameter

As noted in the previous section, GMRES may be restarted after a set number of iterations. This is done with the optional `Subspace_Parameter` subsection of the Solver section, with the following syntax:

`<Subspace_Parameter>`*`iters`*`</Subspace_Parameter>`

where *`iters`* must be a positive integer, with a default of 20.

2.23.1.1.2.3. Convergence_Tolerance

GMRES iterations will stop when the estimated error is below a convergence tolerance. This tolerance is specified in the `Convergence_Tolerance` subsection of the `Solver` section, with the following syntax:

`<Convergence_Tolerance>`*`tolerance`*`</Convergence_Tolerance>`

where *`tolerance`* is a real number between 0 and 1, with a default of 1e-5.

2.23.1.1.2.4. Check_Solution_Convergence

There are multiple loops in the GMRES algorithm, with certain checks for convergence. An additional check may be performed with the optional `Check_Solution_Convergence` subsection of the `Solver` section, with the following syntax:

`<Check_Solution_Convergence>[`*`true`* `|`
`false``]</Check_Solution_Convergence>`

with a default of `false`.

2.23.1.1.2.5. Verbosity

SCEPTRE will output some information concerning the GMRES calculation to `stdout` as it proceeds. The amount of information can be controlled with the optional `Verbosity` subsection within the `Solver` section, as follows:

`<Verbosity>[high | medium | none]</Verbosity>`

with a default of `medium`. Note that this GMRES verbosity setting is independent of the global verbosity setting described above.

2.23.1.1.2.6. Padding

When messages are written to `stdout` during the GMRES process, the user may wish to specify indentations. This is controlled with the optional `Padding` subsection within the `Solver` section, as follows:

`<Padding>`*`padding`*`</Padding>`

where *`padding`* is a string, with a default of four spaces.

2.23.1.1.2.7. Initial_Guess

When GMRES is used, it needs an initial guess to the solution. This is controlled by the optional `Initial_Guess` subsection within the `Solver` section, as follows:

```
<Initial_Guess>["zero_vector" |
"solution_vector"]</Initial_Guess>
```

with a default of `solution_vector`. If `zero_vector` is chosen, the initial guess will be all zeroes. If `solution_vector` is chosen, the algorithm will use the solution to an uncollided flux problem as the starting guess.

2.23.1.1.2.8. Number_Source_Iterations

After the GMRES iterations have completed, the user may wish to perform additional source iterations to further converge the solution. This is done with the optional `Number_Source_Iterations` subsection of the Solver section, with the following syntax:

`<Number_Source_Iterations>`*`iters`*`</Number_Source_Iterations>`

where *`iters`* must be a nonnegative integer, with a default of 0.

2.23.1.1.2.9. Solve_Angular_Moments

The fundamental quantity of interest of a transport solver is the angular flux. An equivalent form is the angular (spherical harmonic) moments. The GMRES algorithm can use either of these quantities as the vector it tries to converge. This is controlled with the optional `Solve_Angular_Moments` subsection of the `Solver` section, with the following syntax:

<Solve_Angular_Moments>[*true* | *false*]</Solve_Angular_Moments>

with a default of `false` (indicating that the angular flux vector will be used).

2.24. Solver_Assignment

Once one or more solvers have been defined in the `Solvers` section, it is necessary to assign them to energy groups. This is indicated in the `Solver_Assignment` section, with the following syntax:

<Solver_Assignment explicit = "true">

assignments

</Solver_Assignment>

where *assignments* are specified below.

2.24.1. Solver_By_Group

Each group may be individually assigned a solver. This is indicated in the optional `Solver_By_Group` subsection of the `Solver_Assignment` section, with the following syntax:

<Solver_By_Group>

group assignments

</Solver_By_Group>

where *group assignments* are specified below.

2.24.1.1. Group

An individual group assignment is performed in the `Group` subsection of the `Solver_By_Group` section, with the following syntax:

<Group index="*group ID*">*solver name*</Group>

where *group ID* is the (1-based) energy group and *solver name* is the name of a solver defined in the `Solver` section above.

2.24.2. Solver_By_Group_Range

Multiple groups may be assigned the same solver at once if they are in a contiguous range. This is indicated in the optional `Solver_By_Group_Range` subsection of the `Solver_Assignment` section, with the following syntax:

<Solver_By_Group_Range>*name ID1 ID2*</Solver_By_Group_Range>

where *name* is the solver name, *ID1* is the first group in the range, and *ID2* is the last group in the range.

2.24.3. Solver_By_All_Groups

If all groups are to be assigned the same solver, this may be specified in the optional Solver_By_All_Groups subsection of the Solver_Assignment section, with the following syntax:

<Solver_By_All_Groups>*solver name*</Solver_By_All_Groups>

where *solver name* is the solver name.

REFERENCES

[1] Pautz, S., Bohnhoff, W., Drumm, C., & Fan, W. (2009). Parallel Discrete Ordinates Methods in the SCEPTRE Project. *International Conference on Mathematics, Computational Methods & Reactor Physics (M&C 2009)*. Saratoga Springs, New York: American Nuclear Society.

DISTRIBUTION

Email—Internal

Name	Org.	Sandia Email Address
Technical Library	1911	sanddocs@sandia.gov
Jarrod Edwards	1341	jdedwa@sandia.gov
Duncan McGregor	1351	damcgre@sandia.gov
Michael Krygier	1421	mkrygie@sandia.gov
Jason Sanchez	1443	jassanc@sandia.gov
Cam McCormick	1543	camccor@sandia.gov
David Ching	8751	dching@sandia.gov
Ryan Keedy	8755	rkeedy@sandia.gov

This page left blank

This page left blank

Sandia National Laboratories

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.