SANDIA REPORT

SAND2017-12576
Unlimited Release
Printed November 15, 2017

EMPHASIS^{TM}/Nevada UTDEM User Guide Version 2.1.2

C. David Turner, Michael F. Pasik, David B. Seidel, Timothy D. Pointon, Keith L. Cartwright, Richard M. Kramer, Duncan A. McGregor

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

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and 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.

Sandia National Laboratories

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

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@adonis.osti.gov
Online ordering:	http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone:	(800) 553-6847
Facsimile:	(703) 605-6900
E-Mail:	orders@ntis.fedworld.gov
Online ordering:	http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2017-12576
Unlimited Release
Printed November 15, 2017

EMPHASIS$^{\mathrm{TM}}$/Nevada UTDEM User Guide Version 2.1.2

C. David Turner, Michael F. Pasik, David B. Seidel,
Timothy D. Pointon, Keith L. Cartwright, Richard M. Kramer
Electromagnetic Theory

Duncan A. McGregor
Computational Multiphysics

Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185

Abstract

The Unstructured Time-Domain ElectroMagnetics (UTDEM) portion of the EMPHASIS suite solves Maxwell's equations using finite-element techniques on unstructured meshes. This document provides user-specific information to facilitate the use of the code for applications of interest.

Acknowledgment

The authors would like to thank all of those individuals who have helped to bring EMPHASIS/Nevada to the point it is today, including Bill Bohnhoff, Rich Drake, and all of the NEVADA code team.

Contents

Appendix

List of Figures

List of Tables

EMPHASIS UTDEM User Guide

1 Introduction

EMPHASIS/Nevada Unstructured Time-Domain ElectroMagnetics (UTDEM) is a general-purpose code for solving Maxwell's equations on arbitrary, unstructured tetrahedral meshes. The geometries and the meshes thereof are limited only by the patience of the user in meshing and by the available computing resources for the solution. UTDEM solves Maxwell's equations using finite-element method (FEM) techniques on tetrahedral elements using vector, edge-conforming basis functions [14].

EMPHASIS/Nevada Unstructured Time-Domain ElectroMagnetic Particle-In-Cell (UTDEM PIC) is a superset of the capabilities found in UTDEM. It adds the capability to simulate systems in which the effects of free charge are important and need to be treated in a self-consistent manner. This is done by integrating the equations of motion for macroparticles (a macroparticle is an object that represents a large number of real physical particles, all with the same position and momentum) being accelerated by the electromagnetic forces upon the particle (Lorentz force). The motion of these particles results in a current, which is a source for the fields in Maxwell's equations.

1.1 UTDEM Simulation Process

The UTDEM simulation process is shown in Fig. 1. The geometry of interest must first be modeled and meshed using appropriate software. I-DEAS is one option that provides both, but successful meshes have been generated using CUBIT and ICEM CFD Tetra as well. Regardless of how the mesh is generated it must ultimately be written or converted to EXODUSII [9] format. CUBIT and ICEM do this directly as does I-DEAS with the addition of a special 3$^{\mathrm{rd}}$ party feature. However, certain features of UTDEM require sets of nodes (nodesets), edges (edgesets), or faces (sidesets) to be defined. In particular, edgesets are not a part of the EXODUSII standard. The creation of these "edgesets" along with nodesets and sidesets are handled along with the conversion from I-DEAS universal format to EXODUSII format by the preprocessing step PREP [5]. Use of virtual edgesets (described later) avoids the requirement to process edgesets through PREP. Further discussion of the use of PREP is found in Appendix A.

The actual input required for UTDEM consists of only two files, the Genesis file containing the mesh and the problem-specific input file. "Genesis" normally describes a mesh description file in EXODUSII format containing mesh only, no data.

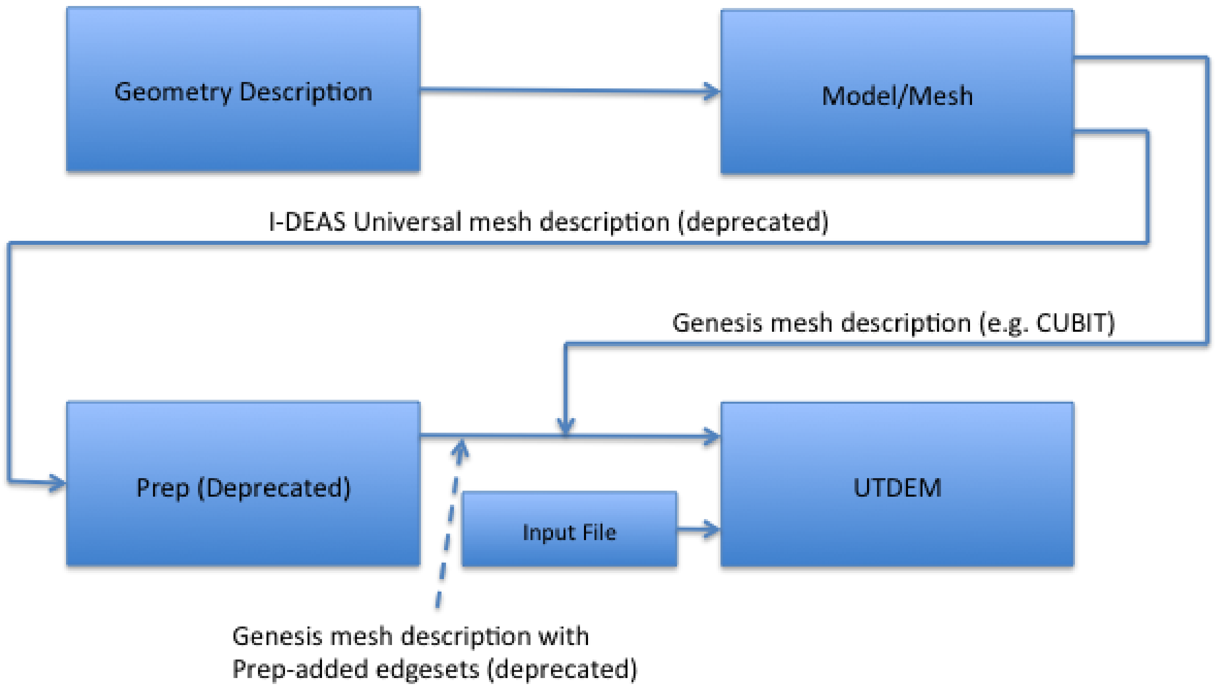

Figure 1. The UTDEM simulation process.

A critical aspect of the mesh for UTDEM is that boundary conditions are described by EXODUSII sidesets, edgesets, or nodesets. These requirements will be described later as each feature is covered in detail. The tetrahedral elements should also be reasonably well shaped.

UTDEM results are available in the form of observer time histories and plot dumps of specified variables in an EXODUSII file. These results can be displayed with most any simple plotting package in the case of time histories and by post-processing tools which can import EXODUSII, such as EnSight or ParaView, in the case of plot dumps.

1.2 Units

All UTDEM simulations are performed in MKS (SI) units. As such, all input file values must be scaled accordingly by the user. This is especially important for `BOX IEMP` simulations since radiation-transport codes generally favor CGS units. Some exceptions are made for certain control inputs, where different units are used for various reasons; these are noted throughout this manual when encountered.

2 UTDEM Input File and Keywords

The UTDEM input file uses the standard NEVADA input file format, which includes keywords for debugging, physics type, solver control, output control, and more. Details of all of these except for the specific physics can be found in the ALEGRA user guide [3]. A complete input file for one of the UTDEM regression problems is given in Appendix B.

```
UNSTRUCTURED TD ELECTROMAGNETICS
  formulation, second order
  aztec set, 0
  abc bc, sideset 4
  pec bc, sideset 2
  observer, nodeset 28
  observer, nodeset 29
  source, nodeset 31, function 1
  slot observer, nodeset 19
  slot, edgeset 123, aztec_set 1, width 0.00001, depth 0.0, int_mat 1, ext_mat 2
  slot observer, nodeset 20
  slot, edgeset 124, aztec_set 2, width 0.00005, depth 0.0, int_mat 1, ext_mat 2
END
```

Figure 2. Typical UTDEM physics keywords.

The format for specifying UTDEM physics and associated keywords is shown in Fig. 2. The physics keyword `UNSTRUCTURED TD ELECTROMAGNETICS` specifies to the Nevada framework that the UTDEM physics model should be used for the simulation. Most of the remaining keywords are specific to UTDEM and are described below along with many others. Case is ignored in the input file. It is important to note that lines are limited to 160 characters or less. Keywords can be abbreviated to stay below this limit. Additionally, when a floating-point value is required, the decimal point needs to be included (i.e., `0.` not simply `0`).

2.1 Solver formulation keyword

`FORMULATION, type`

Specifies the UTDEM solver formulation for this simulation. Options for `type` are:

`SECOND ORDER`

Utilize the unconditionally stable, 2^{nd} order electric-field wave-equation formulation. This formulation works with all physical models implemented in the code and is recommended for most applications.

`SECOND ORDER ARTUZI`

Utilize the `SECOND ORDER` formulation above with Artuzi late-time stability correction. This formulation can suppress late-time drift due to the use of very large time steps. Note that it causes all results to be *advanced* in time by 1/2 time step.

`SECOND ORDER ARTUZI, STATIC LIMIT`

Utilize the `SECOND ORDER` formulation above with Artuzi late-time stability correction and the Artuzi rhs term $[\mathrm{S}]\mathrm{w}^{\mathrm{n}}$ disabled. This should only be applied with caution and only when the problem is essentially static, where element edge lengths are much, much smaller than the wavelength. Although there is no theoretical justification for this modification, it appears to work extremely well in the static limit.

`SECOND ORDER FRIEDMAN, THETA real`

Utilize the `SECOND ORDER` formulation, but with unconditional stability achieved using the Friedman implicit method with adjustable damping [7], instead of the Newmark-Beta method used for standard `SECOND ORDER`. The damping parameter θ (required) should be in the range $0 \leq \theta \leq 1$, where 0 is no damping and 1 is the maximum damping. This solver is intended to be used only with PIC simulations to damp high frequency ($\omega \sim 1/\Delta t$) noise from particle fluctuations. Our experience is that this method can very effectively damp noise as $\theta \to 1$, but occasionally results in numerical instability in large production simulations. The reason for this has not been determined, but is clearly problem-specific. Using $\theta = 0.25$ seems to be a practical safe value, although experimenting with values up to $\theta = 1$ may be successful for further reduction of particle noise. Note that PIC simulations have an additional constraint on the timestep. They must resolve the highest frequency plasma oscillations in the problem, $\omega_p \Delta t \lesssim 1$, where $\omega_p = \sqrt{ne^2/\epsilon_0}$, and n is the plasma density in C/m^3.

`CRANK NICOLSON`

Utilize the `CRANK NICOLSON` time integrator for the first-order form of Maxwell's equations. This formulation is unconditionally stable like the second-order formulation, and can be shown to be algebraically equivalent. It is required for use with PML boundaries and nonlinear material models, and is compatible with PIC. Note that some functionality is not available with Crank-Nicolson.

`FIRST ORDER`

Utilize the conditionally stable, coupled 1$^{\mathrm{st}}$ order formulation (not fully implemented).

For additional information on these options consult the UTDEM theory guide [14].

`CONDITION NUMBER`

Specifies that the system matrix condition number should be displayed whenever the matrix is refilled.

2.2 FEM basis order keyword

`BASIS ORDER, int`

Specifies the order of the FEM basis or shape functions. Available values are:

`BASIS ORDER, 0`

Default if no keyword specified

The default basis uses the Whitney tangential vector finite element with one degree of freedom per edge. This is occasionally referred to as a 0^{th} order or 1^{st} order mixed element. These shape functions are constant along an edge and linear across the element. In reality, they behave more like 0^{th} than 1^{st} order as the spatial convergence is observed to be nearly linear.

`BASIS ORDER, 1`

This option adds one additional degree of freedom per edge by, for tetrahedrons, adding six additional hierarchical shape functions to the original six Whitney functions. This 1^{st} order full element achieves full 2^{nd} order or quadratic spatial convergence with the penalty of doubling the number of unknowns in the FE system.

This optional basis is presently only available for tetrahedral elements, precluding its use with hybrid meshing. It has been successfully tested with all algorithms in the code except for the unconditionally stable wire.

2.3 Solver option keywords for PIC

`GODFREY ALPHA1, real`

Specifies the coefficient of the forward time in the field solve. With `FORMULATION SECOND ORDER`, one can specify `GODFREY ALPHA1` ≥ 0.25. The default is 0.25 and yields the Newmark-Beta time advance [8]. Practical values of `GODFREY ALPHA1` are between 0.25 and 1.0; at a value of 1 it is the same algorithm as Friedman with a theta value of 2. Note that [8] analyzes a "Friedman" solver in detail, but this is a related explicit first-order algorithm, not the one enabled in Emphasis with the `SECOND ORDER FRIEDMAN` keyword.

2.4 Boundary condition keywords

`PEC BC, SIDESET int`

Specifies a perfect electric conductor boundary condition.

The electric fields tangential to the surface described by the sideset having id `int` will be set to zero. This sideset id must exist in the genesis file or Nevada will complain. Multiple pec bc's may exist in the same simulation.

As with all sidesets, multiple disjoint surfaces may exist in the same sideset. PEC surfaces may be grouped into different sidesets for purposes of visualization.

`PMC BC, SIDESET int`

Specifies a perfect magnetic conductor or mirror-symmetry boundary condition.

The pmc condition is the natural boundary condition for the edge-conforming FEM formulation and could simply be left "free". However, for PIC simulations or any simulation containing h-line integral observers, the user *must* specify a sideset defining the pmc using this keyword. PIC simulations requesting the `PIC_CURRENT` plot variable must have multiple pmcs specified using separate side sets.

The sideset id `int` must exist in the genesis file or Nevada will complain. Multiple pmc bc's may exist in the same simulation and PMC surfaces may be grouped into different sidesets for purposes of visualization.

`IBC BC, SIDESET int, IMPEDANCE real`

Specifies an impedance boundary condition.

A 1^{st} order surface impedance boundary condition with surface impedance value `real` will be applied over the surface described by the sideset having id `int`. Presently, this "impedance" must be real, a surface resistance only. This sideset id must exist in the genesis file or Nevada will complain. Multiple IBC bc's may exist in the same simulation.

`ABC BC, SIDESET int`

Specifies an absorbing boundary condition.

A 1^{st} order absorbing boundary condition will be applied over the surface described by the sideset having id `int`. This sideset id must exist in the genesis file or Nevada will complain. Multiple abc bc's may exist in the same simulation.

An absorbing boundary condition is typically specified in conjunction with a port boundary condition when the port is located on an exterior surface.

Boundary condition precedence is presently specified in the code as follows: PEC $\rightarrow$ IBC $\rightarrow$ ABC, i.e., an edge specified as both PEC and IBC or ABC will be PEC. An edge specified as both IBC and ABC will be IBC.

To specify Perfectly Matched Layer (PML) boundaries to truncate a domain, see Chapter 8 for details. These are specified using a special material model in the block that defines the layer, and do not use a boundary condition keyword.

2.5 Virtual edgeset keyword

`PATH, EDGESET int,`
`POINT, X=real Y=real Z=real POINT, X=real Y=real Z=real POINT, ...`

Specifies a virtual edgeset to be created along an arbitrary user-defined path through the geometry, as described by a sequence of `POINT` vectors, which is independent of mesh topology.

The best-fitting path of existing edges to match the desired line segment(s) entered will be found. The user is responsible for picking a unique `EDGESET` id. If an id is chosen which exists in the Genesis file, a warning is issued and the virtual edgeset is not used. If a specified segment endpoint cannot be reached within the default tolerance (half the local minimum edge length) a warning will be issued giving the actual endpoint used.

The order of the edges (and nodes) in the edgeset is determined by the order in which the `POINT`'s are supplied. This is useful for the integration direction around an `H LINE INTEGRAL`.

These virtual edgesets may be used anywhere a mesh-defined edgeset can be used. The `EXODUS EDGE SETS` keyword should not be applied to these edgesets.

2.6 Observer keywords

`OBSERVER, NODESET int`

Specifies the simplest type of single-edge, electric-field-projection observer.

The edge will be located which is defined by the nodeset having id `int`. This nodeset MUST contain only 2 nodes. Presently, UTDEM will not issue an error if it does not, the observer will simply not be created. This will be fixed in a future version. Multiple observers of this type may exist in the same simulation.

The observer time history will be written to a file with a generated name in the following format: *problem_name*`.obs`*nodeset_id*`.`*proc_id*`.dat`, where *nodeset_id* is the requested nodeset id `int`, and *proc_id* will be 0 for serial or the appropriate processor number for parallel. For parallel, each processor who has the observer is synchronized to the correct value from the owner and writes a separate observer time-history file. All of these files are identical at the end of the simulation.

The observer time history will be written to the hisplt database *problem_name*`.his` with the name: `OBS-`*nodeset_id*.

`SLOT VOLTAGE OBSERVER, NODESET int, [DIRECTION X real Y real Z real]`

Specifies a slot voltage observer.

The slot voltage at the node defined by the nodeset having id `int` will be monitored. This nodeset MUST contain only 1 node. An error will be issued if it contains more than 1 node. If the nodeset contains 0 nodes, the observer will simply not be created. Multiple slot observers may exist in the same simulation.

The assumed positive direction of the voltage (actually "magnetic current") can be specified by the optional parameter `DIRECTION`. The direction here is that *along* the slot, normal to the slot "gap" where the voltage would be measured across. If not specified, the natural direction (code internally assumed direction) will be output. This can cause inversion of the results in parallel depending on the number of processors. Consequently, it is recommended that the direction be specified. If the specified direction is not very close to the assumed direction, a warning is issued.

The observer time history will be written to a file with a generated name in the following format: *problem_name*.`slotobs`*nodeset_id*.*proc_id*.`dat`, where *nodeset_id* is the requested nodeset id `int`, and *proc_id* will be 0 for serial or the appropriate processor number for parallel. For parallel, each processor who has the observer is synchronized to the correct value from the owner and writes a separate observer time-history file. All of these files are identical at the end of the simulation.

The observer time history will be written to the hisplt database *problem_name*.`his` with the name: `SLOTVOLOBS-`*nodeset_id*.

`WIRE CURRENT OBSERVER, NODESET int, [DIRECTION X real Y real Z real]`
Specifies the wire current observer.

The wire current at the node defined by the nodeset having id `int` will be monitored. This nodeset MUST contain only 1 node. An error will be issued if it contains more than 1 node. If the nodeset contains 0 nodes, the observer will simply not be created. Multiple wire observers may exist in the same simulation.

The assumed positive direction of the current can be specified by the optional parameter `DIRECTION`. If not specified, the natural direction (code internally assumed direction) will be output. This can cause inversion of the results in parallel depending on the number of processors. Consequently, it is recommended that the direction be specified. If the specified direction is not very close to the assumed direction, a warning is issued.

The observer time history will be written to a file with a generated name in the following format: *problem_name*.`wireobs`*nodeset_id*.*proc_id*.`dat`, where *nodeset_id* is the requested nodeset id `int`, and *proc_id* will be 0 for serial or the appropriate processor number for parallel. For parallel, each processor who has the observer is synchronized to the correct value from the owner and writes a separate observer time-history file. All of these files are identical at the end of the simulation.

The observer time history will be written to the hisplt database *problem_name*.`his` with the name: `WIRECUROBS-`*nodeset_id*.

`E LINE INTEGRAL, EDGESET int, [DIRECTION X real Y real Z real]`
Specifies an electric-field line-integral observer, $\int \mathbf{E} \cdot dl$.

The electric-field projections on the edges in the edgeset are simply summed. The vector `DIRECTION` is used to determine the direction of the integral and therefore determines the signs of all the individual edge contributions to the integral by taking the dot product of the vector edge direction with the `DIRECTION` vector. Generally, the mesh should be designed such that the edges are aligned in the correct direction, but this is not required. Multiple observers of this type may exist in the same simulation.

The observer time history will be written to a file with a generated name in the following format: *problem_name*.`elinint`*edgeset_id*.*proc_id*.`dat`, where *edgeset_id* is the requested edgeset id `int`, and *proc_id* will be 0 for serial or the appropriate processor number for parallel. For parallel, each processor who has a portion of the observer is synchronized to the correct value from the owner and writes a separate observer time-history file. All of these files are identical at the end of the simulation.

The observer time history will be written to the hisplt database *problem_name*`.his` with the name: `EDL-`*edgeset_id*.

`H LINE INTEGRAL, EDGESET int`

Specifies a magnetic-field line-integral observer, $\oint \mathbf{H} \cdot dl$.

The integration direction around the loop is determined by the order of the edges (and nodes) in the edgeset. These edgesets are typically generated using the `PATH` keyword, which generates the edgeset edges (and nodes) in the order in which the `POINT`'s are supplied. An average magnetic field is computed for each edge in the edgeset by computing the magnetic field at the center of each element connected to the edge. This vector magnetic field is then dotted with the edge direction and the "sense" of the edge, as determined by the edgeset edge (and node) ordering. These dot products on the edges in the edgeset are then summed. Multiple observers of this type may exist in the same simulation.

In the case of legacy edgesets created before the `PATH` command existed, the edge "sense" is determined from the beam elements in the original mesh defining the edgeset, specified using the `EXODUS EDGE SETS` keyword. This usage is deprecated with the convenience of the `PATH` command.

The observer time history will be written to a file with a generated name in the following format: *problem_name*`.hlinint`*edgeset_id*`.`*proc_id*`.dat`, where *edgeset_id* is the requested edgeset id `int`, and *proc_id* will be 0 for serial or the appropriate processor number for parallel. For parallel, each processor who has a portion of the observer is synchronized to the correct value from the owner and writes a separate observer time-history file. All of these files are identical at the end of the simulation.

The observer time history will be written to the hisplt database *problem_name*`.his` with the name: `HDL-`*edgeset_id*.

Note that if the simulation contains h-line integrals along with PMC symmetry boundaries, the PMC's must be defined by sidesets using the `PMC BC` keyword.

`SURFACE CURRENT, SIDESET int [int ...]`

Enables the calculation of a surface current, $\mathbf{n} \times \mathbf{H}$, on the nodes of the specified conductor boundary sideset(s).

The surface current is computed from the magnetic field which itself is derived from the electric field. As a result, the magnetic field value used in the surface current calculation is constant over each element. An average normal is computed at each node from the normal of the element faces that contain the sideset node. To enable output of the computed surface current values, be sure to include `SURFACE_CURRENT_DEN` on the list of requested plot variables. Because some post-processing tools may have problems with the length of the variable name the following is suggested:

```
plot variable
```

```
    surface_current_den, as "js"
  end
```

The surface current will be output at all nodes in the simulation but will only have non-zero values on the specified sidesets.

`MAX OBSERVER, ID int, NAME variable_name [BLOCK int int ...]`

Specifies a maximum-field (or other registered variable) observer, where the id `INT` is the user-defined identifier and `NAME` is the registered variable to be monitored such as `ELECTRIC_FIELD`. The `block` list is optional and is used to specify which element blocks are to be monitored. If the name or block listed is invalid then the code will issue an error. If `block` is not supplied then all blocks are used. Multiple max observers can exist for the same variable to monitor different sets of element blocks.

Results are displayed each cycle to the screen and sent to the HISPLT time-history file at the `EMIT SCREEN` frequency. If the variable is a vector, both the maximum vector (magnitude) is monitored along with the individual maxima of each component. For a scalar variable only a single value is monitored and reported. For screen results, in the case that a particular variable is identically zero for that time cycle it is reported as `ELECTRIC_FIELD == 0`.

Typical screen results for two scalar fields and one vector field are:

```
MAX-5: ELECTRIC_FIELD_MAGNITUDE = 1.4325e+00 in Elem 13599
       at (7.8396e+00,7.3799e-01,4.2033e-01)
MAX-6: ELECTRIC_FIELD_PROJECTION = 7.6761e-01 on Edge w/Nodes 10 - 347
       at (7.8750e+00,6.2500e-01,1.6250e+00)
MAX-7: ELECTRIC_FIELD = (-1.8644e-01,3.9830e-01,1.3633e+00) in Elem 13599
       at (7.8396e+00,7.3799e-01,4.2033e-01)
MAX-7: ELECTRIC_FIELD-X = -1.0047e+00 in Elem 8588
       at (7.5214e+00,7.5055e-01,1.6395e+00)
MAX-7: ELECTRIC_FIELD-Y = 1.0235e+00 in Elem 13902
       at (7.7500e+00,1.0000e+00,1.6250e+00)
MAX-7: ELECTRIC_FIELD-Z = 1.3633e+00 in Elem 13599
       at (7.8396e+00,7.3799e-01,4.2033e-01)
```

HISPLT names for the same variables are:

```
MAX-5
MAX-6
MAX-7-X
MAX-7-Y
MAX-7-Z
MAX-7-C-X
MAX-7-C-Y
MAX-7-C-Z
```

2.7 Source keywords

SOURCE

The source keyword may be used in three different modes depending on the type of boundary set (nodeset, edgeset, or sideset) specified along with it.

1. The following syntax specifies the simplest type of single-edge, electric-field-projection Dirichlet source:

   `SOURCE, NODESET int, FUNCTION int, SCALE real SHIFT real`

   The edge will be located which is defined by the nodeset having id `int`. This nodeset MUST contain only 2 nodes. Presently, UTDEM will issue an error if it has >2 nodes. If it has 0 or 1 nodes, possibly due to parallel decomposition, the source will simply not be created. Multiple sources of this type may exist in the same simulation.

   The keyword `FUNCTION int` specifies a source time history defined by a Nevada keyword function definition described later. The function is scaled by `SCALE` and is shifted in time by `SHIFT`.

   The source time history will be written to a file with a generated name in the following format: *problem_name*`.src`*nodeset_id.proc_id*`.dat`, where *nodeset_id* is the nodeset id parameter from the `SOURCE` line and *proc_id* will be 0 for serial or the appropriate processor number for parallel. For parallel, each processor who has the source is synchronized to the correct value from the owner and writes a separate source time-history file. All of these files are identical at the end of the simulation.

   The source time history will be written to the hisplt database *problem_name*`.his` with the name: `SRC-`*nodeset_id*.

2. The following syntax specifies the next simplest electric-field-projection Dirichlet source–a multi-edge, linear, distributed source:

   `SOURCE, EDGESET int, FUNCTION int, SCALE real SHIFT real`

   `, DIRECTION, X real Y real Z real, LENGTH real`

   The edges will be located which are defined by the edgeset having id `int`. The source will be applied polarized along the direction vector `DIRECTION` ($\mathbf{d}$) and scaled to provide the desired magnitude when integrated over the length `LENGTH` (ℓ). The scale factor for each edge in the source is $\mathbf{l} \cdot \mathbf{d}/\ell$, where $\mathbf{l}$ is the vector edge length (not normalized). For this reason, this source makes sense only if the edges are linear and along a Cartesian axis. Multiple sources of this type may exist in the same simulation.

   The source time history will be written to a file with a generated name in the following format: *problem_name*`.linsrc`*edgeset_id.proc_id*`.dat`, where *edgeset_id* is the edgeset id parameter from the `SOURCE` line and *proc_id* will be 0 for serial or the appropriate processor number for parallel. For parallel, each processor who has the source is synchronized to the correct value from the owner and writes a separate source time-history file. All of these files are identical at the end of the simulation.

The source time history will be written to the hisplt database *problem_ name*`.his` with the name: `LINEARSRC-`*edgeset_ id*.

3. The following syntax specifies the final type of electric-field-projection Dirichlet source–a multi-edge, belt-distributed source:

   ```
   SOURCE, SIDESET int, FUNCTION int, SCALE real SHIFT real
    , DIRECTION, X real Y real Z real, LENGTH real
   ```

   This source is applied over a surface and can, for example, be used to create a delta-gap source on a coax center conductor. The edges will be located which are defined by the sideset having id `int`. The source will be applied polarized along the direction vector `DIRECTION` ($\mathbf{d}$) and scaled to provide the desired magnitude when integrated over the length defined by `LENGTH` (ℓ). The scale factor for each edge in the source is $\mathbf{l} \cdot \mathbf{d}/\ell$, where $\mathbf{l}$ is the vector edge length (not normalized). Multiple sources of this type may exist in the same simulation.

   The source time history will be written to a file with a generated name in the following format: *problem_ name*`.dstsrc`*sideset_ id.proc_ id*`.dat`, where *sideset_ id* is the `sideset` id parameter from the `source` line and *proc_ id* will be 0 for serial or the appropriate processor number for parallel. For parallel, each processor who has the source is synchronized to the correct value from the owner and writes a separate source time-history file. All of these files are identical at the end of the simulation.

   The source time history will be written to the hisplt database *problem_ name*`.his` with the name: `DISTSRC-`*sideset_ id*.

`PORT SOURCE`

A port source is specified with one of the following commands:

```
PORT SOURCE, COAXIAL, SIDESET int, FUNCTION int, SCALE real SHIFT real
 , CENTER, X real Y real Z real
```

```
PORT SOURCE, PARALLEL PLATE, SIDESET int, FUNCTION int, SCALE real
 SHIFT real, DIRECTION, X real Y real Z real, SEPAR real
```

```
PORT SOURCE, FIELD DIST, SIDESET int, FUNCTION int, SCALE real SHIFT real
```

Port sources are soft (non-Dirichlet) sources used to drive specific conductor configurations. The simplest are the coax and the parallel plate, which have been implemented. More complicated are rectangular or circular waveguide port sources, which require fourier transforms. These have not yet been implemented. Multiple port sources may exist in the same simulation. When a port source is applied to an external surface, an absorbing boundary condition is also typically applied to the same sideset.

Other than the normal waveform description, the coaxial port requires only the spatial `CENTER` be specified. The TEM excitation is applied over the specified sideset with $\mathbf{E}$ polarized in the radial direction and the proper variation of

$$\frac{1}{r\ln(b/a)}.$$

Here, a is the inner radius and b is the outer radius, which are determined by the code from the sideset information.

The parallel-plate port requires, in addition to the waveform description, an E-polarization `DIRECTION` and a plate separation `SEPAR`. The TEM excitation is applied over the specified sideset with polarization `DIRECTION` and magnitude `SCALE`/`SEPAR` such that the voltage across the plates is `SCALE`.

The field-distribution port obtains the port field from the sideset distribution factors in the original 3D genesis file. For a description of how to obtain this distribution, see section 10 on inlet-port Poisson solutions.

The source time history will be written to a file with a generated name in the following format: *problem_name*`.prtsrc`*sideset_id.proc_id*`.dat`, where *sideset_id* is the `sideset` id parameter from the `port source` line and *proc_id* will be 0 for serial or the appropriate processor number for parallel. For parallel, each processor who has the source is synchronized to the correct value from the owner and writes a separate source time-history file. All of these files are identical at the end of the simulation.

The source time history will be written to the hisplt database *problem_name*`.his` with the name: `COAXPORTSRC-`*sideset_id* or `PPLTPORTSRC-`*sideset_id*.

`WIRE VOLTAGE SOURCE`

Specifies a voltage source applied on a single wire edge:

```
WIRE VOLTAGE SOURCE, EDGESET int, FUNCTION int, SCALE real SHIFT real,
 DIRECTION, X real Y real Z real
```

The source is applied "in series" with this edge, along the wire. If a WIRE LOAD is present on this edge, the two are in series on the edge. The polarity of the applied voltage will be determined by the DIRECTION parameter which should be reasonably close to the edge direction.

The edgeset must contain only 1 edge otherwise an error will be issued.

The source time history will be written to a file with a generated name in the following format: *problem_name*`.wiresrc`*edgeset_id.proc_id*`.dat`, where *edgeset_id* is the EDGESET parameter from the `WIRE VOLTAGE SOURCE` line and *proc_id* will be 0 for serial or the appropriate processor number for parallel. For parallel, the owning processor writes the source time-history file.

The source time history will also be written to the hisplt database *problem_ name*.his with the name: WIREVOLSRC-*edgeset_ id*.

J SOURCE, BLOCK int [int int ...], [options]

Specifies a volumetric vector current-density source. Options available are:

FUNCTION int, SCALE real SHIFT real

DIRECTION, X real, Y real, Z real

CENTER, X real, Y real, Z real

ATTEN real

PROP bool | string

PROPDIR X real, Y real, Z real

RADTRANS bool | string

ANNULUS bool | string

ORIGIN X real, Y real, Z real

If FUNCTION and DIRECTION are specified, the current density will be applied at all *nodes* in one or more mesh BLOCK(s), flowing in the direction DIRECTION, having the time history specified by FUNCTION.

If ATTEN is nonzero, (default is zero) an exponential attenuation is applied across the mesh according to $\exp(-\texttt{ATTEN} * z)$, where z is the normal distance from the source plane to the node at which the current is required. The attenuation is in the PROPDIR direction, so PROPDIR and CENTER must be provided.

If PROP is true or yes, (default is no) the source will emanate from a plane passing through the point CENTER traveling in direction PROPDIR. The source plane must be outside and behind the mesh volume with respect to the propagation direction, otherwise the code will issue a fatal error. If PROP is false or no, the source will simply follow the specified time history simultaneously throughout the mesh block(s). If PROP is yes, then CENTER and PROPDIR must be defined.

If RADTRANS is true or yes, (default is no) then the current direction will be defined by data imported from a radiation-transport code such as CEPTRE. This data is written to the simulation genesis file by the transport code.

If ANNULUS is true or yes, (default is no) then a phi-directed current will be generated about the axis defined by DIRECTION. An ORIGIN for the annulus coordinate system must be provided which lies on the axis defined by DIRECTION.

The source time history will be written to the hisplt database *problem_ name*.his with the name: JSRC-1001. If a FUNCTION is specified, that time history is written but is not related to the time history of the externally defined source. The correct external-source time history can be written to the *problem_ name*.exo file by requesting "CUR_DEN" under plot variables.

`EXTERNAL J SOURCE, BLOCK int [int int ...], ext-j-spec`

Two variations of `ext-j-spec` are available. The first specifies an external EXODUS-format file containing current (and optionally, conductivity) with the options:

`FILE, file-name (problem_name)`

`LOADEXTERNALCONDUCT, bool | string (true)`

`HALTEXTERNALSIM, bool | string (true)`

`CURDENNAME, string (TRANS_CUR_DEN)`

`CONDUCTNAME, string (CONDUCTIVITY)`

This option directs the code to look for time planes of nodal current-source data in either the problem genesis file (if the `FILE` keyword is not set) or the alternate file name specified by the `FILE` keyword. Note that the alternate file only works for serial, if parallel is desired then the data must be placed in the problem genesis file. The `LOADEXTERNALCONDUCT` keyword specifies whether to read external conductivities from the same file; the default is to read them and must be disabled by entering `no` or `false`. The `CURDENNAME` keyword specifies the name in the genesis or alternate file of the current-density data. The default is "TRANS_CUR_DEN". The `CONDUCTNAME` keyword specifies the name of the conductivity data. The default is "CONDUCTIVITY". If the data is not found an error will halt execution.

If the simulation termination time exceeds the external time planes available, the simulation will gracefully terminate and complete post-processing steps such as far-field transient calculations. The `HALTEXTERNALSIM` keyword set to "no" or "false" can be used to modify this behavior to continue the simulation with J fixed at the final time plane provided.

Note that frequency-domain far-field patterns cannot be generated by de-convolution using this current source, so far-field patterns from external j-sources are not supported. Far-field transient waveforms use no de-convolution.

The external current-source time history can be viewed in the *problem_name*.exo file by requesting the plot variable "CUR_DEN". Since the external source is volumetric, there is no associated time history variable in this case.

The second form uses the RTC functionality to allow the user to specify current as a vector function of space and time with the `extj-spec` value `user defined`. The RTC function is C-style code that must be enclosed in double quotation marks ("..."). Special variable names are `time`, which uses the simulation time value, `coord`, an array of three values corresponding to the nodal coordinates (x, y, z), and `field`, an array of three values that specifies the vector $\mathbf{J}$. A full description of the options available with the RTC is given in Appendix E. An example of this usage follows.

```
EXTERNAL J SOURCE, BLOCK 1, USER DEFINED
"
  double pi = acos(-1.0);
  double freq = 1.e8;
  field[0] = 0.0;
  field[1] = 0.0;
  field[2] = -sin(freq*time) * sin(pi*coord[0]);
"
```

`PLANE WAVE SOURCE, SIDESET int, BLOCK int [int int ...], FUNCTION int, POLARIZATION X real Y real Z real, PROPDIR X real Y real Z real, [CENTER X real Y real Z real]`

Specifies a plane-wave source.

A plane wave will be launched in one or more mesh `BLOCK`(s), with polarization given by the vector following `POLARIZATION`, propagating in the direction given by the vector following `DIRECTION`, having the time history specified by `FUNCTION`. These block(s) define the total-field region which is bounded by the supplied `SIDESET`. Any remaining blocks in the simulation will be the scattered-field region.

The optional parameter `CENTER` specifies the location of the phase center of the plane-wave source in mesh coordinates. This can be any point in the source plane. If this parameter is not supplied, the code will compute a phase center which is just on the incident side of the total-field region, normal to the propagation direction `PROPDIR`. The user should take care to provide a phase-center location which makes sense relative to the total-field region and the propagation direction.

The source time history will be written to the hisplt database *problem_name*`.his` with the name: `PWSRC-`*sideset_id*.

2.8 Load keywords

`EDGE LOAD`

The following syntax specifies a single-element load on an element edge:

```
EDGE LOAD, NODESET int, type, VALUE real
EDGE LOAD, EDGESET int, type, VALUE real
```

If `NODESET` is specified, the edge will be located which is defined by the nodeset having id `int`. This nodeset should contain only 2 nodes. UTDEM will issue an error if it contains more than 2 nodes. If it contains less than 2 nodes no load will be applied.

If `EDGESET` is specified, the edge(s) specified in the edgeset is(are) used. Unlike the `NODESET` description above, the edgeset may contain one or more edges. If it contains more than 1 edge, the load will be divided equally between the edges in the edgeset. The assumption is that the edgeset contains edges in a "line" such that series connection of the sub-loads makes sense. If the edgeset contains no edges, no load is applied.

The parameter `type` is limited to "R", "L", or "C": a single resistor, inductor, or capacitor, respectively. Multiple `EDGE LOAD`s of the same or different `type`s may exist in the same simulation, but not on the same edge.

If it is desired to monitor the voltage across the load, an observer should be assigned to the same `NODESET` or `EDGESET` using the `OBSERVER` or `E LINE INTEGRAL` keywords.

`SPICE LOAD`

The following syntax specifies a load described by a Spice deck on an element edge:

```
SPICE LOAD, NODESET int, DECK, filename [, XYCE]
SPICE LOAD, EDGESET int, DECK, filename [, XYCE]
```

If `NODESET` is specified, the edge will be located which is defined by the nodeset having id `int`. This nodeset should contain only 2 nodes. UTDEM will issue an error if it contains more than 2 nodes. If it contains less than 2 nodes no load will be applied. If `EDGESET` is specified, the edge specified in the edgeset is used. The edgeset should contain only 1 edge. If if contains more than 1, UTDEM will issue an error. If it contains no edge, no load will be applied.

If the optional keyword `XYCE` is present on any load, all loads will be solved with Xyce [4] rather than the default Spice library.

The SPICE `DECK` containing the description of the spice load in the form of a sub-circuit must be provided with the `SPICE LOAD` keyword.

`SPICE MODEL spice_option`

Specifies how SPICE/Xyce is utilized and the origin of the spice model deck for the simulation. Options for `spice_option` are:

- `build` Lumped-parameter SPICE/Xyce deck will be auto-created and written to a file during startup, this is the default;
- `use` Lumped-parameter SPICE/Xyce deck will be read from file.

Generally, an initial simulation is accomplished using the `build` (default) option which writes the SPICE/Xyce deck to a file (see `SPICE FILE` keyword below). If custom changes are desired to this deck for subsequent simulations, the file can be edited and the `use` option invoked thereafter. If the `SPICE MODEL` keyword is not present, the default is to auto-build the deck.

`SPICE FILE "filename"`

Specifies the filename from which to read the SPICE/Xyce model deck if `SPICE MODEL`, `use` is specified, or the file to which to write the deck if `SPICE MODEL`, `BUILD` is specified or not present.

The argument `"filename"` is an ascii string and the quotes are required. If `SPICE FILE` is not specified, the SPICE deck is either read from or written to the default filename, *problem_ name*`.in`. Note that this file, either the default or that specified by

SPICE FILE, will be OVERWRITTEN for the next simulation if it is left in place on the file system.

SPICE STEP FRACTION real

Specifies the fraction of the simulation time step which is to be used for the maximum SPICE internal time step. The default value is 0.1.

2.9 Slot keyword

SLOT, EDGESET int, AZTEC_SET int, WIDTH real, DEPTH real, INT_MAT int, EXT_MAT int

Specifies a single sub-grid, thin-slot model.

The slot is defined to lie along the edges defined by the edgeset int with the specified WIDTH and DEPTH. Since the slot is solved using a separate linear system, an independent AZTEC_SET is defined for the slot (see Linear solver keywords). The id of this aztec_set should be something other than 0 (the default), which is reserved for the primary system solve. An example of this can be found in the input file given in Appendix B. Multiple slots may exist in the same simulation.

Since the slot algorithm requires a differential H-field drive based on fields on both sides of the PEC plane containing the slot, the materials on those sides must be defined by different material indices. These are defined by int_mat and ext_mat and are the same material id's as defined in the framework BLOCK keyword described below. These materials can in fact have identical constitutive parameters, but they must be entered as two different materials.

Terminating a slot on a periodic boundary has not been tested and is not recommended.

2.10 Wire keywords

WIRE, EDGESET int, AZTEC_SET int, RADIUS real

Specifies a single sub-grid, thin-wire model.

The wire is defined to lie along the edges defined by the edgeset int with the specified RADIUS. Since the wire is solved using a separate linear system, an independent AZTEC_SET is defined for the wire (see Linear solver keywords). The id of this aztec_set should be greater than 0, which is reserved for the primary system solve.

Multiple wires may exist in the same simulation. Wires end conditions must either be open or terminating on a PEC or periodic boundary. The ends of two wires cannot connect together as this condition is not implemented. If this situation arises, simply use a single wire with a bend or whatever is required.

The ends of the wire may be placed on opposite periodic boundaries to make the wire look "infinitely long". The user is responsible for verifying that the wire indeed lies in the mesh as expected. If somehow only one end of the wire is on a periodic boundary the code will proceed with possibly undesirable results as the periodic boundary implementation makes it difficult to detect this situation.

`UNCSTABWIRE, EDGESET int, AZTEC_SET int, RADIUS real`
Specifies a single sub-grid, unconditionally stable thin-wire model.

The wire is defined to lie along the edges defined by the edgeset `int` with the specified `RADIUS`. Since the algorithm uses a separate linear system, an independent `AZTEC_SET` is defined for the wire (see Linear solver keywords). The id of this aztec_set should be something other than 0, which is reserved for the primary system solve. Multiple wires may exist in the same simulation.

`WIRE LOAD, NODESET int, R, VALUE real`
Specifies a single thin-wire lumped resistive load.

The resistor is defined to lie along the edge defined by the nodeset `int` with resistance `VALUE`.

`WIRE LOAD, EDGESET int, R, VALUE real`
Specifies a single thin-wire lumped resistive load.

The resistor is defined to lie along the edge defined by the edgeset `int` with resistance `VALUE`.

3 Running UTDEM Simulations

Running a UTDEM simulation requires a Genesis mesh geometry file, *problem_ name*.gen, and an input file, *problem_ name*.inp (example in Appendix B). Assuming a standard Nevada user's environment or a subset thereof exists in the user's operating environment, simulations are run in one of two ways:

```
$ alegrabal -p <int> <problem_name>
$ Alegra <problem_name>
```

or

```
$ runAlegra <problem_name>
```

The `alegrabal` script invokes mesh decomposition software to divide and load balance the mesh among `int` processors. If serial execution is desired, this step is not required.

The `Alegra` or `runAlegra` script actually runs the simulation. For a parallel simulation being run for the first time after the `alegrabal` script is invoked, the decomposition is completed before the simulation starts. For batch runs on clusters, this may not be the case and "`Spread <problem_name>`" needs to be run after `alegrabal`. If your cluster job runs extremely slowly this is likely the problem because the job was run serial.

After the parallel simulation is completed, the `Alegra` script performs the necessary joining of the parallel ExodusII results into a single ExodusII file.

For EMPHASIS distribution releases, "alegra" or "Alegra" in the above commands is replaced with "emphasis" or "Emphasis", such that `alegrabal` becomes `emphasisbal` and `runAlegra` becomes `runEmphasis`. Additional options for each command can be obtained using "`-h`", such as "`runAlegra -h`".

4 UTDEM Input Options

`BOX IEMP [, LOAD CURRENT] [, LOAD DOSE]`

Specifies a Box IEMP simulation.

If the `BOX IEMP` keyword is given alone, variables are registered for DOSE and DOSE_RATE and the j source time history is normalized such that it's time integral is unity. If the `LOAD CURRENT` keyword is added, the vector current is loaded as provided by radiation transport through the simulation genesis file. This requires the `RADTRANS` keyword to be specified in the `J SOURCE`. If the `LOAD DOSE` keyword is added, the energy deposition is loaded as provided by radiation transport, again through the genesis file.

Note also that a `BOX IEMP` simulation requires that the RIC Electrical or HP Gas Electrical material model be used for all dielectrics. This requirement is enforced by UTDEM. The box-iemp source also requires that the corresponding material model use nodal variables (selected by *not* specifying NDOF, ie, taking the default).

`BOX CABLE`

Specifies a coupled Box IEMP/Cable SGEMP simulation.

The use of this keyword requires EMPHASIS/CABANA [13] to also use the keyword. The two codes, EMPHASIS/UTDEM 3D and EMPHASIS/CABANA 2D, then communicate through the SPICE or Xyce solve to couple a UTDEM `SPICE LOAD` to a single cable conductor. Both codes must be using the same time step, termination time, and spice control.

For UTDEM, `BOX IEMP` must be specified along with a `SPICE LOAD`. A deck will be required for the `SPICE LOAD`; use something like

```
SPICE LOAD, EDGESET 1, DECK, "resistor.in" [, XYCE]
```

The coupling works with either the default SPICE or Xyce library. Place a resistor in "resistor.in" that will be used in the setup step below and ignored for the coupled simulation. Instead, a pre-edited spice deck will be used by both codes, specified by

```
SPICE MODEL, USE
  SPICE FILE, "box_cable.in"
```

In CABANA, the cable is described as usual along with the two keywords above in the same format.

The coupled simulation is executed similar to single simulations using

```
$ runEmphasis -x $EMPHASIS_3D_EXE -n emphasis box :
   -x $EMPHASIS_2D_EXE -n emphasis cable
```

Instructions for generating the box_cable.in file are the following:

1. In the UTDEM input deck box.inp, comment `SPICE MODEL, USE` and `SPICE FILE, "box_cable.in"` so that the SPICE/Xyce deck is built. Similarly, do this for the CABANA input deck cable.inp while adding "`SPICE MODEL, BUILD`". In addition, comment out the `BOX CABLE` keyword in each input deck. Now run both codes uncoupled, either using the execution line above or separately, generating initial SPICE/Xyce decks for the uncoupled problems, box.in and cable.in.
2. Copy the UTDEM SPICE/Xyce deck box.in to box_cable.in and edit the file:
   (a) Switch nodes 1 and 10001
   (b) Change node 2 to 20001
   (c) Change node 3 to 30001
   (d) Change ISRC1 to ISRC
   (e) Change C1 to C
   (f) Remove the X1 line
   (g) Remove the SUBCKT stuff
   (h) Keep the .TRAN, .OPTIONS, .PRINT TRAN, and END lines
3. From the CABANA SPICE/Xyce deck cable.in:
   (a) Copy in the circuit description
   (b) Remove the first line comment and the R1 line
   (c) Add the V(*)'s from the .PRINT TRAN line(s) to the .PRINT TRAN line from box.in, not exceeding 8 per line
   (d) The .TRAN and .OPTIONS in cable.in should be identical to those already in box_cable.in from box.in. Only one set should remain.

At this point, revert the two input decks by removing the comments from "`BOX CABLE`", "`SPICE MODEL, USE`" and "`SPICE FILE, "box_cable.in"`" lines. Comment the "`SPICE MODEL, BUILD`" line in cable.inp.

`UPDATE MAT STATE, bool | string, TSTART real TEND real INTERVAL real`

Specifies whether material state is updated at the end of each time cycle.

Options are true (or yes) and false (or no) [default]. If specified, the material state is updated and a matrix refill is triggered. This is generally used only with the `BREAKDOWN ELECTRICAL` material model. The time window over which this applies can be controlled using the `TSTART` and `TEND` keywords. The `INTERVAL` keyword specifies how often within the specified time window the update/refill is triggered. If `INTERVAL` is 0., then it is triggered every time cycle.

This keyword is not necessary for `BOX IEMP` simulations using energy deposition to drive the `RIC ELECTRICAL` material model since UTDEM forces the matrix refill in that case.

`TIME STEP MOD, real`

Specifies a global modification to the UTDEM-computed stable time step.

The default is 1. This value does not effect the time step if specified using the framework keyword `CONSTANT TIME STEP`.

Time step information is written to the output stream and appears just after the "EMPHASIS/UTDEM" banner under the title "Time Step Info:".

The first time step given is the 3D Courant time step computed from the global minimum edge length.

The second is the EMPHASIS recommended time step modified by `TIME STEP MOD`. This time step will be used for the simulation unless `CONSTANT TIME STEP` is specified or, for PIC simulations, `COURANT FACTOR` is specified. This time step is slightly larger than 3D Courant and should be fine for most simulations, even PIC simulations with non-relativistic particles.

The third is a larger, empirically determined time step based on the maximum edge length and should provide good results for most pure electromagnetic simulations. To use the first or third time step the simulation will have to be restarted specifying the desired time step using `CONSTANT TIME STEP`.

`COMPUTE ENERGY, bool | string`

Specifies whether or not electric, magnetic, and total energy is computed throughout the computational volume.

The possible options are true (or yes) and false (or no). The default is presently false. If true, appropriate energy global variables are registered and the energy time histories are compute and written to the hisplt database *problem_ name*.his with the names: E-ENERGY, H-ENERGY, and TOT-ENERGY. Energies are defined as

$$\text{E-ENERGY} = \frac{1}{2}\int \epsilon \mathbf{E}\cdot\mathbf{E}dV, \tag{4.1}$$

$$\text{H-ENERGY} = \frac{1}{2}\int \mu^{-1}\mathbf{B}\cdot\mathbf{B}dV, \tag{4.2}$$

$$\text{TOT-ENERGY} = \text{E-ENERGY} + \text{H-ENERGY}. \tag{4.3}$$

`JOULE HEATING, string`

A more detailed energy diagnostic which tallies changes in energy due to Joule heating and currents. The possible options are ON and OFF. When turned on, this option also sets COMPUTE ENERGY to true; therefore, the COMPUTE ENERGY option is redundant when JOULE HEATING is turned on.

In addition the following quanties are computed. Two element variables are generated. Define T as given output time. In each element P the local power densities are stored

$$\text{JOULE\_POWER\_DENSITY} = \frac{1}{|P|}\int_P \sigma\mathbf{E}(T)\cdot\mathbf{E}(T)dV, \tag{4.4}$$

$$\text{SOURCE\_POWER\_DENSITY} = \frac{1}{|P|}\int_P \mathbf{J}(T)\cdot\mathbf{E}(T)dV. \tag{4.5}$$

If requested these outputs appear in *problem_ name*.exo. In addition global time histories are written to the hisplt databas *problem_ name*.his

$$\text{GLOBAL\_JOULE\_POWER} = \int \sigma \mathbf{E}(T) \cdot \mathbf{E}(T) dV \tag{4.6}$$

$$\text{GLOBAL\_SOURCE\_POWER} = \int \mathbf{J}(T) \cdot \mathbf{E}(T) dV \tag{4.7}$$

$$\text{GLOBAL\_JOULE\_ENERGY} = \int_0^T \int \sigma \mathbf{E}(T) \cdot \mathbf{E}(T) dV dt \tag{4.8}$$

$$\text{GLOBAL\_SOURCE\_ENERGY} = \int_0^T \int \mathbf{J}(t) \cdot \mathbf{E}(t) dV dt \tag{4.9}$$

5 UTDEM PIC Input File and Keywords

The UTDEM PIC input file also uses the standard NEVADA input file format. It permits all of the UTDEM physics-specific command keywords described in the preceding sections, as well as several new command keywords that are described later in this section.

The keyword `UNSTRUCTURED TD ELECTROMAGNETIC PIC` specifies to the Nevada framework that the UTDEM PIC physics model should be used for the simulation. All the UTDEM keywords are available, as well as several more that are specific to UTDEM PIC. As before, case is ignored in the input file and lines are limited to 160 characters or less. And remember that when a floating-point value is required, the decimal point needs to be included (i.e., `0.` not simply `0`).

5.1 PIC-Specific Keywords

The following keywords are available to UTDEM PIC physics but are also available to the Structured Time-Domain ElectroMagnetics (STDEM) PIC physics model (see [1]).

`DEFINE SPECIES`

The `DEFINE SPECIES` command keyword group provides a means of defining charged-particle species to be used with the various particle emission and diagnostic commands. One or more particle species may be defined in one `DEFINE SPECIES` keyword group using the following syntax:

```
DEFINE SPECIES
  string, MASS = real, CHARGE STATE = int, [MARK_AS_USED,]
  [REGISTER_DENSITY]
  ...
END
```

Each species is specified by three required parameters. The first parameter is a string that provides a user-specified name for the species. This name is used to reference the defined species in the all other input commands that require the use of a particle species in their specification. The two remaining parameters for specifying a species are the `MASS` and `CHARGE STATE`, which describe the particle species mass (in AMU), and charge state, respectively. The charge state is a signed integer that gives the particle's charge relative to that of a proton. For example, an electron's charge state would be -1 and triply ionized carbon would be +3. The optional `MARK_AS_USED` keyword indicates that this species is to be immediately marked as use without waiting for another command that references it. The optional `REGISTER_DENSITY` keyword indicates that the charge density for this species will be unconditionally stored in its own registered variable. If not supplied, this specie's charge density will only be stored in its own registered variable if the variable name RHO_`string`, where `string` is the name of the species, is specified by the `PLOT VARIABLE` command.

`GAS DRAG`

The `GAS DRAG` command keyword enables a gas drag model for interaction between electrons and a high pressure gas. The drag force slows down the electrons, and the corresponding energy loss is computed in each element as an IONIZATION_RATE variable to be used as a source of electron-ion pairs for the gas-breakdown plasma. If this command keyword is used then a HP Gas Electrical material model must also be used. Currently the gas drag model requires the same high pressure gas be present throughout the entire simulation domain. The gas drag model also requires that the corresponding material model use elemental variables (selected by setting NDOF=1). The HP Gas Electrical model also has keywords to enable angular scattering of the electrons in addition to the drag force.

`ITS FILE, ["surface" | "volume"], [file-name]`

The `ITS FILE` command keyword is used to provide the name of the file containing the ITS distribution datasets needed by either the `BEAM EMISSION` or `VOLUME EMISSION` command keywords when using the `FIELD DISTRIBUTION = ITS` option.

For backward compatibility, if the file type (`surface` or `volume`) is not specified, this command defines the name for the `BEAM EMISSION` file. For `VOLUME EMISSION`, the type is required. If `file-name` is not specified, the name *its.pff* will be used for the surface emission file. Note that unless the file name consists of only identifier characters (uppercase letters, digits, and underscore '_'), the supplied `file-name` must be quoted. Since this restriction eliminates almost all commonly used names for files, it is recommended that the filename always be quoted.

`CROSS SECTION DATABASE string`

The `CROSS SECTION DATABASE` command keyword is used to provide the pathname of the file containing the gas cross section database. Currently this file is used by the `KINETIC GAS ELECTRICAL` material model below.

The required `string` is the pathname of the database. If this keyword is not specified, the file *CrossSections.txt* in the current run directory will be used. Note that unless the file name consists of only identifier characters (uppercase letters, digits, '_'), the supplied filename must be quoted. Since this restriction eliminates almost all commonly used names for files, it is recommended that the filename always be quoted.

`PARTICLE HISTORY`

The `PARTICLE HISTORY` command keyword group provides a means of requesting output history diagnostics for various types of aggregate particle information. One or more such requests may be made in one `PARTICLE HISTORY` keyword group. The required syntax is:

```
PARTICLE HISTORY
  his_type, history_specification
  ...
END
```

The initial `his_type` string defines the type of the history request. Supported request types are:

1. `count`, which gives the number of particles of the specified species.
2. `energy`, which gives the total kinetic energy of all particles of the specified species.
3. `charge`, which gives the total charge of all particles of the specified species.
4. `merge stats`, which gives various statistics regarding the performance of the particle merger algorithm.
5. `merge counts`, which provides various data on merged particles.
6. `kpflux`, which requests one or more "killed particle flux" (KPF) histories tallying information about particles killed on a surface.

Three forms of `history_specification` are available:

1. `his_type, [SPECIES = string,] [STATUS = string,] [LABEL = "string"]`
   This form provides global tallies, which can be filtered by species using the `SPECIES` keyword, and by particle status using the `STATUS` keyword. Options available in this specification are:

   `SPECIES = string | all`
   Specifies whether a single particle species or all species contribute to the history. This keyword can take the value of a single species as defined by the `DEFINE SPECIES` command, or the special value `all` to combine the information for all particle species in the simulation. Default is `all`.

   `STATUS = string`
   Legal values for the `STATUS` keyword depend on the history type, as follows:

   For `merge stats`**:**
   `STATUS = fail | fail_reject | fail_qerr | fail_qlow | fail_emax`
   The `fail` value will report the total number of merge failures for any reason; the others will report counts of merge failures by each specific failure mechanism.

For `merge counts`**:**

`STATUS = sampled | created | killed | netkilled`

Note that `netkilled` is the difference between the total number of killed pre-merge particles and the total number of newly merged particles created to replace them.

For all other history types:

`STATUS = created | killed | surviving`

These request the cumulative number of particles created during the simulation, the cumulative number of particles killed during the simulation, and the number of particles that currently survive in the simulation, respectively.

`LABEL = "string"`

Specifies the label to be used for the history output for this request. If not supplied, a default label will be constructed based on the values of the other keywords.

This form cannot be used with the `kpflux` history type.

2. `his_type, KINETIC GAS MODEL = int, [SPECIES = string,] [GAS = string,] [INTERACTION = string,] [LABEL = "string"]`

This form provides information associated with kinetic gas collisions from a `KINETIC GAS ELECTRICAL` material model, and can be filtered by the incident species of the collision, the gas molecule of the collision, and the type of collision. Options available in this specification are:

`KINETIC GAS MODEL int`

The required `KINETIC GAS MODEL` keyword is specifies the integer ID of the material model, i.e., the ID supplied with the material model's definition.

`SPECIES = primary | secondary | all`

Specifies the electron species of the incident particle in a collision. Defaults to ALL if not supplied. Here, `primary` and `secondary` indicate the electron species specified by the material model's PRIMARY and SECONDARY keywords, respectively, and `all` specifies that the contributions from both species be combined.

`GAS = string`

Limits contributions to those from collisions with a specific gas molecule of the material model, where the supplied string is the name of the molecule as defined in the material model. If this keyword is not supplied, contributions from all gas molecules in the model are combined.

`INTERACTION = elastic | excitation | ionization | attachment | all`

Specifies the type of collisions to be included in the history. If this keyword is not supplied, ALL will be used, indicating that the contributions from all interactions are to be combined.

`LABEL = "string"`

Specifies the label to be used for the history output for this request. If not supplied, a default label will be constructed based on the values of the other keywords.

This form cannot be used with the `kpflux` history type.

3. `kpflux, SPECIES = string, SIDESET = int [... int], TYPE = label, [... TYPE = label]`

   This form is exclusive to the `kpflux` history type and is specific to UTDEM; KPFLUX histories are not currently implemented in STDEM. It provides information on particles killed on boundary surfaces. Options available in this specification are:

   `SPECIES = string`
   (Required) Specifies the name of the single species to be reported. The special value `all` cannot be used with this history type.

   `SIDESET = int [... int]`
   (Required) Defines the boundary surface of interest with one or more sideset indices.

   `TYPE = number | current | mean_energy | mean_px | mean_py | stdv_energy | stdv_px | stdv_py | stdv_pz`
   One or more `TYPE` = `label` pairs may be provided within a KPFLUX history specification. The units for the output values are Amps for current, m/s for momentum, and $\gamma - 1$, i.e., E/mc^2, for energy.

   It is *much* more efficient to group all KPF history types for a given species and sideset(s) into a single KPF request. The reason is that for a given killed particle, the code only needs to find which KPF request to update once. However, it is legal to create histories spread across two or more KPF requests with the same species and sideset(s). A more important reason to group all histories into a single request is that the number of requests is limited. For each species, the maximum number is determined by the smaller of the number of bits in an 'unsigned long' or 'double', For almost all modern 64-bit machines, this limit is 64. Note that this does create a portability issue for running problems with more than 32 histories on older machines where the limit may be 32. However, this is an increasingly unlikely occurrence.

`PARTICLE SNAPSHOT`

The `PARTICLE SNAPSHOT` command keyword group provides a means of requesting output snapshot diagnostics for particles. One or more such requests may be made in one `PARTICLE SNAPSHOT` keyword group. The data for all requested particle snapshots will be written to the file *problem_name.par.pff* in PFF format.

```
PARTICLE SNAPSHOT
  max1, [max2], SPECIES = string, ATTRIBUTES = string, LABEL = string,
  FRACTION real FIRST int LAST int SKIP int
  ...
END
```

Each request has one, or optionally two, integers (`max1` and `max2`), which control the number of particles for which data will be output. The larger is the maximum number of particles to be output. The smaller is the maximum number of particles from a single processor to be output. If only one value is specified, the two values are assumed to be equal. The `SPECIES` keyword allows the specification of output for a single particle species as defined by the `DEFINE SPECIES` command keyword group, or the special value ALL, which outputs data for all particle species in the simulation. If not explicitly specified, `SPECIES` defaults to ALL. The `ATTRIBUTES` keyword is a string containing the characters "`P`" and/or "`Q`", indicating that momentum and/or charge particle attributes are to be output in addition to the particles' spatial locations. Default is an empty string (" "), indicating that only particle spatial locations will be output. The `LABEL` keyword allows the user to supply a title prefix for the PFF dataset that will be written to the output file for this request. The simulation time will be appended to this prefix. If not supplied, a title prefix will be generated automatically from the `SPECIES` and `ATTRIBUTES` keyword values. The `FRACTION` keyword is used to specify a real number, between 0 and 1, indicating the maximum fraction of the simulation particles that will be output. If not specified, it defaults to 1.0. Note that if the product of this fraction and the total number of simulation particles exceeds `max1`, the actual fraction of the particles output will be less than the value specified by the `FRACTION` keyword. The `FIRST`, `LAST`, and `SKIP` keywords are used to control the output frequency for the diagnostic request. The `FIRST` and `LAST` keywords specify the first and last timestep indices, respectively, for which particle data will be output. The `SKIP` keyword is a positive integer that specifies the number of timesteps between outputs for this request. A value for `SKIP` must be specified. If `FIRST` is not specified, its default value is that of `SKIP`. If `LAST` is negative, there is no upper limit for the timestep index. If not specified, `LAST` defaults to -1.

5.2 UTDEM PIC-Specific Keywords

This section describes keywords available only to UTDEM PIC.

`EMISSION` Keywords

Two `EMISSION` command keywords define a surface over which particles are emitted into the simulation region. The `BEAM EMISSION` source creates particles to match a user-specified current, $I(t)$, emitted from the surface. The `FIELD EMISSION` source applies a space-charge-limited (SCL) algorithm locally on each face, creating enough charge to satisfy $E_{\text{normal}} = 0$. The particle species and other emission characteristics are provided by the several available parameter options. The two commands, whose formats are shown below, are grouped together here because they have several common keywords. In addition, there are a few more keywords that apply only to one or the other of the commands, or are interpreted somewhat differently by each command. The specific formats are:

```
BEAM EMISSION
```

```
    SIDESET int
    SPECIES = string
    [CYCLE INTERVAL int]
    [COUNT int]
    [EMIT PROBABILITY real]
    [SPATIAL DISTRIBUTION = FIXED | RANDOM]
    ENERGY DISTRIBUTION = CONSTANT real | RANDOM real [real] |
                          MAXWELLIAN real |
                          ITS [int] [, MERGE_PHI]
                          [, PHI0, X=real Y=real Z=real]
    [ANGLE DISTRIBUTION = NORMAL | CONSTANT, X=real Y=real Z=real |
                          RANDOM [real real] | COSINE]
    [NORMAL TOLERANCE real]
    [AMPLITUDE real]
    [TEMPORAL function-set]
    [QPMIN_FUN function-set]
    [QPMIN_FLOOR real]
  END

  FIELD EMISSION
    SIDESET int
    SPECIES = string
    [CYCLE INTERVAL int]
    [COUNT int]
    [EMIT PROBABILITY real]
    [SPATIAL DISTRIBUTION = FIXED | RANDOM real [real]]
    ENERGY DISTRIBUTION = CONSTANT real | RANDOM real [real]
    [ANGLE DISTRIBUTION = NORMAL | CONSTANT, X=real Y=real Z=real |
                          RANDOM [real real] | COSINE]
    [NORMAL TOLERANCE real]
    HEIGHT DISTRIBUTION = FIXED real | RANDOM real [real]
    BREAKDOWN real
    [QPMIN_FUN function-set]
    [QPMIN_FLOOR real]
  END
```

Options common to `BEAM EMISSION` and `FIELD EMISSION` commands:

`SIDESET int`
: Provides the number of the sideset in the Genesis input file that provides the set of faces that comprise the emission surface for this beam.

`SPECIES = string`
: Provides the name of the particle species to be emitted and must match the name of a species that has been defined using the `DEFINE SPECIES` command keyword.

`CYCLE INTERVAL int`
: Specifies the emission frequency in timesteps, with a default value of 1.

`COUNT int (1)`
: Specifies the number of particles emitted from each face in the emission per timestep, with a default value of 1.

`EMIT PROBABILITY real (1.0)`
: Allows the specification of the probability, between 0 and 1, that any particle will actually be inserted into the system. It defaults to 1.0.

`ENERGY DISTRIBUTION`
: Describes the energy distribution of the emitted beam particles. The `BEAM EMISSION` and `FIELD EMISSION` commands have two common options: CONSTANT and RANDOM. The CONSTANT option has a single value that gives the beam energy in electron Volts (eV). The RANDOM option has two values that give the minimum and maximum energies (in eV) for a uniform distribution of beam energy. If only one value is specified, it is assumed to be the maximum energy of the distribution, and the minimum energy is assumed to be zero. The `BEAM EMISSION` command has two additional options; see below.

`ANGLE DISTRIBUTION`
: Describes the direction of emission with respect to the polar angle (θ) from the surface. Four options are available: NORMAL, CONSTANT, RANDOM, and COSINE. The NORMAL option has no parameters, and specifies that particles are to be emitted with an initial velocity normal to the face from which they are emitted (i.e., $\theta = 0$). The CONSTANT option has three values that represent the three components of a vector in Cartesian (x, y, z) space. Particles emitted from the emission surface will be given an initial velocity in the direction of this vector, regardless of the orientation of the emission face from which they is emitted. The RANDOM option has two values that give the minimum and maximum θ (in degrees) for a uniform distribution over θ. If these values are not specified, default values of 0° and 90° will be used. The COSINE option will use a cosine distribution (i.e., $\mathrm{d}N/\mathrm{d}\theta = \cos\theta$) over the range from 0° to 90°. Note that if `ANGLE DISTRIBUTION` is set to RANDOM or COSINE, the emitted particles will be randomly distributed (uniformly) in azimuth relative to the emission face. Note that if the `ANGLE DISTRIBUTION` keyword is not explicitly specified, it will default to NORMAL, unless `ENERGY DISTRIBUTION` is specified to be MAXWELLIAN, in which case `ANGLE DISTRIBUTION` defaults to COSINE.

`NORMAL TOLERANCE real (0.001)`
: Allows the user to specify the tolerance (ϵ) with which the code determines that the unit normal vectors of two distinct faces are equal. That is, if $|\hat{n}_1 - \hat{n}_2| \leq \epsilon$, they are considered to be equal. If not specified, ϵ will default to 0.001. For all cases except `ANGLE DISTRIBUTION` = CONSTANT, each face in an emission surface needs an object that contains, among other things, the face's unit normal vector in order to generate the velocity of an emitted particle. To the extent that multiple faces have the same normal vector, they can utilize the same object.

Consequently, use of this keyword is primarily related to code efficiency. Note that this keyword's value provides an upper bound for the error in the normal vector for any face.

`QPMIN_FUN function-set`

The optional `QPMIN_FUN` and `QPMIN_FLOOR` keywords define a minimum charge weight for creating particles on the segment. The intention is to reduce the total particle count by inhibiting the creation of "insignificantly low-weight" particles. The minimum particle weight is defined by:

$$\begin{aligned} \text{qpmin}(t) &= \text{qpmin\_floor}, \qquad \text{or} \\ \text{qpmin}(t) &= \max(\text{scale} \times \text{fun}(t), \text{qpmin\_floor}) \end{aligned}$$

Defining just the `QPMIN_FLOOR` keyword selects the first option, a fixed qpmin(t). The `QPMIN_FUN` keyword selects the time-dependent option, and uses the standard Nevada function syntax '`FUNCTION int [SCALE real]`' to define the function number and an optional scale factor. For this option, the `QPMIN_FLOOR` value defines an absolute minimum charge value for a created particle.

Choosing these parameters for `FIELD EMISSION` is necessarily empirical. However, values for `BEAM EMISSION` are easier to determine *a priori*. The charge-to-create at each face of area A, Q_{cre}, is a running sum of $I(t)A\Delta t_{\text{emit}}/A_{\text{segment}}$, where $I(t)$ is the time function defined by the `TEMPORAL` keyword, and $\Delta t_{\text{emit}} = \text{cycle\_interval} \times$ timestep. Once this sum exceeds qpmin(t), N particles are created, where $N = \min(\text{int}(Q_{\text{cre}}/\text{qpmin}), \text{count})$ and Q_{cre} is reset to zero. Information to guide the choice of qpmin parameters for both standard and ITS beam emission is printed to the output file. For each segment, the code prints out values of min(Q_{cre}), mean(Q_{cre}), and max(Q_{cre}) for the faces. The actual runtime values of these quantities are scaled by $I(t) \times \text{cycle\_interval}$.

Options specific to the `BEAM EMISSION` command:

`SPATIAL DISTRIBUTION FIXED | RANDOM`

Describes the spatial distribution of the emitted beam particles. Two options are available: FIXED and RANDOM. If FIXED is specified and the `COUNT` keyword has a value of 1, then the single particle emitted will be placed at the barycenter of the emitting face. If FIXED is specified and the `COUNT` keyword has a value greater than 1, then the particles emitted will be placed according to a fixed, quasi-uniform algorithm over the emitting face. If RANDOM is specified, each emitted particle is placed at a random point on the emitting face. The distribution of these randomly-chosen emission points is uniform per unit area. If not specified, `SPATIAL DISTRIBUTION` defaults to FIXED, unless the `COUNT` keyword is specified to be greater than 1, in which case it defaults to RANDOM.

`AMPLITUDE real (1.0)`

Provides a scalar multiplier to specify the amplitude of the beam current, in amperes, with a default value of 1.0.

`TEMPORAL function-set`

Provides a means to specify non-constant time dependence for the amplitude of the beam current. If specified, the amplitude at any instant in time is the product of the scalar multiplier provided by the cmdoptamplitude keyword and the value of the function-set evaluated at the current simulation time. If `TEMPORAL` is not specified, the beam current is just the value supplied for the `AMPLITUDE` keyword, independent of simulation time.

Presently, the beam current is assumed to be distributed uniformly over the entire emission segment, i.e., the current density is constant. It should be noted that, by convention, the product of the `AMPLITUDE` value and the value of the `TEMPORAL` function at any given time must be non-negative. If the user supplies a combination of `AMPLITUDE` and `TEMPORAL` that results in a negative value for the amplitude of the beam current, it will be clipped to zero.

The `BEAM EMISSION` command has two additional options for `ENERGY DISTRIBUTION`: MAXWELLIAN, and ITS. The MAXWELLIAN option has one value that gives the temperature of the distribution in eV. For this option, the energies of the emitted particles will be distributed using

$$\frac{\mathrm{d}N}{\mathrm{d}E} = 4\pi \left(\frac{m}{2\pi kT}\right)^{3/2} e^{-E/kT},$$

where kT is the supplied temperature of the distribution.

The ITS option requests that the energies (and angles) for emitted particles are obtained by randomly sampling from data in a PFF file [10] computed by the ITS Radiation Transport code [6] (or any other source that follows the ITS/Emphasis convention for writing energy-angle (E, θ, ϕ) emission distributions to PFF datasets). See Ch. 11 for more details. The name of the source PFF file is specified using the `ITS FILE` command keyword.

The optional ITS keyword *int* value depends on the type of PFF file being used. For an *original-version-0* file, this value is ***required***—the user must explicitly define the PFF dataset to use for the sideset. For curved surfaces with these files, the user must also often use the `PHI0` keyword to explicitly set a vector to define the $\phi = 0$ axis. For *version-0* and *version-1* files, Emphasis auto-fits the sideset to the ITS subsurface(s), so the *int* value and the `PHI0` keyword are not required Note that for a *version-0* file, the auto-fit may occasionally fail, and explicitly providing the PFF dataset number may fix the problem. The *int* value is always ignored for a *version-1* file. Setup errors with these files indicates a fatal inconsistency between the ITS and Emphasis geometries. Note that if the *int* value is omitted, a comma is required for between "ITS" and either "MERGE_PHI" or "PHI0".

Finally, the ITS option also has the optional keyword MERGE_PHI, which indicates that multiple azimuth (ϕ) bins in the supplied dataset are to be merged into a single bin. In this case, the code can always internally define a $\phi = 0$ axis, and the emission angle is randomly selected from the range $[0, 2\pi]$.

Options specific to the `FIELD EMISSION` command:

`SPATIAL DISTRIBUTION FIXED | RANDOM`

The `SPATIAL DISTRIBUTION` keyword describes the spatial distribution of the SCL-emitted particles. Two options are available: FIXED and RANDOM. If the `COUNT` keyword has a value of 1, the FIXED and RANDOM options behave identically – the single particle emitted will be placed at a location ($\mathbf{x}_F$) in the emitting face chosen to best correct the charge deficit at each node of the face. If the `COUNT` keyword has a value greater than 1, then the particles are emitted in pairs. If the value of the `COUNT` keyword is odd, an additional single particle is emitted at $\mathbf{x}_F$. If the `SPATIAL DISTRIBUTION` keyword is FIXED, one particle of each emitted pair will be placed according to a fixed, quasi-uniform algorithm over the emitting face. Similarly, if RANDOM is specified, one particle of each emitted pair is placed at a random point on the emitting face. The distribution of randomly-chosen emission points is uniform per unit area. For both the FIXED and RANDOM options, the location of the second particle as well as the charges of both particles are chosen to best correct the charge deficit at each node of the face, with the added constraint that the second particle's location be within the emitting face. If not specified, `SPATIAL DISTRIBUTION` defaults to RANDOM.

`HEIGHT DISTRIBUTION FIXED | RANDOM`

Specifies the height above the emission face that each emitted particle will be injected into the simulation. The value provided should be between 0 and 1, and represents the height as a fraction of the height of the element into which the particle is being emitted. Two options are available: FIXED and RANDOM. The FIXED option has a single value for the emission height fraction. The RANDOM option has two values that give the minimum and maximum for the emission height fraction. If only one value is specified, it is assumed to be the maximum, and the minimum is assumed to be zero. There is no default for the `HEIGHT DISTRIBUTION` keyword so it must always be specified.

`BREAKDOWN real`

Specifies the electric field intensity ($\mathbf{E}$), normal to the surface of an emission face, required to initiate emission from that face. Its value should be provided with units in volts/meter. Until the normal electric field at any face exceeds this supplied breakdown value, that face will not emit any particles. Once the supplied value is exceeded, the face will emit particles in an SCL fashion for the remainder of the simulation. There is no default for the `BREAKDOWN` keyword so it must always be specified.

`VOLUME EMISSION`

This command keyword enables emission of electron/ion pairs or just electrons from a volume defined as the union of one or more element blocks. It is a volume analog of the `BEAM EMISSION` command, in which a total emission current $\mathbf{I}(t)$ is specified for the entire volume. The full syntax is as follows:

```
VOLUME EMISSION
  BLOCK int [int ... int]
```

```
  SPECIES = string
  [COUNT int int int]
  [CYCLE INTERVAL int]
  [EMIT PROBABILITY real]
  [SPATIAL DISTRIBUTION = FIXED | RANDOM]
  ENERGY DISTRIBUTION = MAXWELLIAN real | ITS [MERGE_PHI]
  [MOMENTUM X=real Y=real Z=real]
  [ANGLE DISTRIBUTION = RANDOM [real real]]
  [AMPLITUDE real]
  [TEMPORAL function-set]
  [QPMIN_FUN function-set]
  [QPMIN_FLOOR real]
END
```

The options `CYCLE INTERVAL`, `EMIT PROBABILITY`, `AMPLITUDE`, `TEMPORAL`, `QPMIN_FUN` and `QPMIN_FLOOR` have exactly the same meaning as the beam emission command, while the options `BLOCK`, `COUNT`, and `MOMENTUM` have the same meaning as the particle preload (see below).

The emission volume is defined by the element block list, where each `int` is a block-ID as defined in the Genesis file. The `string` for the `SPECIES` option can either be the name of a single electron species, or a quoted, space-delimited string of an electron species name followed by one or more ion species names (note: currently, only one ion type is supported). Creating just electrons is equivalent to creating electron-ion pairs with infinitely massive ions. However, in this case, the $\nabla \cdot \mathbf{D} - \rho$ diagnostics will be corrupted since they do not currently include the ion space-charge. The `COUNT` option defines three integers whose product is the number of particles loaded per element. These integers denote the spacing of a 3D lattice which distributes the inserted particles uniformly throughout the element. If not supplied, it defaults to "1 1 1". The `SPATIAL DISTRIBUTION` option defines whether to distribute the particles either randomly (the default) or uniformly.

There are two broad categories of supported electron energy-angle distributions: ITS and non-ITS, For non-ITS emission, the `MOMENTUM` keyword is required input, to define $\mathbf{p}_0 = \gamma \mathbf{v}_0$, where γ is the relativistic factor $(1 - (v_0/c)^2)^{-1/2}$. This defines a reference direction for emission that is analogous to the surface normal for `BEAM EMISSION`. The default option is to emit a cold beam with $\mathbf{p} = \mathbf{p}_0$ for all particles. Currently, there are only two other options: (1) a warm beam with an isotropic Maxwellian distribution about $\mathbf{p}_0$ defined by "`ENERGY DISTRIBUTION` = `MAXWELLIAN theta`", where `theta` is the temperature in eV, or (2) all particles emitted with $|\mathbf{p}| = \mathbf{p}_0$ in a cone around $\mathbf{p}_0$ with polar angles defined by the `ANGLE DISTRIBUTION` keyword.

For ITS emission, the user supplies a PFF file with ITS volume electron emission data, analogous to the ITS option for `BEAM EMISSION`. The PFF file name is defined with the `ITS FILE` command. EMPHASIS automatically fits the centroid of each emission element to an ITS emission subzone. At run time, the energy and angles are randomly sampled from a table built from the ITS subzone emission tally, $f(E, \theta, \phi)$. Note that unlike surface emission, the reference frame for θ and ϕ for each tally in the file is fixed for all emission subzones, i.e., with respect to the global (x, y, z) coordinate frame of the ITS simulation.

The ITS emission option also supports the loading of ITS energy deposition data in all elements of the blocks. This is used as a direct ionization source for the HP gas model (in addition to PIC particle energy loss). An ITS volume emission file can contain both energy deposition and particle emission datasets. By default, all data in the file is loaded unconditionally (and transparently to the user). There is very little overhead for implementing energy deposition. However, there are cases where it useful to turn off particle emission. This can be done by explicitly defining the `COUNT` option with three integers whose product is zero, *e.g.* "0 0 0".

Finally, note that there is currently no user control over the created ion energy-angle distribution if ion creation is specified. All ions are created with a non-drifting Maxwellian at room temperature (20°C).

`COURANT FACTOR real, [REFERENCE TYPE = ref_type]`

This command keyword allows the user to specify a factor $F_C > 0$ to multiply the Courant time step $\Delta t_C = \ell_s / c$, where ℓ_s is a specified scale length. This results in a simulation time step $\Delta t = F_C \Delta t_C$.

The specific scale length used can be selected using the `REFERENCE TYPE` keyword. Available values for `ref_type` are:

```
minimum edge length
average edge length
minimum element height
```

The first two length scales are the minimum and average lengths of any edge in the problem domain, respectively, and the third is the minimum height of any element in the problem domain. The minimum height of an element is defined to be the minimum distance from any face of the element to any other node of that element that is not in that face. If not supplied, `minimum edge length` will be used.

If the `CONSTANT TIME STEP` keyword (see [3]) is also specified, then the time step will be set to the smaller of the time steps determined from the two specifications. If neither `COURANT FACTOR` nor `CONSTANT TIME STEP` is specified, the default UTDEM time step (see the `TIME STEP MOD` keyword) will be used (not recommended practice). It is important to note that the time step due to the electromagnetic field solver formulation being used should not be exceeded regardless of the method chosen for setting the time step.

`RANDOM SEED int [, MULTIPLIER = int]`

This keyword changes the initial random seed for random numbers.

Without this keyword, the default behavior is to use an initial random seed of 1 for each new simulation. The optional `MULTIPLIER` defaults to 1. The new initial seed is `RANDOM SEED` + `MULTIPLIER` × processorNumber.

```
ELECTRON SURFACE DATABASE, TYPE=string, NAME=string, FILE=string [, LOGARITHMIC]
```

This command keyword adds a new table to the database of electron-surface interactions. Each table has data defined on a 2D grid of incident electron energy and polar angle.

The `TYPE` keyword takes the values 'scatter' or 'heating'. Data is read from the PFF file specified by the `FILE` keyword, and given a name defined by the `NAME` keyword. The optional `LOGARITHMIC` keyword specifies that logarithmic energy interpolation for the incident electrons is used. By default, linear energy interpolation is used. Note that although this command allows a heating table to be loaded, there is currently no code to implement heating in Emphasis.

`ELECTRON SURFACE INTERACT`

This command defines an electron-surface interaction model for a single electron species on a specified domain with the syntax:

```
ELECTRON SURFACE INTERACT, PRIMARY = string, SIDESET = int [... int],
  SCATTER_TABLE = string, SECONDARY = string
  [, QPMIN_FUN function-set] [, QPMIN_FLOOR real] [, ECUTOFF real]
```

The `PRIMARY` keyword defines the primary (incident) electron species, and the `SIDESET` keyword defines the spatial domain as the union of one or more sidesets. The `SCATTER_TABLE` keyword defines the data table used for the scattering, loaded with the `ELECTRON SURFACE DATABASE` command. The `SECONDARY` keyword defines the secondary electron species, which can be the same as the primary. In typical use, a secondary will be created with smaller charge and lower energy than the primary.

The optional `QPMIN_FUN` and `QPMIN_FLOOR` keywords define a minimum charge weight for creating secondaries:

$$\mathrm{qpmin}(t) = \max(\mathrm{scale} \times \mathrm{fun}(t), \mathrm{qpmin\_floor}).$$

By default, $\mathrm{qpmin}(t) = 0$, and $\mathrm{qpmin\_floor} = 0$ if only the `QPMIN_FUN` keyword is specified. This keyword uses the standard Nevada function syntax '`FUNCTION int [SCALE real]`' to define the function number and optional scale factor. The qpmin_floor value defines an absolute minimum charge value that can be created. Similarly, the optional `ECUTOFF` keyword defines a minimum energy (in MeV) for creating secondaries; the default is `ECUTOFF` $= 0$.

In principle, this command is set up to support both heating and scattering, since it is much more efficient to handle the two together if both features are needed. However, heating is not currently implemented.

`PARTICLE BALANCE [options]`

This keyword controls the dynamic load balancing of the particle workload in parallel simulations. Available `options` are, in order:

TRIGGER = real
: Controls when load balancing is attempted by comparing the specified value to the current imbalance.

TARGET = real
: Specifies the imbalance that the load-balancing algorithm attempts to achieve. A perfectly balanced particle workload has an imbalance of 1.0.

MIN_TOL_INCR = real
: Specifies the fractional amount above the imbalance obtained at the last balance before rebalancing is attempted.

MIN_PART_PER_CELL = real
: Specifies the minimum average number of particles per cell required before load balancing is attempted.

MAX_STEP_COUNT = int
: Specifies the maximum number of time steps after balancing occurs before rebalancing will be attempted again.

ZOLTAN = parameter_name "parameter_value"
: Provides advanced user control over Zoltan library parameters by accepting pairs of values: an identifier for the parameter name and a string containing the parameter value (enclosed in quotes even if the value is numeric). Refer to the Zoltan documentation for a description of the available parameters.

Time history diagnostics indicating the instantaneous and cumulative particle load imbalance are provided in parallel simulations. When dynamic load balancing is enabled the unbalanced values of these imbalance time histories are also included. The total number of elements exported between processors to balance the particle workload for the current time step is also provided as a time history diagnostic. Both absolute and normalized (to the local element count scaled by ratio of the difference between the local and average local particle count to the local particle count) values are provided.

PARTICLE MERGE

This keyword controls the merging of particles with the syntax:

```
PARTICLE MERGE, [SPECIES = name1 [... nameN]] | EXCLUDE = name1 [... nameN]],
  CYCLE INTERVAL = int, TRIGGER = int [int]
  [, options]
```

By default, the merger operates on all particle species. The `SPECIES` keyword defines a list of species on which the merger operates, while the `EXCLUDE` keyword defines a list of species on which the merger does not operate. These two keywords are mutually exclusive.

The `CYCLE INTERVAL` keyword is required and controls the frequency of merging. The `TRIGGER` keyword, also required, specifies the minimum number of particles of a single species in an element that will trigger an attempt to merge those particles. It has an optional second lower value (*trig2*) that specifies the number required to continue the merge operation after some of the particles have been rejected for merging by the merge algorithm. If *trig2* is not supplied, it is set to 90% of the `TRIGGER` value.

Available `options` are, in order:

`TARGET = int`
: Specifies the target number of particles to remain in the cell after a successful merge; its default value is 50% of the `TRIGGER` value.

`BLOCK = int [int ...]`
: By default, particles in all element blocks will be merged. However, if one or more element blocks are specified with the `BLOCK` keyword, only particles in elements of the specified element blocks will be merged.

`SUBGRIDS = int (3)`
: Provides a scale factor for sub-gridding each element, defaulting to 3. The number of actual sub-elements used will be the cube of this value.

`MAX_ITERATIONS = int (1)`
: Specifies the number of iterations used in testing for particle rejection due to thermal velocities much larger than the mean thermal velocity. Default is 1.

`VTH_FRAC = real (8.0)`
: Specifies the multiple of the thermal velocity above which a particle will be rejected due to large thermal velocity. Default is 8.0.

`QTOT_FRAC = real (5.0)`
: Specifies the rejection of heavyweight particles. Any particle whose charge is more than this value times the mean particle charge will be rejected. Default is 5.0.

`QMAX_OVER_QMIN = real (2.0)`
: Specifies the nominal ratio of the maximum and minimum charge of any particle in an element after merging has been performed. Default is 2.0.

`SPREAD = real (1.0)`
: Specifies a scale factor determining the size of the volume into which merged particles associated with a single sub-grid region will be positioned. Default is 1.0.

`QERR_TOL = real (1.0e-6)`
: Controls merge rejection due to nodal charge errors. The merge will be accepted only if the sum of charge errors over all element nodes is less than this value times the total charge in the element. Default is 10^{-6}.

`QLOW_TOL = real (0.2)`
: Controls merge rejection due to low-weight merged particles. The merge will be accepted only if the minimum charge of any merged particle is greater than this value times the mean charge of all merged particles in the element. Default is 0.2.

`EMAX_TOL = real (4.0)`
: Controls merge rejection due to high-energy merged particles. The merge will be accepted only if maximum energy of any merged particle is less than this value

times the maximum energy of any of the original, pre-merged particles in the element. Default is 4.0.

`PARTICLE PRELOAD`

Preloads the specified `BLOCKS` with the specified particle distribution using the syntax:

```
PARTICLE PRELOAD,
  BLOCK int [int ...]
  SPECIES = string
  [COUNT int int int]
  DENSITY = [real | DISTRIBUTION int]
  MOMENTUM = [X real Y real Z real | DISTRIBUTION int]
  [TEMPERATURE = real]
  [POSITION = DISTRIBUTION int]
END
```

Because the particle preload command does not adjust the electromagnetic fields, the user is responsible for ensuring the initial system is charge- and current-neutral. Since only one species can be preloaded at a time, this means the particle preload commands will come in pairs. Options for particle preload are as follows:

`COUNT int int int`

Specifies three integers whose product is the number of particles loaded per element. These integers denote the size of a 3D lattice which distributes the inserted particles uniformly throughout the element. If not supplied, the default is "1 1 1", in which case a single particle is loaded at each element's barycenter. In order to create current-neutral distributions when a mean thermal temperature is specified using the `TEMPERATURE` keyword, pairs of particles will be created (with opposite velocities) at each location.

`DENSITY real | DISTRIBUTION int`

A real value specifies the particle density in units of $\mathrm{m}^{-}3$. Optionally, a spatial distribution may be specified for the density, where the integer refers to the distribution ID that specifies a scalar density distribution for the preload.

`MOMENTUM vector | DISTRIBUTION int`

A vector value specifies the mass normalized momentum of preloaded particles in units of m/s. Optionally, a spatial distribution may be specified for the momentum, where the integer refers to the distribution ID that specifies a vector momentum distribution for the preload.

`TEMPERATURE real`

Optionally specifies the mean temperature of the Maxwellian distribution (in the rest frame) in units of eV. The default value is zero.

`POSITION DISTRIBUTION int`

Optionally specifies a spatial distribution for particle position, where the integer refers to the distribution ID that specifies a vector position distribution for the preload that perturbs the initial particle insertion location from the element lattice locations. Beware that this is not checked for consistency within the element boundaries.

`PRELOAD DISTRIBUTION int`
: Specifies a scalar or vector distribution function for the `PARTICLE PRELOAD` command using the RTC functionality. The command block must be terminated with an `END` key.

`PARTICLE SORT [options]`
: Controls the sorting of particles. Available `options` are:

  `CYCLE INTERVAL int`
  : Controls the frequency of sorting. The default is to sort every cycle.

  `CELL`
  : Sorts particles by cell (default).

  `SPECIES`
  : Sorts particles within a cell by species.

  `BLOCK = int [int ...]`
  : By default, particles in all element blocks will be sorted. If one or more element blocks are specified with this option, only particles in elements of the specified element blocks will be sorted.

6 Framework keywords for UTDEM

This section describes keywords inherited from the Nevada framework, which is shared with the ALEGRA application [3]. These keywords must be placed within the physics block (defined by the keyword `UNSTRUCTURED TD ELECTROMAGNETICS`).

6.1 Block Options

The `BLOCK` keyword defines a finite-element block, to associate a mesh block ID () with a material definition with the syntax:

```
BLOCK {int | int TO int}
  MATERIAL int
END
```

Note that `END` is required. Most block controls available in ALEGRA are not relevant to EMPHASIS, but the following may be used:

`DELETION CYCLE int`
: Specifies the cycle at which the element block is to be deleted from the problem.

`DELETION TIME real`
: Specifies the time at which the element block is to be deleted from the problem.

`DELETE DATA`
: Deletes the element block by deleting all vertex, edge, face, and element data associated with the block. The coordinates are reset to their original value and the block is filled with void.

`DELETE TOPOLOGY`
: Deletes the element block by deactivating all vertices, edges, faces, and elements associated with the block.

6.2 Function Specification

The `FUNCTION` keyword allows the user to define a function, of the form $f(x)$, from a selection of commonly used options. The specified function(s) will be referenced by an integer identifier when used by other input deck commands. The following functions are available:

```
FUNCTION int [LINEAR (default) | SPLINE]
     real real
     real real
     ...
END
```

Defines a function as a table of real ordered $(x, f(x))$ pairs. A valid function must specify at least two pairs, and the `END` keyword is required.

A default, predefined, constant function is provided with ID 0, equivalent to

```
FUNCTION 0
  -REAL_MAX/2.  1.
   REAL_MAX/2.  1.
 END
```

Because this function uses ID 0, any user-specified function cannot reuse the identifier 0 (any positive integer may be used).

The optional keywords, `LINEAR` and `SPLINE`, define how values between the table points are calculated. With the default `LINEAR` interpolation, values are interpolated linearly between the table points and extrapolated to zero order outside the table endpoints. This gives a C^0-continuous result bounded by the table values. For `SPLINE` interpolation, Catmull-Rom cubic splines are used [3], which are guaranteed to pass through the table points, and will be C^1-continuous within the specified range. However, interpolated function evaluations are not bounded by the table values, so undershoots and overshoots will be produced. Addition of "control" points to the table can help control the behavior of the spline near sharp transitions or discontinuities.

`FUNCTION int GAUSSIAN [, SCALE real (1.0)] [, SHIFT real (0.0)] [, WIDTH real (1.0)]`

$$f(x) = \text{scale} \times \exp\left(-\frac{(x - \text{shift})^2}{\text{width}^2}\right).$$

`FUNCTION int DOUBLE EXPONENTIAL [, SCALE real (1.0)] [, SHIFT real (0.0)] [, ALPHA real (0.0)] [, BETA real (0.0)]`

$$f(x) = \text{scale} \times (\exp\left[-\text{alpha} \times (x - \text{shift})\right] - \exp\left[-\text{beta} \times (x - \text{shift})\right]).$$

`FUNCTION int SINE [, SCALE real (1.0)] [, SHIFT real (0.0)] [, FREQUENCY real (0.0)]`

$$f(x) = \text{scale} \times \sin\left(\text{frequency} \times x + \text{shift}\right).$$

`FUNCTION int SINE SQUARED [, SCALE real (1.0)] [, SHIFT real (0.0)] [, WIDTH real (0.0)]`

$$f(x) = \text{scale} \times \sin^2\left(\frac{\pi(x - \text{shift})}{\text{width}}\right), \quad 0 < x < \text{width}.$$

`FUNCTION int TRIANGLE [, SCALE real (1.0)] [, SHIFT real (0.0)] [, WIDTH real (0.0)]`

$$f(x) = \begin{cases} \text{scale} \times \left(\dfrac{x - \text{shift}}{0.5 \times \text{width}}\right), & 0 \le x \le \text{width}/2, \\ \text{scale} \times \left(1 - \dfrac{x - \text{shift} - 0.5 \times \text{width}}{0.5 \times \text{width}}\right), & \text{width}/2 \le x \le \text{width}. \end{cases}$$

6.3 Time Step Controls

The following time step controls are available in addition to the `TIME STEP MOD` option described in Section 4.

`GRADUAL STARTUP FACTOR real (0.01)`

Specifies the factor by which the initial physics-based time step should be multiplied. The default value is 0.01.

This has the effect of gradually marching into an abrupt transient. This value should always be greater than zero and less than or equal to 1.0; for UTDEM, the value is normally set to 1.0.

MAXIMUM INITIAL TIME STEP real (0.0)
: Specifies the maximum initial time step. It is useful where unusual transients would otherwise result in an instability in the starting time step. Ignored if set to zero.

MAXIMUM TIME STEP LIMIT real (1.e30)
: Specifies a maximum time step value that will never be exceeded during the simulation.

MAXIMUM TIME STEP RATIO real (1.2)
: Specified the maximum ratio by which a time step may grow from one cycle to the next.

MINIMUM TIME STEP real (1.e-20)
: Specifies the minimum permissible time step. If the stable time step is computed to be less than this value, the calculation will cease and write the final output records for a normal completion.

CONSTANT TIME STEP real
: Specifies a constant time step for the entire simulation.

If no CONSTANT TIME STEP is specified, UTDEM determines a time step based on the Courant stability criteria. This can provide a starting point for setting the time step. However, for the unconditionally stable SECOND ORDER formulation, a much larger time step may be utilized. If this is done, the solution will remain stable but the required conjugate-gradient iterations required for system solution at each cycle will increase. Generally, for large simulation times, the overall simulation CPU time will be reduced by increasing the time step.

6.4 Initial Refinement

The DOMAIN keyword defines domain-level (i.e., global) options with the syntax:

```
DOMAIN
  [options]
END
```

The domain options specify behavior that is not broken down to the block level, so will apply to multiple blocks. For EMPHASIS, only the initial refinement option available in ALEGRA is relevant. This capability allows the resolution of a GENESIS file to be increased inline, after the parallel decomposition, which could be useful when attempting to run with very large and/or detailed mesh files. For each level of refinement, it will increase the resolution of a tetrahedral or hexahedral mesh by a factor of eight by splitting each refined edge into two new sub-edges. At present, it cannot refine hybrid meshes.

The syntax for initial refinement is:

```
DOMAIN
  MAXIMUM LEVELS = int
  INITIAL REFINEMENT
    [ geometry ]
  END
END
```

The `MAXIMUM LEVELS` keyword specifies the number of levels of refinement to be performed, and is required. Options for `geometry` within the `INITIAL REFINEMENT` block are:

`ALL BLOCK`
: All blocks will be refined to the highest refinement level.

`BLOCK BOUNDARY`
: Elements on the boundaries of all element blocks will be refined to the highest refinement level.

`SIDESET int`
: Refines the sideset to the highest refinement level.

`SIDESET int SPHERE real vector`
: Refines the sideset to the highest refinement level while mapping *surface* nodes to a sphere of radius `real` centered at position `vector`.

`SIDESET int CYLINDER real vector vector`
: Refines the sideset to the highest refinement level while mapping surface nodes to a cylinder of radius `real` with axis defined by the `vector` pair.

`SIDESET int RECONSTRUCT`
: Reconstructs a local smooth surface through the sideset nodes using approximate surface normals at these nodes.

`BLOCK int`
: Refines the specified element block to the highest refinement level.

`BLOCK int, LEVEL int`
: The element block specified will be refined to the refinement level specified. The specified refinement level must be less than the maximum level set by the `MAXIMUM LEVELS` keyword.

Note that some uses of these options will generate irregular meshes, where an element may neighbor another elements that is refined to one level higher or lower than itself.

6.5 Initial Conditions

```
USER DEFINED INITIAL CONDITION, variable
                            [, BLOCK int int ... ]
                            [, MATERIAL int int ... ]
"
  [c-code declarations]
  [c-code initializations]
  [c-code conditionals]
  [c-code assignments]
"
END
```

This provides a general method for initializing known field variables on the mesh, using the runtime compiler (RTC) feature (see Appendix E for more information on writing RTC functions and for using aprepro with RTC functions). The `variable` is a name, such as `ELECTRIC_FIELD_PROJECTION`. Any material, element, node, edge, or face variable (that is a scalar or space-dimensional vector field) may be used, but the name must correspond to the name in the RUNID.out file (including underscores), and is not necessarily the same name as is printed in the RUNID.exo file that may have the material number or the vector/tensor component designator appended to it. For some variables, this feature may give the user more than enough rope to hang themselves with, so if you have trouble with this feature contact alegra-help@sandia.gov to get a developer to assist you. Beware in particular that no attempt is made to ensure that $\nabla \cdot \mathbf{D} - \rho$ or $\nabla \cdot \mathbf{B}$ are zero.

As a C-language function, the double-quoted body has available one input array of coordinates, `coord`, and one output array of field values, `field`. The input coordinates are set prior to executing the quoted function, appropriate to the output variable centering (node coordinates for node variables, cell midpoint coordinates for element variables, etc.). The user may overwrite the `coord` array variables, but they are only RTC temporary variables, and are deleted after execution of the RTC function. More importantly, the `field` array variables that the user defined RTC function acts on are also RTC temporary variables *initialized to zero*. They are copied over to the appropriate EMPHASIS variables after RTC function execution.

The function is expected to use the coordinates in some way to set the return value. The return value is an array whose length depends on the type of variable. A scalar variable has length one, a vector has length 3 (for 3D), etc. Note that in the special case of edge- and face-centered variables, the user specifies a vector field, which is projected onto the edges/faces to determine the scalar values.

The optional `BLOCK` keyword can be used to trigger the initial condition function only for certain element blocks, specified by their ids. For nodal variables, the function is called if the node touches any element that is contained in the block(s) specified.

The optional `MATERIAL` keyword can be used to apply the initial condition function to only those elements which have the given material(s) present. Nodal variables will be set if they touch an element with the designated material(s) present. The integers must specify material ids that are specified in the input deck.

A presentation of the capabilities and limitations of runtime compiled 'C' functions is included in Appendix E.

Example:

```
$ The ELECTRIC_FIELD_PROJECTION is an edge-based scalar,
$ so specify the vector field to be projected onto the edges
user defined initial condition, ELECTRIC_FIELD_PROJECTION
"
  double pi = acos(-1.0);
  field[0] = 0.0;
  field[1] = 0.0;
  field[2] = sin(pi*coord[0])*sin(2.0*pi*coord[1]);
"
end
```

6.6 Periodic Boundary Conditions

```
PERIODIC BC, SIDESET int, TRANSLATE, X float Y float Z float, SIDESET int,
  TOLERANCE float (1.e-5)
```

This applies a periodic boundary condition between the two, generally planar, sidesets given. The TRANSLATE vector points from the first SIDESET to the second and must start and end exactly on those sideset planes. The TOLERANCE parameter is often required to be significantly smaller than the default 1.e-5, generally 1.e-8 is more effective.

The user is responsible for creating a mesh on the two periodic surfaces such that all the edges align, which is not always a trivial task. For example, unless measures are taken, CUBIT will almost certainly not align in the required manner. Assuming the proposed periodic surfaces are nice flat rectangles, such as the ends of a box, a method in CUBIT which has been show to create the desired aligned surface mesh is the following. After setting appropriate mesh size on the volume or surfaces, mesh the surfaces with "surf 1 2 scheme map" then "mesh surf 1 2" which will create nice quads. Then split these into triangles by first setting "set qtri split 4" followed by "qtri surf 1 2". This will place a node at the center of each quad and then split into 4 triangles.

7 Simulation Control Keywords

These keywords are framework keywords [3] which are required for a successful UTDEM simulation and appear outside the `UNSTRUCTURED TD ELECTROMAGNETICS` block. An example of typical simulation control keywords is shown in Figure 3.

```
termination time = 1.e-8

emit screen, cycle interval = 1
emit plot, cycle interval = 10
emit hisplt, cycle interval = 1

plot variable
  electric_field
  econ
end
```

Figure 3. Typical simulation and output control keywords.

7.1 Edgeset Keyword

`EXODUS EDGE SETS (int, int, ...)`

Specifies a list of sideset ids that exist in the Genesis file that are to be converted to Nevada edgesets. The list of sideset ids must be enclosed in parentheses. Virtual edgesets defined by a `PATH` keyword should not be included in this list; the sidesets on this list are the ones encoded by the preprocessor or the mesh generation software for the purpose of being converted to edgeset(s) by the Nevada framework. This step is necessary because the standard ExodusII file format has no notion of a list of edges.

7.2 Simulation Termination Keywords

The following keywords control termination of a simulation. If multiple termination keywords are specified, the simulation will terminate when any of the criteria are satisfied.

`[EXACT] TERMINATION TIME real`

Total time for which to run the simulation. If the optional `EXACT` keyword is specified, the time steps for the last ten cycles will be adjusted as necessary to ensure that the calculation terminates at the exact time specified. Otherwise, by default the termination time may be overshot by some fraction of a time step.

`TERMINATION CYCLE int`
: Total cycles for which to run the simulation.

`TERMINATION CPU real`
: Specifies the CPU time, in seconds, at which the calculation is to terminate. This termination control must be used together with a `TERMINATION TIME` and/or a `TERMINATION CYCLE` keyword.

7.3 Restart Keywords

Controls for restarting from an existing simulation are described briefly below.

`RESTART DUMPS int (2)`
: Retain the last number of restart dumps, defaults to 2.

`READ RESTART DUMP int`
: Restart simulation by reading the restart dump with the given index. Restart will fail if no dump with the given index exists. If the special value -1 is given, the latest available restart dump is used by consulting the dump list file, *problem_ name*.dpl. If this file is empty or does not exist, a new simulation is started.

`READ RESTART TIME real`
: Restart simulation at the closest restart dump to the given time. If a negative time is given, a new simulation is started.

The frequency at which restart dumps are written is controlled by the `EMIT RESTART` keyword. A restart dump is also created at the end of the simulation.

Upon restart, plot-dump data is appended to the *problem_ name*.exo file. However, rather than appending to the *problem_ name*.his file, additional files are created with names *problem_ name*.his_0, *problem_ name*.his_1, etc. These must be concatenated together with the original *problem_ name*.his to generate the entire time history.

7.4 Output Keywords

`EMIT SCREEN, output-frequency`
: Print status line to standard out at the requested frequency.

`EMIT PLOT, output-frequency`
: Write plot variables to Exodus file at the requested frequency.

`EMIT HISPLOT, output-frequency`
: Write global variables to hisplt file at the requested frequency.

EMIT RESTART, output-frequency
Write restart dumps at the requested frequency.

The output-frequency specification takes one of the following forms:

TIME INTERVAL real [output-range]
Output occurs at the specified simulation time interval. Because the simulation time step is not modified, the actual write time may overshoot the requested interval by some fraction of a time step.

EXACT TIME INTERVAL real [output-range]
Output occurs at the exact simulation time interval specified. The time step is modified such that output is triggered at the requested interval.

CYCLE INTERVAL int [output-range]
Output occurs at the specified cycle interval.

WALL CLOCK INTERVAL number | hms [output-range]
Output occurs at the specified wall clock interval. Wall clock time must be specified either as an integer or real number of seconds, or in the form `12h 34m 56s`.

NUMBER int [output-range]
Output occurs the specified number of times over the course of the simulation. If an `output-range` is specified, then the number of output triggers will occur within that range.

The `output-range` option takes one of the following forms:

FROM number [TO number]
Specifies beginning (and optionally ending) values that form a range. The units of the range are assumed to be the same as the units of the `output-frequency` specification.

FROM TIME real [TO real]
Specifies a simulation time range.

7.5 Plot Variables

Variables to be written to the Exodus [9] plot file are specified with the following syntax:

```
PLOT VARIABLES
  [ALL VARIABLES]
  [NO DEFAULT OUTPUT]
  [NO REGION VARIABLES | ALL REGION VARIABLES]
```

```
  [NO MATERIAL GLOBALS | ALL MATERIAL GLOBALS]
  [NO UNDERSCORES]
  registered-variable-name, [modifier]
  registered-variable-name, [modifier]
  ...
END
```

Valid plot variables for UTDEM are shown in Table 1.

Table 1. Plot Variables for UTDEM

Variable Name	Type	Explanation
`ELECTRIC_FIELD`	`vector`	Electric field (**E**).
`MAGNETIC_FIELD`	`vector`	Magnetic field (**H**).
`MAGNETIC_FLUX_DENSITY`	`vector`	Magnetic flux density (**B**).
`CUR_DEN`	`vector`	Current density for external J-sources.
`ECON`	`scalar`	Electrical Conductivity (σ), from Simple Electrical, RIC Electrical and Breakdown Electrical material models, and external J-sources.
`ELECTRON_CONCENTRATION`	`scalar`	Electron concentration, from HP Gas material model.
`NEGATIVE_ION_CONCENTRATION`	`scalar`	Negative ion concentration, from HP Gas material model.
`AVALANCHE_RATE`	`scalar`	Avalanche rate, from HP Gas material model.
`ATTACHMENT_RATE`	`scalar`	Attachment rate, from HP Gas material model.
`PERMITTIVITY`	`scalar`	Electric Permittivity (ϵ) from material model.
`PERMEABILITY`	`scalar`	Magnetic Permeability (μ) from material model.
`RELUCTIVITY`	`scalar`	Magnetic Reluctivity (μ^{-1}) from material model.
`SPEED_OF_LIGHT`	`scalar`	Speed of light from material ($(\epsilon\mu)^{-1/2}$).
`IMPEDENCE`	`scalar`	Real intrinsic impedence of the material ($\sqrt{\frac{\mu}{\epsilon}}$)

Additional valid plot variables for UTDEM_PIC are shown in Table 2. Averaged variables, prefixed with `AVE_`, are quantities averaged over cycle intervals. An individual particle species' charge density is separately computed only if it is referenced using the `RHO_species` or `AVE_RHO_species` plot variables, or explicitly defined with the `REGISTER_DENSITY` option of the `DEFINE SPECIES` input command (34). If the `AVE_RHO_species` plot variable is listed then EMPHASIS calculates the `RHO_species` variable, but the non-averaged charge density is only output if specifically requested in the plot variable list. If there are any remaining particle species whose charge densities are not computed separately due to one of these two reasons, their charge densities are combined and available in aggregate using the

CHARGE_DENSITY plot variable. Note that multiple PMC symmetry planes must be defined by separate side sets for the PIC_CURRENT variable to be accurate.

Table 2. Plot Variables for UTDEM_PIC

Variable Name	Type	Explanation
PIC_CURRENT	vector	Particle current applied to the mesh.
AVE_PIC_CURRENT	vector	Average PIC current.
RHO_species	scalar	Charge density of species.
AVE_RHO_species	scalar	Average charge density of species.
AVE_ELECTRIC_FIELD	vector	Average electric field.
AVE_MAGNETIC_FIELD	vector	Average magnetic field.
CHARGE_DENSITY	scalar	Aggregate species charge density.
DIVD_M_RHO	scalar	$\nabla \cdot \mathbf{D} - \rho$.

For a complete list of available plot variables and options, consult the Alegra manual [3].

Time-history variables to be written to the hisplt file are specified with the following syntax:

```
HISTORY PLOT VARIABLES
  [NO DEFAULT OUTPUT]
  [NO REGION GLOBALS]
  [NO MATERIAL GLOBALS]
  [NO UNDERSCORES]
  registered-variable-name, [modifier]
  registered-variable-name, [modifier]
  ...
END
```

Time-history data for all specified global variables, and for spatial locations specified in the TRACER POINTS keyword, will be written to the hisplt database file, *problem_ name*.his. By default, all global variables will be written. Tracers located in wedge elements will not be found.

7.6 Linear Solver Keywords

Solution of an EMPHASIS problem requires specification of a linear solver, which uses the AZTEC syntax [12]. A typical example is:

```
aztec
  solver,   cg
```

```
  precond,  none
  scaling,  sym_diag
  output,   none
  tol      = 1.e-9
  polynomial order, 1
  max iterations, 1000
end
```

This specifies the conjugate gradient (cg) method with no preconditioning, but with symmetric diagonal scaling. No output is requested from AZTEC after each solve to a tolerance level of 1.e-9, with a maximum number of cg iterations set to 1000 (default is 500). The `POLYNOMIAL ORDER` should always be set to "1" for efficiency.

The next example again specifies cg, but with jacobi preconditioning and without scaling. The convergence norm is set to be relative to the rhs rather than default, which is the initial residual. If large conductivity values (on the order of 1 or higher) exist in the simulation, the convergence norm must be set to rhs to achieve convergence due to numerical considerations.

```
aztec
  solver,   cg
  precond,  jacobi
  scaling,  none
  convergence norm, rhs
  output,   last
  tol      = 1.e-9
  polynomial order, 1
end
```

For convergence with very large time steps, the Multi-Level (ML) preconditioner should be specified. The following settings are typical to achieve a successful ML solution with limited testing. Others are possible and perhaps even desirable. The Alegra-MHD manual [2] may provide some guidance for advance ML settings.

```
aztec
  solver,   cg
  precond,  jacobi
  output,   none
  tol      = 1.e-9
  polynomial order, 1
  multilevel
    fine sweeps = 1
    fine smoother = Hiptmair
    coarse sweeps = 6
    coarse smoother = Hiptmair
```

```
    multigrid levels = 10
    interpolation algorithm = AGGREGATION
    smooth prolongator
    hiptmair subsmoother = MLS
  end
end
```

This final example mimics as closely as possible using Aztec the solver technology used in the legacy unstructured code:

```
aztec
  solver,   cg
  precond,  dom_decomp
  subdomain solver, icc
  type overlap, symmetric
  output,   none
  tol      = 1.e-15
  polynomial order, 1
end
```

7.7 Debug Mode Keyword

Specific code debug information at run time can be requested using the following syntax:

`DEBUG MODE: debug-opt`

Some relevant values for `debug-opt` are:

`LOCATION`
: Provides a quick way to observe in the standard out stream progress through the code execution, and in particular to see where the most time is being spent.

`SIGNALS | FPE`
: These will catch and report events such as floating-point exceptions.

`AUTO ITS BEAM SETUP`
: This mode is specific to PIC. It requests detailed data to compare the auto-fit ITS beam setup with the corresponding ITS simulation that built the input PFF file. The results are placed in the file *auto_its_beam.dat*.

The *auto_its_beam.dat* file contains a table with an entry for each ITS subsurface fitted to at least one emitting face of an Emphasis beam-emission sideset, with the columns listed in Table 3.

Table 3. Column listing with descriptions for the *auto_its_beam.dat* debug file.

Column	Description
EmSurf#	Emphasis emission surface number (1–N)
SideSet	Emphasis Sideset ID
ITSsurf#	ITS surface # in the PFF file
ITSsubsurf#	Subsurface # of the ITS surface: $1 - \text{Nudiv} \times \text{Nvdiv}$, where Nudiv and Nvdiv are the two ITS surface subdivision parameters, and the 1-D # is $v + \text{Nvdiv} \times (u - 1)$ (ITS ordering).
#-cells	# emitting faces fitted to the subsurface
ITS-emitfrac	Total emission fraction (per single source particle) for the subsurface
area	Sum of Emphasis emission face areas fitted to subsurface
emDensity	emitfrac/area

8 Material Models

8.1 Material Block

Defines a model or models for a material with the syntax:

```
MATERIAL int
  MODEL int
  ...
END
```

Relates the material identified with the index `int` to material `MODEL`(s). For UTDEM, only one model applies to each material.

8.2 Material Model Block

Defines a material model with the syntax:

```
MODEL int model-name
  [PARAMETER value]
  [PARAMETER value]
  ...
END
```

Relates the material model identified with the index `int` to a specific model type defined by the name `model-name` and defines parameters specific to that model.

8.3 Material Models for UTDEM

The following models are valid `model-name` options for UTDEM. Every model will specify `EPS`, the relative permittivity of the material (ε), `MU`, the relative permeability of the material (μ), and `SIGMA` (or `SIGMA0`), the conductivity of the material (σ). Models that have a `NTIMESTATES` parameter *must* have it set to 3 for proper UTDEM functionality. If not, the code will halt and notify the user.

8.3.1 Simple Electrical

```
SIMPLE ELECTRICAL
  EPS real
  MU real
  SIGMA real
  [NDOF int]
  [NTIMESTATES 3]
END
```

This is the simplest electrical material model, specifying constant values for ε, μ, and σ. A simple usage example is

```
MODEL 1 SIMPLE ELECTRICAL
  EPS 2.
  MU 1.
  SIGMA 1.e-3
END
```

The NDOF and NTIMESTATES keywords are only needed for the specific case of UTDEM PIC simulations with multiple element blocks, some using Simple Electrical, and others using HP Gas Electrical. In this case NDOF and NTIMESTATES must have the same values for both models. In particular, NTIMESTATES = 3 to consistently handle time-dependent conductivity.

8.3.2 Breakdown Electrical

```
BREAKDOWN ELECTRICAL
  EPS real
  MU real
  SIGMA0 real
  THRESHOLD real
  SIGMA_BKDN real
END
```

The BREAKDOWN ELECTRICAL model is a simple material breakdown model whereby the electric field in each element is monitored and if it reaches THRESHOLD V/m, the conductivity in that element is changed to SIGMA_BKDN, where it remains for the remainder of the simulation. The UPDATE MAT STATE keyword must be used with this model. A simple usage example is

```
MODEL 2 BREAKDOWN ELECTRICAL
  EPS 2.
  MU 1.
  SIGMA 0.
  THRESHOLD 2000.0
  SIGMA_BKDN 1.e7
END
```

8.3.3 HP Gas Electrical

```
HP GAS ELECTRICAL
  EPS real
  MU real
  SIGMA0 real
  DENSITY real
  WATER_FRACTION real
  [GASNAME "string"]
  [XIEV real]
  [NDOF int]
  NTIMESTATES 3
  [SCATTER "string"]
  [NKEBINS int]
  [KEMIN real]
  [KEMAX real]
  [NSUBCYCLE int]
  [LIMIT_PROB real]
  [ELECTRON_EMIN real]
END
```

The HP GAS ELECTRICAL model is a high-pressure gas chemistry model containing the additional parameters DENSITY, WATER_FRACTION, XIEV, and NDOF, and optional parameters to enable angular scattering of electrons from the gas (see below). The gas density is specified in MKS units, kg/m^3. The XIEV and NDOF parameters are optional. The parameter XIEV can be used to override the default mean ionization energy of the gas (34 eV). If not specified, NDOF will default to the number of nodes in an element, in which case the model variables will be node based. Setting NDOF to one will make the model variables element based. The optional GASNAME parameter can be used to specify the gas. The model will read the gas parameters from the file "gasname.dat" in the run directory. If not specified, the default gas model is the ITT model for air which can also be specified explicitly with the string `itt_air`. Note that `WATER_FRACTION` is a *required* parameter for `itt_air`, but is not used for an explicitly requested gas file.

In a PIC simulation, the `HP GAS ELECTRICAL` model can used in combination with the `GAS DRAG` command to enable interaction of electrons with the background gas. This applies a drag force to slow down the electrons, but does not change their direction. The `SCATTER` parameter enables angular scattering of the electrons. The string takes exactly the same form as the mixture parameter for the `KINETIC GAS ELECTRICAL` model, and uses the same cross section data file (see below). The optional parameters `NKEBINS`, `KEMIN`, and `KEMAX` correspond to the `KINETIC GAS ELECTRICAL` parameters `NEBINS`, `EMIN` and `EMAX` respectively, and with the same defaults (renamed here to avoid confusion with electric field bins). Elastic scattering is enabled with a table of the total cross section (summed over all processes) for each gas on the specified energy grid. The optional `NSUBCYCLE` and `LIMIT_PROB` keywords provide user control to reduce the maximum probability of interaction, $P = n\sigma v\Delta t$, for very high density gas. By default `NSUBCYCLE` = 1. If a value greater than 1 is entered, the scattering is done `NSUBCYCLE` times per PIC timestep, with $\Delta t_{\mathrm{scat}} = \Delta t_{\mathrm{PIC}}/$`NSUBCYCLE`. The `LIMIT_PROB` explicitly limits P by brute force for all processes at all energies.

The `ELECTRON_EMIN` parameter (in eV) requests that electrons be killed if their energy falls below this value. Typically, this value will be near the ionization threshold for the gas. Killing electrons in space will corrupt $\nabla \cdot \mathbf{D} - \rho$ diagnostics, but here the electrons are presumably being killed where the conductivity is non-zero. In this case the $\nabla \cdot \mathbf{D} - \rho$ diagnostics are already corrupted, and additional errors from killing particles are reduced over time by being conducted away.

A simple usage example is

```
MODEL 3 HP GAS ELECTRICAL
  EPS 2.
  MU 1.
  SIGMA0 0.
  DENSITY 1.23
  WATER_FRACTION 0.02
  NDOF 1
  NTIMESTATES 3
END
```

8.3.4 Foam Electrical

```
FOAM ELECTRICAL
  EPS real
  MU real
  SIGMA0 real
```

```
  COEF1 real
  COEF2 real
  COEF3 real
  DENSITY real
  NDOF int
  NTIMESTATES 3
END
```

The `FOAM ELECTRICAL` model is an empirical radiation-induced conductivity (RIC) model for foam that takes the following form:

$$\sigma = \sigma_0 + \varepsilon \left(c_1 \dot{\gamma}^2 + c_2 \dot{\gamma} + c_3 \right), \tag{8.10}$$

where σ is the conductivity in Mho/m, σ_0 is the dark conductivity in Mho/m, ε is the permittivity in F/m, c_1, c_2, and c_3 are the coefficients `COEF1`, `COEF2`, and `COEF3` in (Mho/F)/(Rad/s), and $\dot{\gamma}$ is the dose rate in Rad/s.

If the `DENSITY` (pounds/ft^3) is specified as either 5 or 10, the correct coefficients will be supplied by the code for polyethylene foam and any supplied coefficient values will be ignored. Supplying other densities will generate an error. If `DENSITY` is not supplied, arbitrary coefficients can be supplied and will be used. In either case, the user must supply the appropriate dielectric constant. Typical values for polyethylene foam are $\varepsilon = 1.105\varepsilon_0$ for a density of 5 lbs/ft^3, and $\varepsilon = 1.2\varepsilon_0$ for a density of 10 lbs/ft^3. Table 4 shows the code-stored coefficients for 5 and 10 pound foams.

Table 4. Values for `FOAM ELECTRICAL` model coefficients

Density (lbs/ft^3)	5	10
c_1	1.963×10^{-12}	1.347×10^{-12}
c_2	5.027×10^{-1}	8.265×10^{-2}
c_3	-7.863×10^{8}	-2.228×10^{7}

8.3.5 HP Foam Electrical

```
HP FOAM ELECTRICAL
  EPS real
  MU real
  SIGMA0 real
  DENSITY real
  WATER_FRACTION real
  GASNAME "string"
  VF real
  XIEV real
```

```
  NDOF int
  NTIMESTATES 3
END
```

The HP FOAM ELECTRICAL model is an *experimental* foam model based on the field-exclusion foam model of Stringer and Dumcum [11]. This model is similar to HP Gas Electrical with one additional parameter VF, the volume fraction of gas in the foam. The gas definition here refers to the gas in the foam "bubbles". SIGMA0 is the background conductivity of the gas and EPS is the dielectric constant of the solid portion of the foam forming the bubble walls.

8.3.6 Kinetic Gas Electrical

```
KINETIC GAS ELECTRICAL
  EPS real (1.0)
  MU real (1.0)
  SIGMA0 real (0.0)
  P_TORR real
  MIXTURE "‘/Estring"
  [NO_ATTACHMENT]
  [FRACTIONAL IONIZATION fractional-ionization-specification]
  [PRIMARY string]
  [SECONDARY string]
  [EMIN real]
  [EMAX real]
  [NEBINS int]
END
```

The KINETIC GAS ELECTRICAL model is a gas chemistry model that handles collisions between electrons and molecules in a background gas kinetically. Only the MIXTURE and P_TORR parameters are required. If not supplied, EPS, MU, and SIGMA0 default to values of 1.0, 1.0, and 0.0, respectively. P_TORR is the pressure of the gas at standard temperature, in torr. The MIXTURE parameter is a string containing space-delimited pairs, each pair comprised of a string for the name of a gas molecule followed by a real value representing this molecule's fraction of the mixture (by volume). Note that the total of the fractions for all molecules in the mixture must sum to one. If the fraction for the last gas is not specified, its value will be computed automatically subject to this constraint. For example, the following MIXTURE specification could be used as an approximate model for dry air: "N2 0.79 O2".

The NO_ATTACHMENT parameter is used to indicate that all attachment interactions should be neglected; if not supplied, attachment will be modeled. The PRIMARY and SECONDARY parameters provide the species names of two electron species that are used in the material model when treating collisions. These are the only particles that will be considered as incident electrons for collisions with the background gas molecules. The primary electron species is used for electrons which come from other sources in the problem, such as beam or field emission surfaces. The secondary electron species is used by the material model to create the electron byproducts of ionizing collisions with the background gas. If either is unspecified, both default to the string `electron`. To compute the probability of any interaction with the gas, the energy-dependent cross sections of the various interactions are interpolated from a table that is constructed from data in the Cross Section Database (see the `CROSS SECTION DATABASE` command for more details). Three input parameters are used to control the construction of the table. The `EMIN` and `EMAX` parameters control the upper and lower energy bounds of the table, respectively, and are specified in eV. Default values are 10^{-1} and 10^{5}. The `NEBINS` parameter controls the number of bins into which the energy range of the table is logarithmically divided, and its default value is 600. Consequently, the default values for these three parameters provide 100 bins/decade.

The `FRACTIONAL IONIZATION` parameter is used to help control exponential growth in particle count by providing a time-dependent function specifying minimum charge magnitude q_{min} for secondary electrons created by an ionizing collision. If the magnitude of the charge of a secondary electron to be produced by an ionizing collision, $|q_{\mathrm{sec}}|$, is less than q_{min}, an ionization fraction F_{ion} is computed using

$$F_{\mathrm{ion}} = |q_{\mathrm{sec}}|/q_{\mathrm{min}},$$

subject to the additional constraint that

$$F_{\mathrm{ion}} \geq F_{\mathrm{base}}.$$

The probability of the collision is then reduced by F_{ion}, and if the collision still occurs, q_{sec} is increased $1/F_{\mathrm{ion}}$. There are two forms available for `fractional-ionization-specification`:

`FUNCTION int [, SCALE = real] [, BASELINE = real]`
: Provides a time-dependent specification of q_{min} (in Coulombs). The `SCALE` parameter defaults to 1.0 if not provided.

`real [, SCALE = real] [, BASELINE = real]`
: Specifies a constant value, such that q_{min} is the product of the constant and the value of the `SCALE` parameter.

In either case, the optional `BASELINE` parameter specifies the value of F_{base}, which defaults to 0.0. Finally, if the `FRACTIONAL IONIZATION` keyword is not provided, the fractional ionization model is not used.

A database of gas cross section data must be available to the `KINETIC GAS ELECTRICAL` material model. See the `CROSS SECTION DATABASE` command for how to specify this file. In addition to the particle species specified by the `PRIMARY` and `SECONDARY` parameters, any gas molecules in the gas mixture that have ionizing interactions will require a corresponding positive ion species. Similarly, any gas molecules with attachment interactions will require a corresponding negative ion species (assuming the `NO_ATTACHMENT` parameter was not specified). By convention, the model assumes the names of any required positive or negative ion species will be the name of the gas molecule with the suffix "_P" or "_N" appended, respectively. If any of these particle species needed by the model are not explicitly defined using the `DEFINE SPECIES` command above, the model will automatically attempt to define them.

8.3.7 RIC Electrical

```
FOAM ELECTRICAL
  EPS real (default 1.0)
  MU real  (default 1.0)
  SIGMA0 real (default 0.0)
  COEFFICIENT real
  EXPONENT real
  NDOF int
  NTIMESTATES 3
END
```

The `RIC ELECTRICAL` model is an empirical radiation-induced conductivity (RIC) model which takes the following form:

$$\sigma = \sigma_0 + \varepsilon K \dot{\gamma}^e,$$

where σ is the conductivity in Mho/m, σ_0 is the dark conductivity in Mho/m, ε is the permittivity in F/m, K is specified by `COEFFICIENT` in (Mho/F)/(Rad/s), $\dot{\gamma}$ is the dose rate in Rad/s, and e is specified by `EXPONENT`. Typical values for kapton are $\varepsilon = 3.5\varepsilon_0$, $K = 3.23 \times 10^{-6}$, and $e = 0.95$.

A simple usage example is:

```
MODEL 4 RIC ELECTRICAL
  EPS 2.
  MU 1.
  SIGMA0 0.
  COEFFICENT 3.23e-6
  EXPONENT 0.95
  NTIMESTATES 3
END
```

8.3.8 Face PML

This material model is only supported by the Crank-Nicolson time integrator.

```
FACE PML
```

```
  EPS real (default = 1.0)
  MU real  (default = 1.0)
  SIGMA real (default = 0.0)
  NORMAL0_X real
  NORMAL0_Y real
  NORMAL0_Z real
  INTERFACE_START0 real
  LAYER_THICKNESS0 real
  CENTER_X real (default = 0.0)
  CENTER_Y real (default = 0.0)
  CENTER_Z real (default = 0.0)
  POLYNOMIAL_GRADING real (default = 4.0)
  SCALE_FACTOR real (default = 0.01)
  NDOF int
  NTIMESTATES 3
END
```

`FACE PML` is a one dimensional sponge layer used to truncate domains. For some physical, 3D domain call Γ a planar surface which we wish to model as an "open boundary." We use the Unsplit Perfectly Matched Layer (UPML) technique and therefore attenuate outgoing waves by using a non-physical, anisotropic, dispersive material. The relative permittivity ϵ_r, relative permeability μ_r, and electrical conductivity σ are passed to inputs `EPS`, `MU`, `SIGMA` respectively. If these parameters are unspecified then vacuum conditions are assumed, i.e. $\epsilon_r = \mu_r = 1$, and $\sigma = 0$. This layer should be impedence matched to the physical domain. That is,

$$Z(\omega) = \sqrt{(\epsilon + i\omega\sigma)^{-1}\mu} \tag{8.11}$$

should be the same in the PML as the physical domain. In practice one selects these parameters identically to the physical domain although in principal only the ratio must remain constant. Impedance matching is only guaranteed when truncating `SIMPLE ELECTRICAL` materials with `SIGMA =0.0`.

Parameters `NORMAL0_X`, `NORMAL0_Y`, `NORMAL0_Z` define the outward normal for Γ. The resulting vector need not be unit length, i.e the unit outward normal $\mathbf{n}$ is given by

$$\mathbf{n} = |\tilde{\mathbf{n}}|^{-1}\tilde{\mathbf{n}}, \qquad \tilde{\mathbf{n}} = (\texttt{NORMAL0\_X},\texttt{NORMAL0\_Y},\texttt{NORMAL0\_Z}). \tag{8.12}$$

The vector

$$\mathbf{c} = (\texttt{CENTER\_X},\texttt{CENTER\_Y},\texttt{CENTER\_Z}) \tag{8.13}$$

is point inside the physical domain. It must satisfy a condition, namely

$$(\mathbf{x} - \mathbf{c}) \cdot \mathbf{n} = \xi_0 \geq 0, \quad \forall \mathbf{x} \in \Gamma \tag{8.14}$$

where ξ_0 is a non-negative constant. The value ξ_0 determined by $\mathbf{c}$ must be passed to to `INTERFACE_START0`. If the physical domain is convex then $\mathbf{c}$ may be any point in the physical domain including Γ. For any physical domain, if $\mathbf{c}$ is chosen on Γ then $\xi_0 = 0$.

The parameter `LAYER_THICKNESS0` is the depth of the layer, ℓ_0. It is recommended that the layer be meshed by sweeping from the surface Γ 15–20 elements deep. This formulation is of course agnostic to element type although we have found improved results using wedges aligned with $\mathbf{n}$ to truncate unstructured tetrahedral meshes. Call the argument passed to `SCALE_FACTOR` α. By default this parameter is assumed to be $\alpha = 0.01$. The minimum relaxation time of the PML is given by

$$\tau = \alpha \frac{\ell}{c}, \qquad c = \frac{c_0}{\sqrt{\epsilon_r \mu_r}}. \tag{8.15}$$

While UPML are impedence matched in the continuum, discretization errors can induce spurious reflections at the boundary of the PML and the physical domain. The standard methodology to overcome this problem is to smoothly turn on the PML. We employ polynomial grading to achieve this effect. Let p be the argument passed to `POLYNOMIAL_GRADING`. By default this parameter is assumed to be $p = 4$. The PML is functionally graded such that for some point $\mathbf{x}$ in the PML we have

$$\frac{1}{\tilde{\tau}(\xi)} = \frac{1}{\tau} \left(\frac{\xi - \xi_0}{\ell} \right)^p, \qquad \xi = (\mathbf{x} - \mathbf{c}) \cdot \mathbf{n}. \tag{8.16}$$

Default parameters were selected so that a plane wave travelling incident to Γ would attenuate by -80 dB in 2ℓ. For $p = 0$ we would have the spatial decay after 2ℓ meters

$$20 \log_{10}(\exp\left(-\frac{2\ell}{c\tau}\right)) = 20 \log_{10}\left(\exp\left(-\frac{2}{\alpha}\right)\right) = -80 \text{ dB} \tag{8.17}$$

This gives $\alpha \approx 0.2$. Dividing by by the mean of x^4 on $(0, 1)$ gives 0.05. We then rounded the value of α to the next smallest order of magnitude.

Note that while it may be tempting to reduce α to very low values (i.e 10^{-12}) this will increase the stiffness of the PML. Thus some balancing between reducing τ and increasing layer thickness is required.

8.3.9 Edge PML

This material model is only supported by the Crank-Nicolson time integrator.

```
EDGE PML
  EPS real (default = 1.0)
  MU real  (default = 1.0)
  SIGMA real (default = 0.0)
```

```
  NORMAL0_X real
  NORMAL0_Y real
  NORMAL0_Z real
  INTERFACE_START0 real
  LAYER_THICKNESS0 real
  NORMAL1_X real
  NORMAL1_Y real
  NORMAL1_Z real
  INTERFACE_START1 real
  LAYER_THICKNESS1 real
  CENTER_X real (default = 0.0)
  CENTER_Y real (default = 0.0)
  CENTER_Z real (default = 0.0)
  POLYNOMIAL_GRADING real (default = 4.0)
  SCALE_FACTOR real (default = 0.01)
  NDOF int
  NTIMESTATES 3
END
```

Edge PMLs are used when two face PMLS are applied to truncate a corner. For example, if physical domain is cube and two open boundaries meet at an edge. Call their physical boundaries Γ_0 and Γ_1.

If $\mathbf{n}_i$ is the outward normal of Γ_i, the center vector $\mathbf{c}$ and interface start locations ξ_i must satisfy

$$\mathbf{n}_i \cdot (\mathbf{x} - \mathbf{c}) = \xi_i \geq 0, \forall \mathbf{x} \in \Gamma_0 \cap \Gamma_1 \tag{8.18}$$

If $\mathbf{c}$ is chosen as a point on $\Gamma_0 \cap \Gamma_1$ then $\xi_i = 0$.

Scale factors, polynomials gradings, relative permittivities and permeabilities, and electrical conductivities should agree between the two Face PMLs and the edge PML. If both Face PMLs are meshed with swept wedges then an edge PML can be a structured hexahedral mesh.

8.3.10 Node PML

This material model is only supported by the Crank-Nicolson integrator.

```
NODE PML
  EPS real (default = 1.0)
  MU real  (default = 1.0)
  SIGMA real (default = 0.0)
  NORMAL0_X real
```

```
  NORMAL0_Y real
  NORMAL0_Z real
  INTERFACE_START0 real
  LAYER_THICKNESS0 real
  NORMAL1_X real
  NORMAL1_Y real
  NORMAL1_Z real
  INTERFACE_START1 real
  LAYER_THICKNESS1 real
  NORMAL2_X real
  NORMAL2_Y real
  NORMAL2_Z real
  INTERFACE_START2 real
  LAYER_THICKNESS2 real
  CENTER_X real (default = 0.0)
  CENTER_Y real (default = 0.0)
  CENTER_Z real (default = 0.0)
  POLYNOMIAL_GRADING real (default = 4.0)
  SCALE_FACTOR real (default = 0.01)
  NDOF int
  NTIMESTATES 3
END
```

Node PMLs are used when three face PMLS are applied to truncate a corner. A node PML will also have three neighboring edge PMLs. For example, if the physical domain is a cube and three open boundaries meet at node or vertex. Call their physical boundaries Γ_0, Γ_1, and Γ_2.

If $\mathbf{n}_i$ is the outward normal of Γ_i, the center vector $\mathbf{c}$ and interface start locations ξ_0, ξ_1, ξ_2 must satisfy

$$\mathbf{n}_i \cdot (\mathbf{x} - \mathbf{c}) = \xi_i \geq 0, \forall \mathbf{x} \in \Gamma_0 \cap \Gamma_1 \cap \Gamma_2 \tag{8.19}$$

If $\mathbf{c}$ is chosen as a point as the point in $\Gamma_0 \cap \Gamma_1 \cap \Gamma_2$ then $\xi_i = 0$.

Scale factors, polynomials gradings, relative permittivities and permeabilities, and electrical conductivities should agree between all adjacent PMLs. If the Face PML are meshed with swept wedges then the node PML may be meshed with structured hexahedra.

9 Hybrid FETD/FDTD

Presently only STDEM, which is classical Finite-Difference Time-Domain (FDTD), contains both the Perfectly Matched Layer (PML) boundary condition and the Near-To-Far (NTF) field transformation. Therefore it remains useful for EM scattering and RCS simulations. Additional useful keywords for STDEM can be found in [1].

A key requirement with HTDEM is that since it involves FDTD, it is conditionally stable and the Courant stability condition must be met. If a very rapid instability is seen this is the first place to look. For cubical cells with Δ the globally smallest spatial dimension, the condition is

$$\Delta t < \frac{\Delta}{c\sqrt{3}}.$$

In general

$$\Delta t < \frac{1}{c\sqrt{\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2} + \frac{1}{\Delta z^2}}}.$$

The user can specify the time step to be used using the `CONSTANT TIME STEP` keyword. If no `CONSTANT TIME STEP` keyword is present, the code will compute an appropriate Courant time step using the first equation above with a safety factor of 0.9 assuming non-spatially varying cells which is typical for hybrid simulations.

9.1 Keywords

The physics keyword `HYBRID TD ELECTROMAGNETICS` specifies that hybrid time-domain electromagnetics (HTDEM) physics is to be used to couple structured and unstructured meshes. This goes in place of the `UNSTRUCTURED TD ELECTROMAGNETICS` physics keyword (see Section 2).

`HEX MASS LUMP, bool | string`

Specifies whether mass lumped integration is activated for hex elements.

Mass-lumped integration is required for successful hybrid simulations. The possible options are true (or yes) and false (or no). The default is presently false.

`WRAPPER, SIDESET int`

Specifies the sideset identifying the boundary of the unstructured mesh.

The `WRAPPER` keyword is only processed if the edgesets identifying the fields to exchange between the structured and unstructured meshes are not detected in the mesh description. For I-DEAS generated meshes the preprocessor can automatically generate these edgesets and the elements required to interface the unstructured mesh to the structured mesh for models without material variations in the interface region. However, in cases where material variations are present in this interface region, the user needs to manually generate the interface elements and use the wrapper keyword to identify the boundary of the unstructured mesh. The specified sideset and the first element block found containing hexahedral elements are used to generate the edgesets needed to connect the meshes.

`FIELD SOLVER, PML, PROFILE=X, FUNCTION= int, Y, FUNCTION= int, Z, FUNCTION= int, BLOCK=1,2,3,4,5,6`

Specifies the use of the STDEM PML solver.

The `FIELD SOLVER` keyword requires HTDEM physics. The required sub-keyword `PML, PROFILE` provides the functional description of the PML in each of the three coordinate dimensions. Often these are the same function so a single function number can be referenced three times. The required sub-keyword `BLOCKS` specifies the user block numbers of the PML blocks. These are always 1-6 as shown if using the provided structured mesh/PML generation template discussed in section 9.2.

`FAR FIELD, SEGMENT = real real real real real real [, PHASECENTER = real real real, LOOKANGLES real real real real ...]`

Specifies that far-field signals are to be computed.

The `FAR FIELD` keyword requires HTDEM physics. The required sub-keyword `SEGMENT` provides the coordinates of two points on the structured grid describing the virtual surface which surrounds the scatterer which resides in the unstructured mesh. This `SEGMENT` should reside just outside the hybrid interface region and inside the PML absorbing boundary condition which will typically terminate the structured grid for scattering simulations. The optional `PHASECENTER` keyword provides the 3D Cartesian coordinates (x, y, z) of the desired phase reference for the far fields. If not provided, the Cartesian coordinate origin is used as the phase reference. The other optional keyword `LOOKANGLES` provides (θ, ϕ), pairs of look angles (in degrees) where the transient far-field waveforms Etheta and Ephi are desired.

If desired, the frequency content of the excitation waveform can be deconvolved from the transient field waveforms as a post-processing step. A PFIDL script is available for accomplishing this deconvolution.

The transient fields are written to the ascii file *problem_name*.EtEp.asc. They are also available in the *problem_name*.exo file but only the time cycles corresponding to the `EMIT PLOT` keyword will be available there. The excitation waveform can be obtained from the *problem_name*.his file. All time cycles will be available if "`EMIT HISPLT, CYCLE INTERVAL = 1`" is used.

`FAR FIELD PATTERN, PHISTART =  real, [PHIEND = real, NUMPHI = int,] THETASTART = real, [THETAEND = real, NUMTHETA = int,] FREQUENCIES real real real ...`

Specifies that far-field patterns are to be computed.

The `FAR FIELD PATTERN` requires the `FAR FIELD` keyword specify at a minimum the location of the virtual-surface `SEGMENT`. The start and end keywords of each angle specify the starting and ending angles (in degrees) of each for the particular pattern.

If only a single phi cut is desired, only `PHISTART` need be specified and similarly only `THETASTART` for a single theta cut. At least one frequency must be provided at which to compute the patterns.

Unlike the transient far-field results, the frequency content of the excitation waveform time history has been removed from the resulting patterns. An inherent assumption is that a single source is used. Possible sources include those described by the `SOURCE` keyword, the `PORT SOURCE` keyword, and the `PLANE WAVE SOURCE` keyword.

The patterns are written to the ascii file *problem_ name*.Pat.asc. A script is available to read these patterns using PFIDL.

Since the patterns involve Fourier transforms of transient results, the user must verify that the simulation is run long enough that all relevant far-field transients have damped to zero. This is to avoid problems with aliasing which would render the pattern results invalid. Restart can be used to extend the simulation in the case that steady state has not yet been reached.

9.2 Mesh Generation

Creation of a suitable unstructured mesh boundary to couple to the structured mesh requires a few specialized steps.

Two specialized templates (see Appendix F) are required to generate the unstructured mesh using Cubit and the structured mesh using the legacy preprocessor for Quicksilver, Mercury. The first is a Cubit journal file creating for this example a simple spherical scatterer and containing the appropriate commands to create the wrapper for connecting to the structured mesh. The second is a Mercury input file to generate a structured mesh to couple with the unstructured mesh.

Significant changes may be required in the journal template depending on the user's desired scattering model. This example generates the mesh shown in Fig. 4.

The dark blue surface represents the (very coarse in this case) scattering sphere and the magenta surface the total/scattered field boundary. This boundary is where a plane-wave source would launch from. The volume between the blue and magenta surface is filled with tetrahedrons making up the total-field region. The volume between the magenta and the green surface contains tetrahedrons in the scattered-field region. The orange is the transition between tetrahedrons and hexahedrons which includes some pyramids to mate with the outer hexahedrons in light blue.

In addition to a plane-wave source, the total-field region may contain other source types such as the volume current source (J Source). A useful exercise is to modify the mesh such that the scatterer is removed and replaced with it's internal mesh. A plane-wave source can now be launched and observed to propagate across the total-field region. The fields in the scattered-field region will be zero for all time since there is no scatterer.

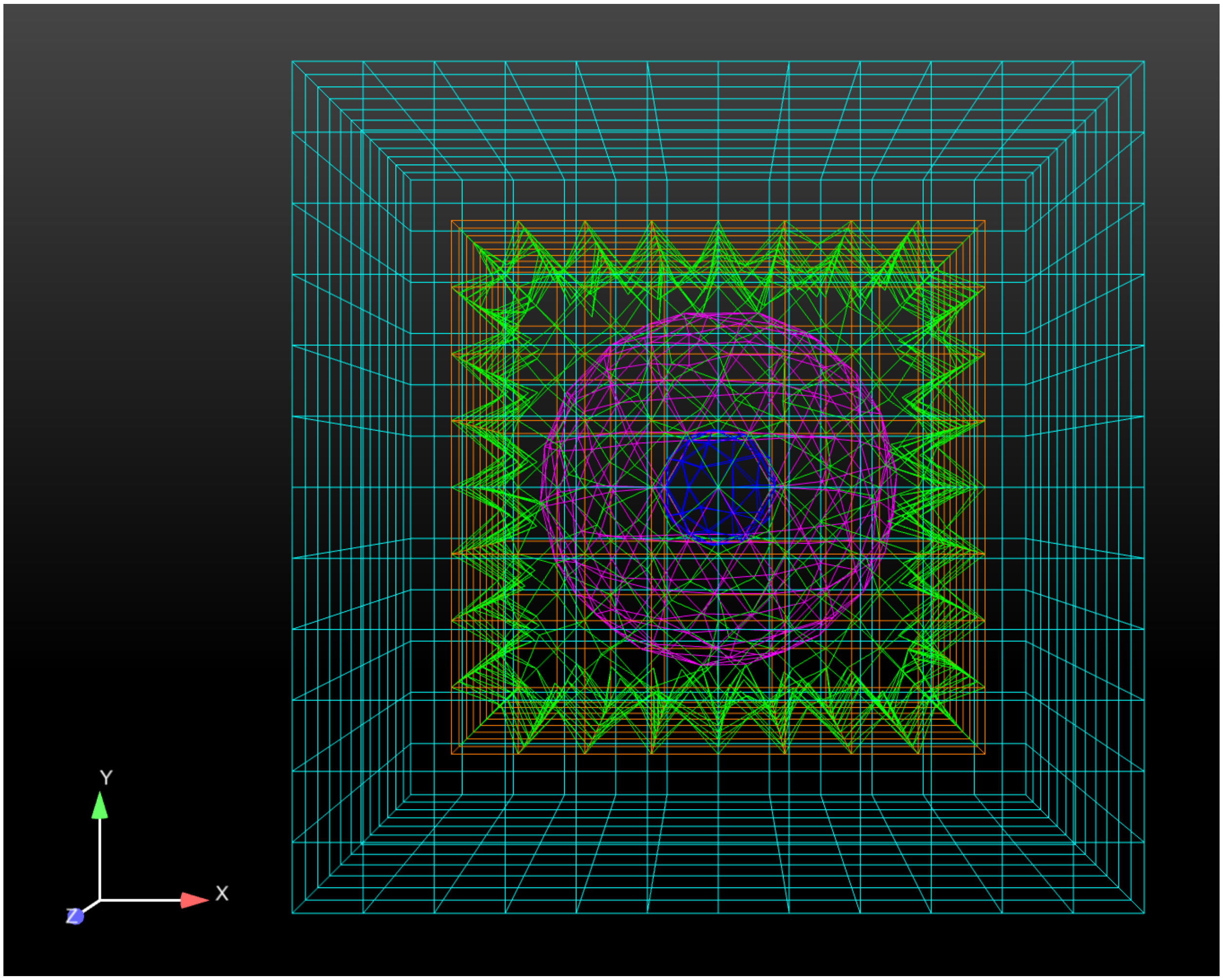

Figure 4. Example hybrid mesh.

The FDTD mesh generated by the Mercury template overlaps with the blue hexahedrons in Fig. 4 to form the hybrid communication. A few changes are required in this template including 1) the mesh deltas in each direction (template lines 10-12), 2) the overall mesh dimensions (template lines 14-16), and the depth of the PML boundary (line 27). The minimum overall mesh dimensions should extend two FDTD cells outside the journal file mesh in each dimension, ie, journalSize+4 each dimension. The PML boundaries are added outside of this dimension. If more space is desired between the hybrid interface and the start of the PML, the overall mesh dimensions can be made larger than the minimum values.

Presently the Mercury preprocessor is available on the PPIC LAN and tlcc2 cluster (chama, uno, etc.) linux machines. On the PPIC LAN, define:

```
export QSROOT=/remote/tdpoint/qsroot
export QS_SYS_TYPE=linux_x86_64_intel
```

On the tlcc2 cluster machines, define:

```
export QSROOT=/projects/emphasis/qsroot
export QS_SYS_TYPE=tlcc2
```

Then on any machine, define:

```
export MERKmaster=$QSROOT/qs/master.merk
alias merk="$QSROOT/bin/$QS_SYS_TYPE/merk_ng.exe -e20 -q -E"
```

At this point the file is processed as follows, where the number of parallel processors desired must be specified. Whatever that number is, the file will also function in serial.

```
merk -Dnproc=int hybrid.mrk outfile [> merk.out]
```

In this example "hybrid.mrk" is the name chosen for the Mercury input file. This command directs Mercury to write two files for EMPHASIS, "*outfile*.pff" and "*outfile*.inp". The *pff* file contains the structured mesh and is ready to go for serial and the specified number of parallel processors. The *inp* file is a skeletal version which contains primarily the PML description which the user must transfer into his own EMPHASIS input file, or use this one as a starting point. Mercury outputs information about the Courant limit and block decomposition across processors to *stdout*. Redirecting this to a file provides a useful archive for the block decomposition.

The *outfile*.inp file created by the example hybrid.mrk file is also shown in Appendix F. The PML functions of interest are FUNCTIONS 3, 4, and 5. In most cases the functions are identical since normally the PML thickness is the same in all three dimensions. Therefore, only one function need be copied and placed into the EMPHASIS input for the scattering problem. This function is then referenced in the `FIELD SOLVER` keyword.

A full working EMPHASIS input file using the PML description from the *outfile*.inp file is shown at the end of Appendix F.

If errors exist in the hybrid connection, EMPHASIS will generate errors such as

```
*** ERROR: STDEM_Point_History::Initialize
Unable to locate point

P0: *** ERROR:   Interface_Direction(): Illegal PML block configuration
 detected PML GID= 9 non-PML GID=4
```

In this case the user should verify that the mesh deltas and dimensions are correct in the Mercury template.

10 Inlet-port Poisson Solutions

The simulation of inlet ports where the port field distribution is derived from the static field distribution over the port is accomplished using the following process:

1. Create the full 3D mesh for the simulation, including an inlet port. The inlet must be *planar*, but can reside at an arbitrary spatial location. If meshing with I-DEAS, export this mesh to a genesis database using PREP [5]. PREP will initialize number-of-nodes-per-side distribution factors per side to zero. Other mesh generators producing generic must do the same. A suggested name for this mesh file is *problem_name*`.gen`.

2. Generate a 2D mesh description of the inlet for the Poisson solver by invoking `UTDEM SIDESET EXTRACTOR` physics in Emphasis (3D). The input file for this step contains the sideset id of the inlet port to be extracted as well as the ids of any sidesets intersecting the extracted sideset. These intersecting sidesets are those required to set boundary conditions for the 2D Poisson solve. The sideset extractor will extract the inlet sideset, convert it to a 2D mesh description, and write a 2D gen file for Emphasis. The extractor will also record in this gen file the appropriate transformation matrix to be applied to the Poisson results to properly position the Poisson solution into the 3D gen file. An example input file is shown in Appendix C. The extractor will create the 2D mesh file using the name *problem_name*`.2D.gen`.

3. Solve the Poisson problem by invoking `CABANA POISSON` physics with the `POISSON SOLUTION` keyword in Emphasis (2D). After the solution is obtained, Emphasis will write the solution and the transformation matrix into the 3D gen file for UTDEM. For this step, a new *copy* of the full 3D gen file created in step 1 must be available in the directory so that the Poisson results can be written into it. If the 3D gen file has already been modified in this manner, the Poisson solution will note this and ask the user to provide a new copy. Since this is a generated file, a suggested name for this copy is *problem_name*`.inlet.gen`.

   An example input file is shown in Appendix D. Here, the `POISSON SOLUTION` keyword specifies that Cabana obtain a single Poisson solution on the mesh with the given boundary conditions and exit. In addition, a constant `CHARGE DENSITY` can be supplied for the mesh volume. An ascii file will also be written containing the results with filename *results_file*. Dirichlet boundary conditions are supplied with the `CONDUCTOR` keyword. The `EXPORT RESULTS` keyword controls the writing of results to the UTDEM 3D genesis file and `SIDESET` specified. Further information can be found in [13].

4. Invoke `UTDEM` physics in Emphasis (3D) to complete the full 3D solution, specifying `FIELD DIST` in the port source keyword descriptor.

11 Running UTDEM PIC With ITS Source Data

UTDEM PIC can be used to simulate EMP (electromagnetic pulse) effects resulting from X-ray or gamma ray sources. The ITS (Integrated Tiger Series) codes [6] are used to simulate coupled electron/photon transport in complex geometries with multiple materials. However, for electron transport in vacuum or gas, ITS cannot self-consistently model the non-linear electromagnetic response when the space-charge and current of the electrons has a significant effect on the EM fields. On the other hand, UTDEM PIC is specifically designed to self-consistently handle electron motion and EM fields.

For many EMP problems, a reasonable approximation is a "weak coupling" approach in which ITS does the radiation transport and computes electron source data for use in an Emphasis simulation for the self-consistent EM response. This weak coupling is usually applicable when the ITS simulation does not have high-current electron flow in the vacuum and/or gas-filled regions where the EMP response is needed. ITS and Emphasis currently support three types of coupling:

1. Electron emission from surfaces: emitting Emphasis sidesets are matched to corresponding ITS emission subsurfaces.
2. Electron emission from volumes: emitting Emphasis element blocks are matched to corresponding ITS emission subzones.
3. ITS energy deposition in subzones is used as a direct source for updating the electrical conductivity in elements using the HP Gas Electrical model.

The data files built by ITS for Emphasis use the binary "portable file format" (PFF). PFF was developed in the late 1980's to provide a compact format for transferring data between very different platforms (Cray and VAX) at a time when disk space was very limited. It has been continuously improved over time, and supports a wide array of applications. PFF is now open-source software, part of the "Hermes Utilities" package [10]. A PFF file is a collection of randomly accessible datasets. A dataset is an aggregation of all the data needed to define a high-level object, *e.g.* scalar or field data on a grid, or a list of particle ordinates and attributes (momenta, charge, *etc.*) The two dataset types used here are NGD (m-dimensional vectors on an n-dimensional non-uniform grid), and IFL (a generic dataset with one integer and one float array). All real data in PFF is single-precision, but PFF provides utilities to encode a double into a short int array, and decode the int array back.

11.1 ITS Electron Emission Tallies

An ITS electron emission tally is a discrete ("histogram") distribution on a 3-D energy-angle grid: energy E in MeV, polar angle θ, and azimuthal angle ϕ. The two angles are application-specific, and will be described later. The tally is a set of counts on a 3-D array of sampling bins with:

N_E energy bins: $\{E_i \leq E < E_{i+1}; 1 \leq i \leq N_E\}$, of size $\Delta E_i = E_{i+1} - E_i$,

N_θ θ bins: $\{\theta_j \leq \theta < \theta_{j+1}; 1 \leq j \leq N_\theta\}$, of size $\Delta\mu_j = cos\theta_j - cos\theta_{j+1}$, and

N_ϕ ϕ bins: $\{\phi_k \leq \phi < \phi_{k+1}; 1 \leq k \leq N_\phi\}$, of size $\Delta\phi_k = \phi_{k+1} - \phi_k$.

The tally data is the *density* of counts in each bin:

$$T_{ijk} = \frac{C_{ijk}}{\Delta E_i \Delta\theta_j \Delta\phi_k}.$$

ITS simulations track a large number of source particles, and normalizes each count to a single source particle. The sum of the counts over all bins is an important quantity for normalizing Emphasis emission amplitudes, and is denoted C_0.

Historically, a single emission tally has been written to a single NGD dataset. To store all the bin-boundary values, the size of the 3-D NGD dataset is $(N_E+1) \times (N_\theta+1) \times (N_\phi+1)$. The upper planes and edges of the data array in each direction are filled with 2-D and 1-D distributions respectively (integrating over one or two directions), while C_0 is put in the uppermost corner, $(N_E+1, N_\theta+1, N_\phi+1)$. An ITS simulation uses the same energy-angle grid for all emission tallies of the same type (surface or volume). Furthermore, Emphasis only needs the $N_E N_\theta N_\phi$ 3D tally data, and the reduced distributions are easy to compute if needed. For a small number of tallies, the wasted file space is not much of a concern. However, time-dependent ITS simulations with many emission subsurfaces and/or subzones can now generate huge number of tallies, $>10^5$. Not only is file space wasted, but it cumbersome to have so many datasets in a PFF file.

To handle large numbers of tallies efficiently, a new "multi-tally" format has been developed, using the float array of two IFL datasets. For a set of N_T tallies, the energy-angle grid data is written only once to one dataset, and the tally data is written to the float array of a second IFL dataset. The first N_T data values are C_0 for each tally, followed by $N_T N_E N_\theta N_\phi$ values for the 3D data. These datasets are currently implemented only for volume emission.

11.2 Surface Emission

ITS combinatorial geometry builds a domain using "bodies" of primitive shapes: box, sphere, cylinder, *etc.* For surface emission, ITS also defines a numbering scheme for the surfaces on each body type. For example, there are three surfaces on a cylinder: #1 and #2 are the disk endfaces, and #3 is the $r = const$ surface between the two disks. A local 2-D coordinate system (U, V) is defined for each surface on each body type. Each surface can then be divided up into $N_U \times N_V$ subsurfaces. An ITS simulation can request surface emission tallies from one or more surfaces from one or more bodies, each with its own (N_U, N_V) subsurfacing parameters. An emission tally is generated for each subsurface. For a time-dependent simulation, there is a tally for each subsurface and time bin.

In a surface emission tally, The angles θ and ϕ are computed from the momentum components of each electron in a local right-handed Cartesian coordinate system, $(\hat{n}, \hat{\phi}_0, \hat{\phi}_{90})$, where $\hat{n}$ is the normal to the surface, $\hat{\phi}_0$ defines the $\phi = 0$ axis, and $\hat{\phi}_{90} = \hat{n} \times \hat{\phi}_0$ defines the $\phi = 90^o$ axis. ITS systematically defines $\hat{\phi}_0$ for each surface type. For curved surfaces, the local basis varies with position, but the global variation over a single subsurface decreases with increasing number of subdivisions. The range of θ is 0^o (normal to the surface) to 90^o (tangential).

In UTDEM PIC, a BEAM EMISSION sideset is mapped to the tallies of one or more ITS subsurfaces. Three problems must be addressed:

1. Spatially locating which subsurface(s) to use.
2. Building momentum components in a local basis matching the ITS basis $(\hat{n}, \hat{\phi}_0, \hat{\phi}_{90})$.
3. Correctly normalizing the beam current emission amplitude.

Item #3 will be discussed in section 11.4. Surface emission has undergone several iterations since its inception in *c.* 2006, embedding more geometry information in the PFF file to automate the simulation setup. UTDEM supports the following versions of a surface emission PFF file:

1. The "original-version 0" files require a 1-1 mapping between a sideset and a single ITS subsurface tally in a PFF dataset, and have no embedded geometry information. The file is just a collection of NGD datasets with tally data. The user is responsible for explicitly providing a dataset number in the `BEAM EMISSION` command using the keyword/value pair "`ITS` *ds_num*". Furthermore, for most curved surfaces, $\hat{\phi}_0$ needs to be explicitly defined with the `PHI0` keyword. Newer versions of ITS no longer generate these files, but they are still supported.
2. "Version 0" files still require a 1-1 mapping between a sideset and a single ITS subsurface, but include some embedded geometry information. As above, the file is just a collection of NGD tally datasets. However, each dataset also includes (1) the smallest bounding box in the global ITS coordinate system that enclosed the subsurface, and (2) the $\hat{\phi}_0$ vector, if needed. With these files, the `PHI0` keyword is no longer required, and (in most cases) the dataset number for the `ITS` keyword. The required dataset is the one with the smallest bounding box enclosing the sideset. For complicated geometry, this auto-fitting algorithm can still fail; in this case, the user must still explicitly provide the dataset number.
3. "Version 1" files adds structure and more detailed geometry information by organizing the file into sets of tallies for each ITS emission surface, in the order requested in the ITS input file's `ELECTRON-EMISSION` command. For each surface, there is a new

"surface header" IFL dataset, describing the body geometry, the surface on the body, and subsurfacing parameters. This is followed by the $N_U \times N_V$ NGD datasets for the tallies on each subsurface. Emphasis loads the data in the file, building ITS_Body and ITS_Surface objects. A sideset is no longer restricted to having a 1-1 mapping with a subsurface. Instead, Emphasis can fit each face to the appropriate subsurface. Furthermore the sideset can overlap multiple ITS surfaces.

Using *version-1* files removes the restriction that the Emphasis mesh be conformal to ITS subsurface boundaries—a major improvement. However, there are limitations on accuracy that should be noted. First, consider the face fitting algorithm. A face is on a surface if all three nodes are on it (to with a given tolerance). The subsurface assigned to the face is determined by the face centroid. With a non-conforming mesh, a face can overlap other subsurfaces. Second, the actual ***emitting*** area of an ITS subsurface may be smaller than its geometric area because it is obscured by another ITS body. For this reason, Emphasis computes the total emission area of the ITS subsurface, A_{SS}, as the sum of the area of the fitted faces. It then assigns a fraction of the total ITS emission count to each face, $C_{face} = C_0 A_{face}/A_{SS}$. For a fully emitting surface with many subzones and a coarse Emphasis mesh, there may be large fluctuations in each A_{SS}, and thus the local face emission density. Thus, there is still some incentive to coerce the Emphasis mesh to conform to subsurface boundaries, but it does not require creating sidesets exactly matching subsurfaces. Finally, note that a sideset must completely cover the emitting area of a subsurface, otherwise the emission density will be artificially enhanced.

11.3 Volume Emission and Energy Deposition

In ITS combinatorial geometry, boolean operations on the bodies previously discussed for surface emission are used to create "zones". Zones can be subdivided into "subzones", analagous to the (U,V) subsurfacing of emission surfaces. ITS has conventions for automatically subzoning a wide variety of single and two body zones with three subzoning parameters.

ITS has the option to save volumetric electron emission tallies in subzones. These are tallies of electrons created in space, either with a photon or primary electron source. All such tallies use the global ITS Cartesian coordinate system for defining the angles θ and ϕ. Here θ is in the range $[0, 180]$ degrees, with $\theta = 0^o$ along the +z axis and $\theta = 180^o$ along the -z axis.

A "version-1" volumetric emission PFF file is structured in the same way as *version-1* surface files. For each emitting zone, there is a "zone header" IFL dataset describing the zone geometry and the three subzoning parameters, followed by N_{SZ} NGD datasets with emission tallies for each subzone, in ITS order. A "version-2" volumetric emission file is structured similarly, but with the N_{SZ} NGD tally datasets replaced by one or more *multi-tally* IFL datasets with the tally data (ITS has the option to split output for very large numbers of tallies into several IFL datasets to limit the dataset size). A single copy of the common energy-angle grid is written to the float array of the *zone header* dataset.

In Emphasis, elements in a specified block are auto-fitted to a subzone by requiring that the centroid be within the subzone to a specified tolerance. There are considerations for accuracy similar to surface emission. The volume of each subzone, V_{SZ}, is the sum of the volume of all the elements fitted to it. The ITS emission count assigned to each element is $C_{el} = C_0 V_{el}/V_{SZ}$. To reduce fluctuations in the local emission density, it is worth the effort to build an Emphasis mesh that conforms to subzone boundaries as much as possible.

With volume emission, it is easy to generate very large numbers of emission tallies, and other approximations are needed to make some simulations feasible. An important case is doing large-scale EMP simulations in air with pressure in the range $\sim 1-10 < P < 760 Torr$. Here, the HP Gas Electrical model is applicable for the air breakdown plasma. However, it is only necessary to create kinetic electrons if their collisional range is larger than an element size. If not, the electron will lose all its energy in the element it was created in. We therefore approximate the system with two sources from ITS. Volume emission is used for creating kinetic electrons above some cutoff energy E_{cut}, and energy deposition ITS electrons created below E_{cut} is deposited locally in each element. This deposited energy is used as an extra source term for creating electron-ion pairs in the air breakdown plasma.

ITS energy deposition is stored in the PFF file using IFL datasets, one per zone. The integer array contains the same data to define the zone geometry and subzoning parameters as the *zone header* for volume emission. The float array contains the N_{SZ} values (in MeV). Emphasis automatically loads both types of data, and creates ITS_Zone objects that have either energy deposition or volume emission, or both.

11.4 Setting Up Time-Dependent Emission

Traditionally, ITS simulations have been time-independent. To drive a UTDEM PIC simulation of an experiment with ITS data from a a time-independent simulation, the following additional data is required:

1. A source pulse waveform, f(t), typically from one or more PCD signals.
2. The total number of source particles, N_{src}, to absolutely normalize the emission amplitude.

The `BEAM EMISSION` command uses the function f(t) for the `TEMPORAL` option and the scale factor for the `AMPLITUDE` keyword is

$$I_0 = \frac{eN_{src}}{F_0} f(t),$$

where,

$$F_0 = \int_0^\infty f(t)\mathrm{d}t.$$

Note that I_0 has units of current. This approach assumes that the transit time between all emission surfaces is short compared to the time scale of the driving pulse. For SGEMP simulations of small cavities, this is a very reasonable assumption.

In fact, ITS can do time-dependent simulations. This capability was initially added to diagnose the output of standard ITS sources with time-dependent spectra. ITS can now also be driven with time-dependent sources built from Quicksilver PIC simulations. Emphasis can read time-dependent ITS PFF files and use them as a source with no need for an external $f(t)$ function.

For a time-dependent ITS simulation, a set of "time bin" boundaries defines tally output, $\{t_n; 1 \leq n \leq N_t + 1\}$, which can be non-uniformly spaced if requested. The first dataset of all output PFF files is an IFL dataset with the time bin boundaries. When Emphasis first opens a source ITS PFF file, it checks to see if the first dataset is a time bin IFL dataset. If it is, it handles the time-dependence automatically, with no additional user input required. However, one important difference with time-independent simulations must be addressed. To correctly handle non-uniform time bin spacing, all ITS tally data is divided by the time bin width, *i.e.* counts are now count rates. The `AMPLITUDE` scale factor for these simulations is thus $Q_0 = eN_{src}$, in units of charge.

References

[1] R. S. Coats, M. F. Pasik, and D. B. Seidel. EMPHASIS^{TM}/Nevada STDEM user's guide version 1.0. Technical Report SAND2005-1024, Sandia National Laboratories, April 2005.

[2] A. C. Robinson et al. ALEGRA-MHD user manual. Technical Report SAND2014-16032, Sandia National Laboratories, 2014.

[3] A. C. Robinson et al. ALEGRA user manual. Technical Report SAND2014-16031, Sandia National Laboratories, 2014.

[4] E. R. Keiter et al. Xyce^{TM} parallel electronic simulator reference guide, version 6.2. Technical Report SAND2014-18057, Sandia National Laboratories, September 2014.

[5] M. F. Pasik et al. Volmax user's guide. Technical report, Sandia National Laboratories, unpublished.

[6] B. C. Franke, R. P. Kensek, T. W. Laub, and M. J. Crawford. ITS Version 6: The integrated TIGER series of coupled electron/photon monte carlo transport codes, revision 4. Technical Report SAND2008-3331, Sandia National Laboratories, July 2009.

[7] A. Friedman. A second-order implicit particle mover with adjustable damping. *Journal of Computational Physics*, 90:292–312, 1990.

[8] A. D. Greenwood, K. L. Cartwright, J. W. Luginsland, and E. A. Baca. On the elimination of numerical Cerenkov radiation in PIC simulations. *Journal of Computational Physics*, 201:665–684, 2004.

[9] L. A. Schoof and V. R. Yarberry. EXODUS II: A Finite Element Data Model. Technical report SAND92-2137, Sandia National Laboratories, Albuquerque, NM, November 1995.

[10] D. B. Seidel, M. F. Pasik, and T. D. Pointon. Hermes utilities. https://www.sourceforge.net/projects/hermes-util.

[11] T. A. Stringer and N. S. Dumcum. Comprehensive report on gas/foam RIC test and analyses. Technical report, ITT Industries, September 2002.

[12] R. S. Tuminaro et al. Official Aztec Users's Guide - Version 2.1. Technical report SAND99-8801J, Sandia National Laboratories, Albuquerque, NM 87185, November 1999.

[13] C. D. Turner, W. J. Bohnhoff, and J. L. Powell. EMPHASIS^{TM}/Nevada CABANA user guide version 2.1.1. Technical Report SAND2014-16736, Sandia National Laboratories, August 2014.

[14] C. D. Turner, T. D. Pointon, and K. L. Cartwright. Emphasis/NEVADA unstructured FEM implementation version 2.1.1. Technical Report SAND2014-16737, Sandia National Laboratories, August 2014.

A Use of the PREP Mesh Preprocessor

The mesh preprocessor PREP is a holdover from legacy code that has been refactored to provide conversion from I-DEAS Universal format to ExodusII format for EMPHASIS/Nevada UTDEM. It is also used to provide encoded sidesets for subsequent conversion to edgesets by Nevada. Use of PREP is not required if 1) the chosen mesh-generation software creates ExodusII format directly including all required nodesets, sideset-coded edgesets, and sidesets, or 2) the simulation requires no edgesets (or only virtual edgesets) but otherwise the mesh-generation software can provide the required ExodusII description including nodesets and sidesets.

Reference [5] provides general guidance on the use of PREP but a few additional comments are warranted. A typical PREP input file for a pure unstructured UTDEM simulation is shown below.

```
\$INPUT
flagwrap   =  'N'
flagblocks =  'N'
flagchaco  =  'Y'
flagalegra =  'Y'
nodeblu    =  'HLinInt1'
nodegryblu =  'ELinInt1 HLinInt1'
nodeltblu  =  'ELinInt2 HLinInt1'
nodemag    =  'ELinInt1'
nodepnk    =  'ELinInt2'
nodecyn    =  'Load1'
nodegrn    =  'Load2'
nodeorg    =  'Source1'
nodeltmag  =  'Obs'
\$END
```

The “flagwrap”, “flagblocks”, “flagchaco”, and “flagalegra” keywords must be set as shown for pure unstructured. If this were a hybrid FDTD/FETD simulation, “flagwrap” and “flagblocks” would be set to ‘Y’.

The “node*” node color attribute keywords are required to assist PREP in making connections between our I-DEAS meshing convention of using node color to represent attributes requiring nodesets or edgesets, including non-port sources, observers including slot and wire observers, loads, slots, and wires. Attributes requiring mesh-related edgesets, such as ElinInt (E Line Integral) and wire, also require beam elements along the path to further assist PREP in correctly defining the edgeset. Attributes utilizing sidesets and virtual edgesets do not appear in the PREP input file. Sidesets are generated within I-DEAS by utilizing “pressure” or “force” boundary conditions.

Node color keywords containing more than one attribute, such as "nodegryblu" above, indicate that those paths intersect in the mesh. In this case, an E Line Integral path intersects an H Line Integral path. Therefore, both attributes must be assigned to that node color so PREP can correctly include that node in both paths.

B Complete UTDEM input file

```
$------------------------------------BEGIN_QA---------------------------
$ ID:         ucavabc_slots
$ Title:      Cavity w/slots surrounded by ABC
$ Category:   Regression
$ Physics:    electromagnetics
$ Dimension:  3D
$ Owner:      C. David Turner
$
$ Description:
$
$   Conducting cavity with internal source and slots in walls.
$   Energy leaks out through slots to observer outside cavity.
$   ABC surrounds entire object.
$
$-------------------------------------END_QA----------------------------

TITLE
Unstructured 3D cavity with edge source and observer

$
$ The following line will have nevada print out the sequence of calls
$ it makes during execution.
$
$DEBUG MODE, LOCATION

$ The following line dumps out node/edge/face/element connectivity info.
$DUMP FACES

UNSTRUCTURED TD ELECTROMAGNETICS

  formulation, second order
  aztec set, 0

  abc bc, sideset 4
  pec bc, sideset 2
```

```
  observer, nodeset 28
  observer, nodeset 29

  source, nodeset 31, function 1

  slot observer, nodeset 19
  slot, edgeset 123, aztec_set 1, width 0.00001, depth 0.0, int_mat 1, ext_mat 2

  slot observer, nodeset 20
  slot, edgeset 124, aztec_set 2, width 0.00005, depth 0.0, int_mat 1, ext_mat 2

  CONSTANT TIME STEP 1.01197539e-09

  GRADUAL STARTUP FACTOR 1.0

  BLOCK 1
    MATERIAL 1
  END

  BLOCK 22
    MATERIAL 2
  END

END

FUNCTION 1 GAUSSIAN, SCALE=1.0 WIDTH=2.0e-9 SHIFT=10.0e-9

EXODUS EDGE SETS (123 124 153)

$$$$$$$$$$$$$$$$$$$$ execution control $$$$$$$$$$$$$$$$$$$$

TERMINATION TIME 9.0e-9

$$$$$$$$$$$$$$$$$$$$ solver control    $$$$$$$$$$$$$$$$$$$$

aztec
  solver,    cg
  precond,   jacobi
  output,    none
  tol        = 1.e-12
  polynomial order,  1
end

aztec 1
  solver,    cg
```

```
  precond,   jacobi
  output,    none
  tol        = 1.e-12
  polynomial order,  1
end

units, si

$$$$$$$$$$$$$$$$$$$$ output control    $$$$$$$$$$$$$$$$$$$$

EMIT SCREEN, CYCLE INTERVAL = 1
EMIT PLOT, CYCLE INTERVAL = 2

PLOT VARIABLE
  ELECTRIC_FIELD
END

$$$$$$$$$$$$$$$$$$$$ material models $$$$$$$$$$$$$$$$$$$$

MATERIAL 1
  model 1
END

MATERIAL 2
  model 2
END

MODEL 1 SIMPLE ELECTRICAL
  EPS 1.
  MU 1.
  SIGMA 0.
END

MODEL 2 SIMPLE ELECTRICAL
  EPS 1.
  MU 1.
  SIGMA 0.
END

EXIT
```

C Sideset Extractor input file

```
$------------------------------------BEGIN_QA---------------------------
$
$ Sideset extract example
$
$-------------------------------------END_QA----------------------------

$debug mode, LOCATION

title
  SIDESET extract: extract a side set and turn it into a 2D mesh file

$$$$$$$$$$$$$$$$$$$$ physics options $$$$$$$$$$$$$$$$$$$$

UTDEM Sideset Extractor
  extract, sideset 10
  intersect, sideset 9, sideset 14, sideset 15, sideset 16, end

  block 1
    material 1
  end

end

$$$$$$$$$$$$$$$$$$$$ execution control $$$$$$$$$$$$$$$$$$$$

termination cycle = 1

emit plot, cycle interval = 1

$$$$$$$$$$$$$$$$$$$$ material models $$$$$$$$$$$$$$$$$$$$

material 1
end

exit
```

D Poisson Solution input file

```
$------------------------------------BEGIN_QA---------------------------
$
```

```
$ Poisson solution example
$
$------------------------------------END_QA----------------------------

$debug mode, LOCATION
$debug mode, CABANA

title
  CABANA: Poisson solution

$$$$$$$$$$$$$$$$$$$$ physics options $$$$$$$$$$$$$$$$$$$$

CABANA POISSON

  Poisson solution, charge density 0., results file "inlet.out"

  conductor, sideset 9, potential 0.
  conductor, sideset 14, potential 0.
  conductor, sideset 15, potential 1.
  conductor, sideset 16, potential 1.

  export results, genesis file "z_vert.inlet.gen", sideset 10

  block 10
    material 1
  end

end

double precision exodus

aztec
  solver,   cg
  precond,  none
  scaling,  sym_diag
  output,   none
  tol      = 1.e-6
  polynomial order, 1
end

units, si

$$$$$$$$$$$$$$$$$$$$ execution control $$$$$$$$$$$$$$$$$$$$

$ Nevada requires termination time to be specified:
```

```
termination time = 1.e-8

$$$$$$$$$$$$$$$$$$$$ material models $$$$$$$$$$$$$$$$$$$$

emit screen, cycle interval = 1
emit plot, cycle interval = 1

plot variable
  potential
  electric_field
  charge_density
end

$$$$$$$$$$$$$$$$$$$$ material models $$$$$$$$$$$$$$$$$$$$

Material 1
  Model 1
end

Model 1 Simple Electrical
  eps 1.
  mu 1.
  sigma 0.
end

crt: off

exit
```

E Runtime Compiler Functionality

The runtime compiler allows inclusion of double quoted (" ") 'C' language style functions within unformatted input files. The functions are evaluated during program setup or execution to calculate independent solution variables.

This provides the user with an endlessly flexible method for describing boundary conditions, initial conditions, source terms, material properties, or any other independent variable.

The specific variable names expected within runtime compiled functions depends on the host code and the context of the function use. In general it should be remembered that the runtime functions return quantities by modifying variables that are passed in by reference.

E.1 The RTC language

The RTC language can be thought of as a small subset of the C language with a few minor modifications.

E.1.1 Operators

The RTC language has the following operators that work exactly as they do in C and have the same precedence as they do in C:

- $+$ Addition
- $-$ Subtraction
- $-$ Negation
- $*$ Multiplication
- $/$ Division
- $==$ Equality
- $>$ Greater than
- $<$ Less than
- $>=$ Greater than or equal to
- $<=$ Less than or equal to
- $=$ Assignment
- $||$ Logical or
- && Logical and
- $! =$ Inequality
- % Modulo
- ! Logical not

The following operators do not occur in the C language, but were added to the RTC language for convenience:

- $\hat{}$ Exponentiation

E.1.2 Control flow

The RTC language has the following control flow statements:

- for(expr ; expr ; expr) { ... }
- while(expr) { ... }
- if (expr) {...}
- else if (expr) {...}
- else {...}

These control flow statements work exactly as they do in C except that the code blocks following a control flow statement **MUST** be enclosed within braces even if the block only consists of one line.

E.1.3 Line Structure

The line structure in the RTC language is the same as that of C. Expressions end with a semicolon unless they are inside a control flow statement.

E.1.4 Variable Types and Default Variables

Declaring scalar variables in RTC is done exactly as it is done in C except that only the following types are supported:

- int
- float
- double
- char

For scalars, variables can be declared and assigned all at once. Both of the following approaches will work:

```
int myVar = 9;
```

```
OR
```

```
int myVar;
myVar = 9;
```

Arrays work a little differently in RTC than they do in C. There are no *new* or *malloc* operators, instead the user may declare dynamically sized arrays in the same manner as statically sized arrays. Also, in C all the values of an array may be initialized at once by putting the values within braces. This is not supported in the RTC language. Users will have to loop through the array and assign the values one by one. For example:

LEGAL:

```
int ia[x*y];  //Note: in C this would not be legal for non-const x,y
int ia2[3];
```

NOT LEGAL:

```
int ia[3] = {1, 2, 3};
```

Indexing arrays can be done using the index operator: array[expr] = ...;

Bounds checking is done at run time. If the bounds of an array are exceeded, it will dump an error to stdout.

For all user-defined initial conditions the following default variables are available:

- coord - An array of coordinates. Use an index of zero to get the x coordinate, an index of one to get the y coordinate, and an index of two to get the z coordinate (z is available in 3D only).
- field - This is the means by which the function returns its results. The variable (ex: density, velocity) specified above the function is set according to the values of the field array. If a scalar variable is being set, then the value should be assigned to field[0].

E.1.5 Math

The following math.h functions are available in RTC:

- asin(arg) : returns the arc sine of arg
- acos(arg) : returns the arc cosine of arg
- atan(arg) : returns the arc tangent of arg

- atan2(y, x): returns the arc tangent of y/x
- sin(arg) : returns the sine of arg
- cos(arg) : returns the cosine of arg
- tan(arg) : returns the tangent of arg
- sqrt(arg) : returns the square root of arg
- exp(arg) : returns the natural logarithm base e raised to the arg power
- sinh(arg) : returns the hyperbolic sine of arg
- cosh(arg) : returns the hyperbolic cosine of arg
- tanh(arg) : returns the hyperbolic tangent of arg
- log(arg) : returns the natural logarithm for arg
- log10(arg) : returns the base 10 logarithm for arg
- rand() : returns a system-generated random integer between 0 and RAND_MAX
- drand() : returns a system-generated random double between 0.0 and 1.0
- fabs(arg) : returns the absolute value of arg
- pow(b, e) : returns b to the e power (Note: the Exponentiation operator is available)
- j0(arg) : Bessel function of order zero
- j1(arg) : Bessel function of order one
- i0(arg) : Modified Bessel function of order zero
- i1(arg) : Modified Bessel function of order one
- erf(arg) : Error function
- erfc(arg) : Complementary error function (1.0 - erf(x))
- gamma(arg) : returns $\Gamma(arg)$

E.1.6 Strings

The user may pass quoted strings as arguments to functions. Note: it may be necessary to escape-out the double quotes so that they do not confuse the input-file parser. See printf section below for an example.

E.1.7 Printf

The RTC printf method is called just like its C counterpart. The first argument is a quoted character string. This string will contain the % symbol which will tell RTC to output the corresponding argument. The only difference between RTC's printf and C's printf is that in RTC's version, a type character after the % is unnecessary. For example, inside an RTC method the following is appropriate:

```
printf(\"One:% Two:% Three:% \", 5-4, 2.0e0, 'c');
```

Which would generate this output: One:1 Two:2 Three:c

E.1.8 Comments

The traditional C-comment mechanism may be used inside RTC functions. Use /* to begin a comment and */ to end the comment.

E.1.9 Unsupported Features

Implementing the entire C-language was well beyond the intent of RTC. Features that were too difficult or did not add enough value have been left out. The following is a list of common C features that are unsupported in RTC:

- There are no ++ or −− operators. Use $i = i + 1$ instead of $+ + i$
- Structs
- Pointers
- Instant array initialization: int array[5] = 1,2,3,4,5;
- Case statements
- Casting
- Labels and gotos
- Function definition/declaration
- stdio
- Keywords: break, continue, const, enum, register, return, sizeof, typedef, union, volatile, static.

E.1.10 Examples

The following series of examples illustrate the use of the RTC language within the context of some simple `USER DEFINED INITIAL CONDITIONS`.

Example 1

```
USER DEFINED INITIAL CONDITION, DENSITY
  "field[0] = 5000.0 + (1.0 / (coord[0] + atan2(2,3)));"
END
```

In this example the density of every element is set equal to 5000 plus one over the x coordinate of the element plus the arc tangent of 2 and 3.

Example 2

```
USER DEFINED INITIAL CONDITION, DENSITY, BLOCK 5
"
  double sum = coord[0] + coord[1] + coord[2];
  field[0] = sum / 0.0001;
"
END
```

In this example, the initial density of elements in block 5 is set to to the sum of the x, y, and z coordinates divided by 0.0001;

Example 3

```
USER DEFINED INITIAL CONDITION, DENSITY, BLOCK 5
"
  double newarray[10];
  field[0] = 0;
  for (int i = 0; i < 10; i = i + 1) {
    newarray[i] = -sin(-i*2) + 2;
  }
  for (int i = 0; i < 10; i = i + 1) {
    field[0] = field[0] + newarray[i % 10];
  }
"
END
```

This example shows how to use for-loops and arrays in the RTC language. The density of the elements in block 5 is being set to $\sum_{i=0}^{9}(-\sin(-2i)+2)$

Example 4

```
USER DEFINED INITIAL CONDITION, VELOCITY, BLOCK 5 10
"
   if ( fabs(coord[0]) < 1.10 ) {
     field[0] = 0.;
     field[1] = 0.;
     field[2] = 0.;
   }
   else {
     field[0] = 100.;
     field[1] = 200.;
     field[2] = 300.;
   }
"
END
```

This example uses the absolute value of the x-coordinate of the element. If this value is less that 1.10, the velocity is set to zero in each direction, otherwise the velocity is set to 100,200,300.

Since velocty is not a scalar value, assignments must be made to several indices of the field array.

E.2 Using RTC and APREPRO

Frequently ALEGRA users will place `aprepro` constructs into their input decks and then preprocess the input deck with `aprepro` by issuing the command:

```
Alegra -a runid.inp
```

A problem may exist with curly braces, { and }, in the runtime compiler coding as in the above examples. When the input deck is sent through `aprepro`, the preprocessor will evaluate expressions in curly braces, and the braces will not appear in the processed input deck read by ALEGRA. This will cause an error when the runtime compiler processes the coding.

There are three solutions:

1. Place the following lines before and after the runtime compiler coding so that `aprepro`

will copy the input lines to the output exactly as they are written:

```
${VERBATIM(ON)}
    ... runtime compiler coding ...
${VERBATIM(OFF)}
```

2. Omit the verbatim commands, but put the curly braces into string expressions that will be processed by `aprepro`. Make the following substitutions:

```
{   ->   {"{"}
}   ->   {"}"}
```

   The outer pair of opening and closing braces will be processed by `aprepro`, but the inner brace in quotes will be sent as a string to the output deck.

3. A backslash can also be placed in front of curly braces. This will tell APREPRO to ignore the curly brace. The RTC parser knows to ignore the backslash but not the curly brace. This method will work regardless of whether aprepro is run on the input deck or not.

For instance,

```
${VERBATIM(ON)}
user defined initial condition, density, block 5
"
  field[0] = 100.0;
  if(coord[0] > 0.0){
    double distance = sqrt ((coord[0]^2) + (coord[1]^2) + (coord[2]^2));
    field[0] = field[0] + distance;
  }
"
${VERBATIM(OFF)}
```

or it could be written as,

```
user defined initial condition, density, block 5
"
  field[0] = 100.0;
  if(coord[0] > 0.0) {"{"}
    double distance = sqrt ((coord[0]^2) + (coord[1]^2) + (coord[2]^2));
    field[0] = field[0] + distance;
   {"}"}
"
```

or it could be written as,

```
user defined initial condition, density, block 5
"
  field[0] = 100.0;
  if(coord[0] > 0.0) \{
    double distance = sqrt ((coord[0]^2) + (coord[1]^2) + (coord[2]^2));
    field[0] = field[0] + distance;
  \}
"
```

E.3 Using RTC and ALEGRA Functions

Another useful feature is the ability to call ALEGRA functions from within the RTC (see Section 6.2 on page 52). All user-defined ALEGRA functions are registered with the RTC and any tabular data can be read and interpolated. The ALEGRA function is accessed through the RTC function interface `Function_Evaluate(real,int,int)`. The first `real` argument is the abscissa value at which to evaluate the function. The second `int` argument is the function-id number. The third `int` argument is the function evaluation type. The value 0 will return the function ordinate value, and the value 1 will return the function derivative value.

The following example illustrates the use of the ALEGRA functions within the RTC language in the context of a user defined initial condition.

```
user defined initial condition, density, block 1,
  " double scale = 2.49692e-06 / 2.5126e-06;
    double dens  = Function_Evaluate(coord[0],11,0);
    field[0] = dens * scale ; "
end
```

F Templates for Hybrid Meshes

The following is a starting template for generating an unstructured mesh wrapped with structured hex which can couple with STDEM, using Cubit 15.2 or later. This example creates a simple spherical scatterer.

```
reset
$ Scattering sphere
create sphere radius 0.005
```

```
$ PW launcher and total/scattered field boundary sphere (could be any shape)
create sphere radius 0.015

$ Size of wrapper box
${wsize=0.04}

$ Wrapper box
brick x {wsize} y {wsize} z {wsize}

$ Mesh delta
${dx=.005}

$ 2-cell thick hex wrapper
brick x {wsize+4.*dx} y {wsize+4.*dx} z {wsize+4.*dx}

${zshift = 0.}
move vol all z {zshift}

chop vol 4 with vol 3
chop vol 5 with vol 2
chop vol 7 with vol 1

webcut vol 6 with sheet extended from surf 3
webcut vol 11 with sheet extended from surf 4

imprint vol all
merge all

vol 6 11 12 size {dx}

surf 24 26 scheme map
mesh surf 24 26
vol 6 scheme sweep source surf 24 target surf 26
mesh vol 6

surf 38 40 scheme map
mesh surf 38 40
vol 12 scheme sweep source surf 38 target surf 40
mesh vol 12

vol 11 scheme sweep source surf 27 target surf 32
mesh vol 11

vol 8 9 10 scheme tetmesh
```

```
surf 3 to 8 scheme map
mesh surf 3 to 8

vol 8 9 10 size {dx}

mesh vol 8 9 10
merge all

block 1 vol 9 10
block 2 vol 8
block 3 vol 6 11 12

sideset 1 surf 1 wrt vol 10
sideset 2 surf 2 wrt vol 10
sideset 3 surf 3 to 8

export mesh "meshHybrid.gen" overwrite
```

The following is a starting template for generating the structured mesh using the Quicksilver preprocessor Mercury.

```
^char comment #

^ifndef nproc then
  ^def nproc 1
^endif
^if $nproc gt 1 then
  PROCESSORS $nproc no-edge-connect
^endif

^define dx 5.0e-3
^define dy 5.0e-3
^define dz 5.0e-3

^define nx 16
^define ny 16
^define nz 16

^define ztrans 0.

^define xmax $(nx*dx/2)
^define xmin -$xmax
^define ymax $(ny*dy/2)
^define ymin -$ymax
```

```
^define zmax $(nz*dz/2)
^define zmin -$zmax

^define npml 12
^define npmlx $npml
^define npmly $npml
^define npmlz $npml

^define xminp $(xmin - npmlx*dx)
^define xmaxp $(xmax + npmlx*dx)
^define yminp $(ymin - npmly*dy)
^define ymaxp $(ymax + npmly*dy)
^define zminp $(zmin - npmlz*dz)
^define zmaxp $(zmax + npmlz*dz)

SYSTEM CARTESIAN
BLOCK $xmin $ymin $zmin  $xmax $ymax $zmax
GRID  1  I  $xmin        $nx  $dx  0.0
GRID  1  J  $ymin        $ny  $dy  0.0
GRID  1  K  $zmin        $nz  $dz  0.0

BLOCK pml_block1  $xminp $yminp $zminp  $xmin  $ymaxp $zmaxp  # xmin:full y & z
BLOCK pml_block2  $xmax  $yminp $zminp  $xmaxp $ymaxp $zmaxp  # xmax:full y & z
BLOCK pml_block3  $xmin  $yminp $zminp  $xmax  $ymin  $zmaxp  # ymin:full z
BLOCK pml_block4  $xmin  $ymax  $zminp  $xmax  $ymaxp $zmaxp  # ymax:full z
BLOCK pml_block5  $xmin  $ymin  $zminp  $xmax  $ymax  $zmin   # zmin
BLOCK pml_block6  $xmin  $ymin  $zmax   $xmax  $ymax  $zmaxp  # zmax

GRID pml_block1  I   $xminp  $npmlx  $dx   0.0
GRID pml_block2  I   $xmax   $npmlx  $dx   0.0
GRID pml_block3  J   $yminp  $npmly  $dy   0.0
GRID pml_block4  J   $ymax   $npmly  $dy   0.0
GRID pml_block5  K   $zminp  $npmlz  $dz   0.0
GRID pml_block6  K   $zmax   $npmlz  $dz   0.0

^# define sigma(x) function

^define delta $dx
^define smax $(1.0/(150.0 * 3.14159 * delta))
^define sfact $(smax/8.854e-12)
^define funpar [$(4*npmlx+4)]

^# xmin section

^define ifun 0
```

```
^define funpar[$ifun]     $(xminp-dx)
^define funpar[$(ifun+1)] $sfact
^define ifun $(ifun+2)
^define x1 $(xmin-dx)
^for i 0 $(npmlx-2)
  ^define x $(xminp + i*dx)
  ^define u $((x1-x)/(dx+x1-xminp))
  ^define funpar[$ifun]   $x
  ^define funpar[$(ifun+1)] $(sfact*u^2)
  ^define ifun $(ifun+2)
^endfor

^# center section
^define funpar[$ifun]    $x1
^define funpar[$ifun+1]  0.0
^define funpar[$ifun+2]  $(xmax+dx)
^define funpar[$ifun+3]  0.0
^define ifun $(ifun+4)

# xmax section
^define x0 $(xmax+dx)
^for i 1 $(npmlx-1)
  ^define x $(x0 + i*dx)
  ^define u $((x-x0)/(dx+xmaxp-x0))
  ^define funpar[$ifun]   $x
  ^define funpar[$ifun+1] $(sfact*u^2)
  ^define ifun $(ifun+2)
^endfor
^define funpar[$ifun]     $(xmaxp+dx)
^define funpar[$(ifun+1)] $sfact

FUNCTION 0 $(xminp - delta) $(xmaxp + delta) bowl $funpar

^# define sigma(y) function

^define delta $dy
^define smax $(1.0/(150.0 * 3.14159 * delta))
^define sfact $(smax/8.854e-12)
^define funpar [$(4*npmly+4)]

^# ymin section

^define ifun 0
^define funpar[$ifun]     $(yminp-dy)
^define funpar[$(ifun+1)] $sfact
```

```
^define ifun $(ifun+2)
^define y1 $(ymin-dy)
^for i 0 $(npmly-2)
  ^define y $(yminp + i*dy)
  ^define u $((y1-y)/(dy+y1-yminp))
  ^define funpar[$ifun]   $y
  ^define funpar[$(ifun+1)] $(sfact*u^2)
  ^define ifun $(ifun+2)
^endfor

^# center section
^define funpar[$ifun]    $y1
^define funpar[$ifun+1]  0.0
^define funpar[$ifun+2]  $(ymax+dy)
^define funpar[$ifun+3]  0.0
^define ifun $(ifun+4)

# ymax section
^define y0 $(ymax+dy)
^for i 1 $(npmly-1)
  ^define y $(y0 + i*dy)
  ^define u $((y-y0)/(dy+ymaxp-y0))
  ^define funpar[$ifun]   $y
  ^define funpar[$ifun+1] $(sfact*u^2)
  ^define ifun $(ifun+2)
^endfor
^define funpar[$ifun]     $(ymaxp+dy)
^define funpar[$(ifun+1)] $sfact

FUNCTION 0 $(yminp - delta) $(ymaxp + delta) bowly $funpar

^# define sigma(z) function

^define delta $dz
^define smax $(1.0/(150.0 * 3.14159 * delta))
^define sfact $(smax/8.854e-12)
^define funpar [$(4*npmlz+4)]

^# zmin section

^define ifun 0
^define funpar[$ifun]     $(zminp-dz)
^define funpar[$(ifun+1)] $sfact
^define ifun $(ifun+2)
^define z1 $(zmin-dz)
```

```
^for i 0 $(npmlz-2)
  ^define z $(zminp + i*dz)
  ^define u $((z1-z)/(dz+z1-zminp))
  ^define funpar[$ifun]   $z
  ^define funpar[$(ifun+1)] $(sfact*u^2)
  ^define ifun $(ifun+2)
^endfor

^# center section
^define funpar[$ifun]    $z1
^define funpar[$ifun+1]  0.0
^define funpar[$ifun+2]  $(zmax+dz)
^define funpar[$ifun+3]  0.0
^define ifun $(ifun+4)

# zmax section
^define z0 $(zmax+dz)
^for i 1 $(npmlz-1)
  ^define z $(z0 + i*dz)
  ^define u $((z-z0)/(dz+zmaxp-z0))
  ^define funpar[$ifun]   $z
  ^define funpar[$ifun+1] $(sfact*u^2)
  ^define ifun $(ifun+2)
^endfor
^define funpar[$ifun]     $(zmaxp+dz)
^define funpar[$(ifun+1)] $sfact

FUNCTION 0 $(zminp - delta) $(zmaxp + delta) bowlz $funpar

^define dtcour $(1.0/sqrt(1/dx^2 + 1/dy^2 + 1/dz^2) / 3.0e8)
^define dt $(0.9*dtcour)
TIMESTEP   $dt

# material numbers (1-dummy, 2-ground, 3-vacuum)

CONDUCTOR CONFORMAL 0.0 0.0 0.0 $dx $dy $dz dummy

PEC  $xminp $yminp $zminp  $xminp $ymaxp $zmaxp  ground
PEC  $xmaxp $yminp $zminp  $xmaxp $ymaxp $zmaxp  ground
PEC  $xminp $yminp $zminp  $xmaxp $yminp $zmaxp  ground
PEC  $xminp $ymaxp $zminp  $xmaxp $ymaxp $zmaxp  ground
PEC  $xminp $yminp $zminp  $xmaxp $ymaxp $zminp  ground
PEC  $xminp $yminp $zmaxp  $xmaxp $ymaxp $zmaxp  ground
```

The following is the resulting emphasis.inp file created using the above template with Mercury.

```
TITLE
  emphasis
TERMINATION CYCLE=         0
EMIT HISPLT, CYCLE INTERVAL=    1 FROM TIME   8.6603E-12 TO   8.6603E-04
STRUCTURED TD ELECTROMAGNETICS
  MESH, PFF
    FILE="emphasis.pff"
  END
  FIELD SOLVER, EXPLICIT
  CONSTANT TIME STEP=  8.66025E-12
  FUNCTION     1
    0.00000E+00    0.00000E+00
    1.00000E+30    0.00000E+00
  END
  FUNCTION     2
    0.00000E+00    1.00000E+00
    1.00000E+30    1.00000E+00
  END
  FUNCTION     3
   -1.05000E-01    4.79347E+10
   -1.00000E-01    4.02785E+10
   -9.50000E-02    3.32880E+10
   -9.00000E-02    2.69633E+10
   -8.50000E-02    2.13043E+10
   -8.00000E-02    1.63111E+10
   -7.50000E-02    1.19837E+10
   -7.00000E-02    8.32201E+09
   -6.50000E-02    5.32607E+09
   -6.00000E-02    2.99592E+09
   -5.50000E-02    1.33152E+09
   -5.00000E-02    3.32880E+08
   -4.50000E-02    0.00000E+00
    4.50000E-02    0.00000E+00
    5.00000E-02    3.32880E+08
    5.50000E-02    1.33152E+09
    6.00000E-02    2.99592E+09
    6.50000E-02    5.32607E+09
    7.00000E-02    8.32201E+09
    7.50000E-02    1.19837E+10
    8.00000E-02    1.63111E+10
    8.50000E-02    2.13043E+10
```

```
  9.00000E-02    2.69633E+10
  9.50000E-02    3.32880E+10
  1.00000E-01    4.02785E+10
  1.05000E-01    4.79347E+10
END
FUNCTION     4
 -1.05000E-01    4.79347E+10
 -1.00000E-01    4.02785E+10
 -9.50000E-02    3.32880E+10
 -9.00000E-02    2.69633E+10
 -8.50000E-02    2.13043E+10
 -8.00000E-02    1.63111E+10
 -7.50000E-02    1.19837E+10
 -7.00000E-02    8.32201E+09
 -6.50000E-02    5.32607E+09
 -6.00000E-02    2.99592E+09
 -5.50000E-02    1.33152E+09
 -5.00000E-02    3.32880E+08
 -4.50000E-02    0.00000E+00
  4.50000E-02    0.00000E+00
  5.00000E-02    3.32880E+08
  5.50000E-02    1.33152E+09
  6.00000E-02    2.99592E+09
  6.50000E-02    5.32607E+09
  7.00000E-02    8.32201E+09
  7.50000E-02    1.19837E+10
  8.00000E-02    1.63111E+10
  8.50000E-02    2.13043E+10
  9.00000E-02    2.69633E+10
  9.50000E-02    3.32880E+10
  1.00000E-01    4.02785E+10
  1.05000E-01    4.79347E+10
END
FUNCTION     5
 -1.05000E-01    4.79347E+10
 -1.00000E-01    4.02785E+10
 -9.50000E-02    3.32880E+10
 -9.00000E-02    2.69633E+10
 -8.50000E-02    2.13043E+10
 -8.00000E-02    1.63111E+10
 -7.50000E-02    1.19837E+10
 -7.00000E-02    8.32201E+09
 -6.50000E-02    5.32607E+09
 -6.00000E-02    2.99592E+09
 -5.50000E-02    1.33152E+09
```

```
   -5.00000E-02    3.32880E+08
   -4.50000E-02    0.00000E+00
    4.50000E-02    0.00000E+00
    5.00000E-02    3.32880E+08
    5.50000E-02    1.33152E+09
    6.00000E-02    2.99592E+09
    6.50000E-02    5.32607E+09
    7.00000E-02    8.32201E+09
    7.50000E-02    1.19837E+10
    8.00000E-02    1.63111E+10
    8.50000E-02    2.13043E+10
    9.00000E-02    2.69633E+10
    9.50000E-02    3.32880E+10
    1.00000E-01    4.02785E+10
    1.05000E-01    4.79347E+10
  END
  ELECTRICAL CONDUCTIVITY THRESHOLD= 1.0e6
END

MATERIAL    1 dummy
  MODEL=    1
END
MATERIAL    2 ground
  MODEL=    1
END
MATERIAL    3 vacuum
  MODEL=    3
END
MODEL    1 SIMPLE ELECTRICAL
  EPS=  1.0000E+00
  MU=  1.0000E+00
  SIGMA=  1.0000E+07
END
MODEL    3 SIMPLE ELECTRICAL
  EPS=  0.0000E+00
  MU=  1.0000E+00
  SIGMA=  0.0000E+00
END

EXIT
```

The following is a complete EMPHASIS input file for this sphere scattering problem.

```
TITLE
```

```
  Planewave scattering from a PEC sphere w/far-field

DEBUG MODE, CONNECTIVITY=1

HYBRID TD ELECTROMAGNETICS

$-------------------------------- UTDEM --------------------------------

  MESH, GENESIS
    FILE = "hpwscatterHybrid.gen"
  END

  FORMULATION, SECOND ORDER
  HEX MASS LUMP = yes
  WRAPPER, SIDESET 3

  BLOCK 1           $ total
    MATERIAL 1
  END
  BLOCK 2           $ scattered/wrapper
    MATERIAL 1
  END
  BLOCK 100002      $ pyramid
    MATERIAL 1
  END
  BLOCK 3           $ hex
    MATERIAL 1
  END

  PEC BC, SIDESET 1

  SURFACE CURRENT, SIDESET 1

  PLANE WAVE SOURCE, SIDESET 2, BLOCK 1 FUNCTION 101
    POLARIZATION, X=1.0 Y=0.0 Z=0.0, PROPDIR, X=0.0 Y=0.0 Z=1.0

  TRACER POINTS
    EULERIAN TRACER 1 X=0.0 Y=0.0 Z=-10.0e-3
    EULERIAN TRACER 2 X=0.0 Y=0.0 Z=-17.5e-3
  END

  FUNCTION 101 GAUSSIAN, SCALE=1.0 WIDTH=0.23263e-9 SHIFT=0.73048e-9

  GRADUAL STARTUP FACTOR 1.0
```

```
$-------------------------------- STDEM ---------------------------------

  MESH, PFF
    FILE="emphasis.pff"
  END

$ DIELECTRIC ALGORITHM, CONSTANT=1.0
  ELECTRICAL CONDUCTIVITY THRESHOLD=1e6

  SURFACE FIELD CHECK, FIELDS=EB
  FIELD SOLVER, PML, PROFILE=X, FUNCTION=201 Y, FUNCTION=201 Z, FUNCTION=201
    BLOCK=1,2,3,4,5,6

  FUNCTION 201
   -1.05000E-01    4.79347E+10
   -1.00000E-01    4.02785E+10
   -9.50000E-02    3.32880E+10
   -9.00000E-02    2.69633E+10
   -8.50000E-02    2.13043E+10
   -8.00000E-02    1.63111E+10
   -7.50000E-02    1.19837E+10
   -7.00000E-02    8.32201E+09
   -6.50000E-02    5.32607E+09
   -6.00000E-02    2.99592E+09
   -5.50000E-02    1.33152E+09
   -5.00000E-02    3.32880E+08
   -4.50000E-02    0.00000E+00
    4.50000E-02    0.00000E+00
    5.00000E-02    3.32880E+08
    5.50000E-02    1.33152E+09
    6.00000E-02    2.99592E+09
    6.50000E-02    5.32607E+09
    7.00000E-02    8.32201E+09
    7.50000E-02    1.19837E+10
    8.00000E-02    1.63111E+10
    8.50000E-02    2.13043E+10
    9.00000E-02    2.69633E+10
    9.50000E-02    3.32880E+10
    1.00000E-01    4.02785E+10
    1.05000E-01    4.79347E+10
  END

  HISTORY FIELD, LABEL="EI_1" TYPE=POINT, FIELD=ELECTRIC_FIELD_I,
                 SEGMENT=0.0 0.0 -0.035  0.0 0.0 -0.035
  HISTORY FIELD, LABEL="EJ_1" TYPE=POINT, FIELD=ELECTRIC_FIELD_J,
```

```
                SEGMENT=0.0 0.0 -0.035  0.0 0.0 -0.035
  HISTORY FIELD, LABEL="EK_1" TYPE=POINT, FIELD=ELECTRIC_FIELD_K,
                SEGMENT=0.0 0.0 -0.035  0.0 0.0 -0.035

  HISTORY FIELD, LABEL="BI_1" TYPE=POINT, FIELD=MAGNETIC_FLUX_DENSITY_I,
                SEGMENT=0.0 0.0 -0.035  0.0 0.0 -0.035
  HISTORY FIELD, LABEL="BJ_1" TYPE=POINT, FIELD=MAGNETIC_FLUX_DENSITY_J,
                SEGMENT=0.0 0.0 -0.035  0.0 0.0 -0.035
  HISTORY FIELD, LABEL="BK_1" TYPE=POINT, FIELD=MAGNETIC_FLUX_DENSITY_K,
                SEGMENT=0.0 0.0 -0.035  0.0 0.0 -0.035

  FAR FIELD, SEGMENT=-0.035 -0.035 -0.035 0.035 0.035 0.035, PHASECENTER=0.0 0.0 -0.015,
  FAR FIELD PATTERN, PHISTART=0., THETASTART=-180., THETAEND=180., NUMTHETA=361, FREQUEN
  FAR FIELD PATTERN, PHISTART=90., THETASTART=-180., THETAEND=180., NUMTHETA=361, FREQUE

$-------------------------------------------------------------------------

  CONSTANT TIME STEP 4.0e-12

END

UNITS = si

TERMINATION TIME 2000e-12

AZTEC
  SOLVER = cg
  PRECOND = jacobi
  OUTPUT = none
  TOL = 1.0e-8
  POLYNOMIAL ORDER = 1
END

$
$-------------------------------------------------------------------------
$                           P L O T T I N G
$-------------------------------------------------------------------------
$
EMIT SCREEN, CYCLE INTERVAL = 1

EMIT HISPLT, CYCLE INTERVAL = 1
HISTORY PLOT VARIABLES
  ELECTRIC FIELD
  MAGNETIC FIELD
END
```

```
EMIT PLOT, TIME INTERVAL = 8e-12, FROM TIME 160e-12 TO 2000e-12
PLOT VARIABLES
  NO DEFAULT OUTPUT
  ALL REGION VARIABLES
  ELECTRIC FIELD
  MAGNETIC FIELD
  SURFACE CURRENT DEN, AS "JS"
END

$
$-----------------------------------------------------------------------
$                            M A T E R I A L S
$-----------------------------------------------------------------------
$

MATERIAL 1 vacuum
  MODEL = 1
END

MATERIAL 2 cond
  MODEL = 2
END

MATERIAL 3 vacuum
  MODEL = 1
END

MODEL 1 SIMPLE ELECTRICAL
  EPS = 1.0
  MU = 1.0
  SIGMA = 0.0
END

MODEL 2 SIMPLE ELECTRICAL
  EPS = 1.0
  MU = 1.0
  SIGMA = 1e7
END

EXIT
```

DISTRIBUTION:

1	MS 1152	K. L. Cartwright, 1352
1	MS 1152	R. S. Coats, 1352
1	MS 1152	J. D. Kotulski, 1352
1	MS 1152	T. D. Pointon, 1352
1	MS 1152	C. D. Turner, 1352
1	MS 0899	Technical Library (electronic copy), 9536

Sandia National Laboratories