SANDIA REPORT

SAND2014-16735
Unlimited Release
Printed August 2014

EMPHASIS™/Nevada UTDEM User Guide Version 2.1.1

C. David Turner, Michael F. Pasik, David B. Seidel,
Timothy D. Pointon, Keith L. Cartwright

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

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

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 order: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2014-16735
Unlimited Release
Printed August 2014

EMPHASIS/Nevada UTDEM User Guide Version 2.1.1

C. David Turner, Michael F. Pasik, David B. Seidel
Timothy D. Pointon, Keith L. Cartwright

Electromagnetics and Plasma Physics Analysis

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.

Acknowledgement

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

List of Figures

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 [1].

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.

UTDEM Simulation Process

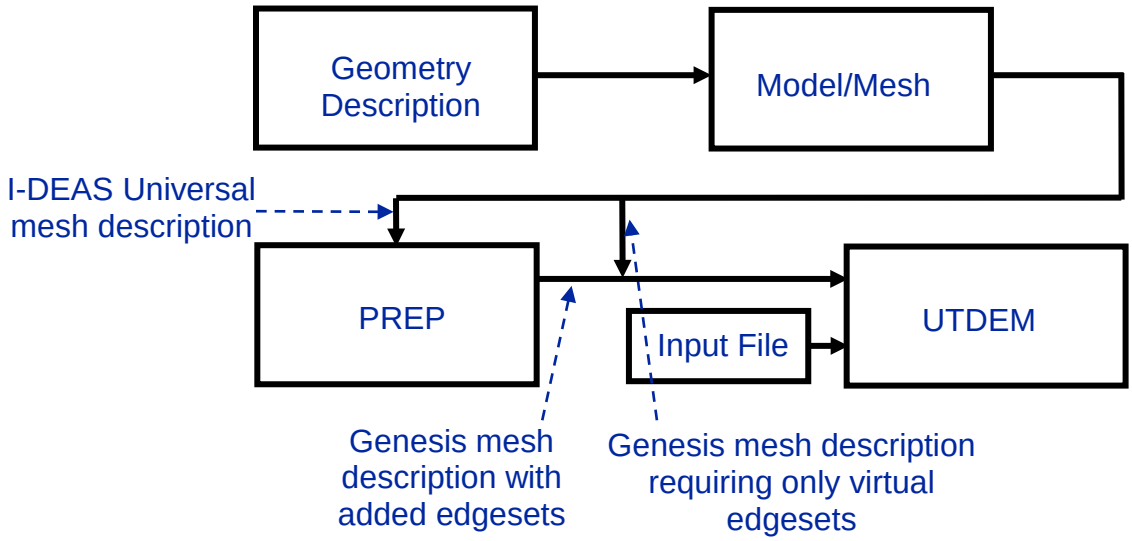

Figure 1. UTDEM Simulation Process.

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 [2] format. CUBIT and ICEM do this directly as does I-DEAS with the addition of a special 3rd 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 [3]. 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 I.

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.

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 Blot, in the case of plot dumps.

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 [4]. A complete input file for one of the UTDEM regression problems is given in Appendix II.

```
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).

Solver formulation keyword

formulation

Specifies the UTDEM solver formulation for this simulation

formulation, *type*

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 ½ time step.

second order artuzi, static limit

Utilize the **second order** formulation above with Artuzi late-time stability correction and the Artuzi rhs term $[S]w^n$ 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 [5], instead of the Newmark-Beta method used for standard **second order**. The damping parameter **theta** (required) should be in the range, $0 <= \theta <= 1$, 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 $\theta \rightarrow 1$, 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, $\omega_p \Delta t <$ ~1, where $\omega_p = (ne^2/\varepsilon_0)^{1/2}$, and *n* is the plasma density in C/m^3.

first order

Utilize the conditionally stable, coupled 1st order formulation (not fully implemented)

For additional information on these options consult the UTDEM theory guide [1].

condition number

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

FEM basis order keyword

basis order

Specifies the order of the FEM basis or shape functions

basis order, *int*

The default basis uses the Whitney tangential vector finite element with one degree of freedom per edge, specified with **basis order, 0** or with no **basis order** keyword. This is occasionally referred to as a 0th order or 1st order mixed element. These shape functions are constant along an edge and linear across the element. In reality, they behave more like 0th than 1st order as the spatial convergence is observed to be nearly linear.

An optional **basis order, 1** is available which adds one additional degree of freedom per edge by, for tetrahedrons, adding six additional hierarchical shape functions to the original six Whitney functions. This 1st order full element achieves full 2nd 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 it's use with hybrid meshing. It has been successfully tested with all algorithms in the code except for the unconditionally stable wire.

Solver option keywords for PIC

godfrey alpha1

Specifies the coefficient of the forward time in the field solve

Godfrey alpha1, *real*

With **formulation second order** one can specify **godfrey alpha1** $\geq$ 0.25. The default is 0.25 and yields the Newmark-Beta time advance [6]. 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 Ref. 6 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.

Boundary condition keywords

pec bc

Specifies a perfect electric conductor boundary condition

pec bc, sideset *int*

The electric fields tangential to the surface described by the sideset having id **sideset** 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

Specifies a perfect magnetic conductor or mirror-symmetry boundary condition

pmc bc, sideset *int*

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 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.

abc bc

Specifies an absorbing boundary condition

abc bc, sideset *int*

A 1st order absorbing boundary condition will be applied over the surface described by the sideset having id **sideset**. 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.

ibc bc

Specifies an impedance boundary condition

ibc bc, sideset *int***, impedance** *real*

A 1st order surface impedance boundary condition with surface impedance value **impedance** will be applied over the surface described by the sideset having id **sideset**. 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.

Boundary condition precedence is presently specified in the code as follows: PEC->IBC->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.

Virtual edgeset keyword

path

Specifies a virtual edgeset to be created along an arbitrary user-defined path through the geometry, independent of mesh topology.

path, edgeset *int*, **point**, **x**=*real* **y**=*real* **z**=*real* **point, x**=*real* **y**=*real* **z**=*real* **point, . . .**

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.

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

Observer keywords

observer

The following syntax specifies the simplest type of single-edge, electric-field-projection observer:

observer, nodeset *int*

The edge will be located which is defined by the nodeset having id **nodeset**. 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 **nodeset** id parameter from the **observer** line 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

The following syntax specifies the slot voltage observer:

slot voltage observer, nodeset *int*, [**direction x** *real* **y** *real* **z** *real*]

The slot voltage at the node defined by the nodeset having id **nodeset** 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 **nodeset** id parameter from the **observer** line 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

The following syntax specifies the wire current observer:

wire current observer, nodeset *int*, [**direction x** *real* **y** *real* **z** *real*]

The wire current at the node defined by the nodeset having id **nodeset** 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 **nodeset** id parameter from the **observer** line 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

The following syntax specifies an electric-field line-integral observer, $\int \overline{E} \bullet dl$:

e line integral, edgeset *int,* **direction x** *real* **y** *real* **z** *real*

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 **edgeset** id parameter from the **observer** line 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: **E*DL-***edgeset_id*.

h line integral

The following syntax specifies a magnetic-field line-integral observer, $\oint \overline{H} \bullet dl$:

h line integral, edgeset *int*

In this case, a direction vector is not required but the **user is responsible** for creating the mesh such that the beam elements defining the edgeset are oriented in the correct direction around the integration loop. 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, which is stored with the edgeset and is derived from

the beam elements in the original mesh defining the edgeset. These dot products on the edges in the edgeset are then summed. Multiple hlinints 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***.hlinint***edgeset_id***.***proc_id***.dat**, where *edgeset_id* is the **edgeset** id parameter from the **hlinint** line 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: **H*DL-***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

The following syntax enables the calculation of a surface current $n \times \overline{H}$ on the nodes of the specified conductor boundary sideset(s)

surface current, sideset *int* [*int* …]

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

The following syntax specifies a maximum-field (or other registered variable) observer:

max observer, id *int*, **name** "*variable_name*" [**block** *int int* …]

where **id** 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
```

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 **nodeset**. 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 **edgeset**. The source will be applied polarized along the direction vector **direction** and scaled to provide the desired magnitude when integrated over the length **length.** The scale factor for each edge in the source is $\overline{edge\_length} \bullet$ **direction** / **length**, where $\overline{edge\_length}$ 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 sideset. The source will be applied polarized along the direction vector **direction** and scaled to provide the desired magnitude when integrated over the length **length.** The scale factor for each edge in the source is $\overline{edge\_length} \bullet$ **direction** / **length**, where $\overline{edge\_length}$ 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

The following syntax specifies a port source:

port source, coaxial, sideset *int***, function** *int***, scale** *real* **shift** *real***, center, x** *real***, y** *real***, z** *real*

or
port source, parallel plate, sideset *int***, function** *int***, scale** *real* **shift** *real***, direction, x** *real* **y** *real* **z** *real***, separ** *real*
or
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 E polarized in the radial direction and the proper $\frac{1}{r \ln \frac{b}{a}}$ variation. 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**/**separation** 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 the section 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

The following syntax specifies a wire voltage source applied at a single wire node:

wire voltage source, nodeset *int***, function** *int***, scale** *real* **shift** *real*

The nodeset MUST contain only 1 node. UTDEM will issue an error if it does not.

The source time history will be written to a file with a generated name in the following format: *problem_name***.wiresrc***nodeset_id.proc_id***.dat**, where *nodeset_id* is the **nodeset** id 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 be written to the hisplt database *problem_name*.**his** with the name: **WIREVOLSRC-***nodeset_id.*

j source

The following syntax specifies a volumetric vector current-density source:

j source, block int [*int int* **…**]**, [function** *int***, scale** *real* **shift** *real*, **direction, x** *real***, y** *real***, z** *real* **center, x** *real***, y** *real***, z** *real*, **atten** *real*, **prop** *bool* or *string*, **propdir x** *real* **y** *real* **z** *real*, **radtrans** *bool* or *string***, annulus** *bool* or *string*, **origin x** *real* **y** *real* **z** *real*, **external**, **file**, *string*, **loadexternalconduct**, *bool* or *string*, **haltexternalsim**, *bool* or *string*, **curdenname**, *string*, **conductname**, *string*]

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(**-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**.

If **external** is specified, this directs the code to look for time planes of nodal current-source data in either the problem genesis file (if no **file** keyword) 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 as well, 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 with **external** specified, no time-history **function** is required. Since this time-history function is the one used to generate frequency-domain far-field patterns by de-convolution, far-field patterns from external j-sources are not supported. Far-field transient waveforms use no de-convolution. One can get an *approximate* pattern by specifying a **function** or by using the default which is a step function.

If **external** is not specified, the source time history will be written to the hisplt database *problem_name*.**his** with the name: **JSRC-1001**. If **external** is specified the time history written to *problem_name*.**his** is not correct. 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.

plane wave source

The following syntax specifies plane-wave source:

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*]

A plane wave will be launched in one or more mesh **block**(s), with polarization **polarization,** propagating in the direction **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.*

Load keywords

edge load

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

edge load, nodeset *int*, *type*, **value** *real*
or
edge load, edgeset *int*, *type*, **value** *real*

If **nodeset** is specified, the edge will be located which is defined by the nodeset having id **nodeset**. 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. 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 observe the voltage across the load, an observer should be assigned to the same **nodeset** using the **observer** keyword.

spice load

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

spice load, nodeset *int*, **deck**, "*filename*"
or
spice load, edgeset *int*, **deck**, "*filename*"

If **nodeset** is specified, the edge will be located which is defined by the nodeset having id **nodeset**. 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.

The SPICE **deck** containing the description of the **spice load** must be provided with the **spice load** keyword.

spice step fraction

The following syntax 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 .

spice step fraction *real*

Slot keyword

slot

The following syntax specifies a single sub-grid, thin-slot model:

slot, edgeset *int*, **aztec_set** *int*, **width** *real*, **depth** *real*,
int_mat *int*, **ext_mat** *int*

The slot is defined to lie along the edges defined by **edgeset** 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 II. 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.

Wire keywords

wire

The following syntax specifies a single sub-grid, thin-wire model:

wire, edgeset *int***, aztec_set** *int***, radius** *real*

The wire is defined to lie along the edges defined by **edgeset** 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 something other than 0, which is reserved for the primary system solve. Multiple wires may exist in the same simulation.

uncstabwire

The following syntax specifies a single sub-grid, unconditionally stable thin-wire model:

uncstabwire, edgeset *int***, aztec_set** *int***, radius** *real*

The wire is defined to lie along the edges defined by **edgeset** 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

The following syntax specifies a single thin-wire lumped resistive load:

wire load, nodeset *int***, R, value** *real*
wire load, edgeset *int***, R, value** *real*

The resistor is defined to lie along the edge defined by the **nodeset** or **edgeset** with resistance **value**.

Hybrid keywords

hybrid td electromagnetics

This physics keyword specifies the hybrid time-domain electromagnetics (HTDEM) physics is to be used to couple structured and unstructured mesh. This goes in place of the **unstructured td electromagnetics** physics keyword (see pages 12-13).

hex mass lump

The following syntax specifies whether mass lumped integration is activated for hex elements:

hex mass lump, *bool* or *string*

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

The following syntax specifies the sideset identifying the boundary of the unstructured mesh:

wrapper, sideset *int*

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.

far field

The following syntax specifies that far-fields are to be computed:

far field, segment=*real real real real real real* [, **phasecenter**=*real real real,* **lookangles** *real real real real*…]

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 (theta, phi) 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

The following syntax specifies that far-field patterns are to be computed:

far field pattern, phistart=*real,* [**phiend**=*real,* **numphi**=*int,*] **thetastart**=*real,* [**thetaend**=*real,* **numtheta**=*int,*] **frequencies** *real real real…*

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.

Box IEMP keyword

box iemp

The following syntax specifies a Box IEMP simulation:

box iemp [, **load current] [,load dose]**

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).

Update Mat State keyword

update mat state

The following syntax specifies whether material state is updated at the end of each time cycle:

update mat state, *bool* or *string*, **tstart** *real* **tend** *real* **interval** *real*

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 keyword

time step mod

The following syntax specifies a global modification to the UTDEM-computed stable time step:

time step mod, *real*

The default is 1. This value does not effect the time step if specified using the framework keyword **constant time step**.

Compute Energy Keyword

compute energy

The following syntax specifies whether or not electric, magnetic, and total energy is computed throughout the computational volume:

compute energy, *bool* or *string*

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.**

Framework keywords within the physics block

These keywords are really framework keywords [4] but must be placed within the physics definition keyword ("unstructured td electromagnetics") block.

block

Define a finite-element block

block *int* **material** *int*

Relates the mesh block id **block** to material definition **material**

function

Define a user-defined function for use by the **source** keyword

function *int*
time0 value0
time1 value1
time2 value2
.
.
.
end

function *int* **gaussian**, **scale** *real* **shift** *real* **width** *real*

$$scale \times \exp\left(-\left((t - shift)/width\right)^2\right)$$

function *int* **double** exponential, **scale** *real* **shift** *real* alpha *real* **beta** *real*

$$scale \times \left(\exp\left(-alpha \times t\right) - \exp\left(-beta \times t\right)\right), \quad t > 0$$

function *int* **sin squared**, **scale** *real* **shift** *real* **width** *real*

$$scale \times \sin^2\left(\frac{\pi \times t}{width}\right), \quad 0 < t < width$$

function *int* **triangle**, **scale** *real* **shift** *real* **width** *real*

$$scale \times \left(\frac{t}{width/2}\right), \quad 0 \le t < width/2$$

$$scale \times \left(1 - \frac{1}{width/2}(t - width/2)\right), \quad width/2 \le t < width$$

function *int* **sine**, **scale** *real* **shift** *real* **frequency** *real*

$$scale \times \sin\left(frequency \times t + shift\right)$$

gradual startup factor

Factor by which the initial time step is multiplied, default is 0.01

gradual startup factor *real*

Gradually increases to the initial time step. For UTDEM, the value is normally set to 1.0.

maximum time step ratio

Maximum ratio by which a time step may grow in a given cycle

maximum time step ratio *real*

Gradually increases the time step from the old to the new value.

constant time step

Specifies a constant time step for the entire simulation

constant time step *real*

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.

domain

Define global Adaptive Mesh Refinement (AMR) behavior.

domain
max level = *int*
initial refinement
block *int***, level** *int*
end
end

The **max level** keyword defines a global maximum refinement level for all blocks for either initial or dynamic refinement. The **block** keyword under the **initial refinement** keyword selects specific blocks for refinement before the simulation begins.

adaptivity specification

adaptivity specification
enable adaptivity
end

The **enable adaptivity** keyword specifies that dynamic AMR is to be activated in the simulation.

Further information on these AMR-related keywords can be found in the ALEGRA user guide [4].

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 above, 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).

PIC-Specific Keywords

The following keywords are available to UTDEM PIC physics but are also available to the STDEM PIC physics model (see [7]):

define species

The **define species** command keyword group provides a means of defining charged-particle species to be used with the various particle source and diagnostic commands (currently only **beam emission** and **particle history**). One or more particle species may be defined in one **define species** keyword group.

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 (see on page 58).

gas drag

The **gas drag** command keyword enables a gas drag model which causes an IONIZATION_RATE variable to be computed in each element from the interaction between the particles and a high pressure gas. 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).

its file

The **its file** command keyword is used to provide the name of the file containing the ITS distribution datasets needed by the **beam emission** command keyword when using the **field distribution**=ITS option.

its file *"string"*

If this keyword is not specified, the name *its.pff* 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.

cross section database

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.

cross section database *"string"*

where *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.

particle history
  history_specification
  …
end

Where **history_specification** has one of the three following forms:

string, [**species**=*string*,] [**status**=*string*,] [**label**="*string*"]
or
string, **kinetic gas model**=*int*, [**species**=*string*,] [**gas**=*string*,]
[**interaction**=*string*,] [**label**="*string*"]

or

kpflux, species=*string*, **sideset**=*int […int]*, **type$_1$**=*label$_1$*, [**type$_2$**=*label$_2$*,] … [**type$_n$**=*label$_n$*,]

Each request has an initial string that gives the type of the request. The 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.

Each request has a **species** keyword to specify 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. The **species** keyword is optional for the first two forms, defaulting to ALL, but a KPF history request requires a single species name to be input (ALL is not an option). The first two forms also use an optional **label** keyword, which 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 keyword values. The KPF history syntax is different, and will be described below.

The first form of the history specification listed above is used to provide global tallies of particle count, charge, or energy, which can be filtered by species and particle status. If the particle history type is MERGE STATS, then the **status** keyword determines the specific statistic requested; legal values are SUCCESS, FAIL, FAIL_REJECT, FAIL_QERR, FAIL_QLOW, and FAIL_EMAX. Note that FAIL indicates the total number of merge failures for any reason – the other four FAIL_... values allow counts of merger failures for each specific failure mechanism. If the particle history type is MERGE COUNTS, then the **status** keyword determines the specific count category requested; legal values are SAMPLED, CREATED, KILLED, and 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 particle history types, the **status** keyword filters the aggregated information based upon the present status of the particles. Legal values for this keyword are CREATED, KILLED, and

SURVIVING, which 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.

The second form of history specification is used to provide 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. The required **kinetic gas model** keyword is used to specify the integer ID of the material model, i.e., the ID supplied with the material model's definition. The **species** keyword is used to specify the electron species of the incident particle in a collision, and has legal values of PRIMARY, SECONDARY, or ALL, where ALL is the default 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. The **gas** keyword is used to limit 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. The **interaction** keyword is used to specify the type of collisions to be included in the history. Legal values are ELASTIC, EXCITATION, IONIZATION, ATTACHMENT, or ALL. If this keyword is not supplied, ALL will be used, indicating that the contributions from all interactions are to be combined.

The KPFLUX history specification is used to provide information on particles killed on boundary surfaces. As mentioned above, a single species is required input. The boundary surface is defined by one or more sideset indices defined with the **sideset** keyword. Note that this keyword is specific to UTDEM; KPFLUX histories are not currently implemented in STDEM. The remaining keywords are one or more **type**=*label* pairs, each one specifying a single history with the specified output label. The legal types are: NUMBER, CURRENT, MEAN_ENERGY, MEAN_PX, MEAN_PY, MEAN_PZ, STDV_ENERGY, STDV_PX, STDV_PY, and STDV_PZ. 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, 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.

UTDEM PIC-Specific Keywords

The following keywords are available only to UTDEM PIC:

beam emission

field emission

These two 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 $E_{normal} = 0$. The particle species and other emission characteristics are provided by the several available parameter options. The two commands, whose formats are shown below, 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 the two commands.

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 *rea*l|
ITS|ITS_SURFACE [*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*]

field emission, **sideset** *int* **species**=*string*
[**cycle interval** *int*] [**count** *int*] [**emit probability** *real*] [**spatial distribution**=FIXED|RANDOM] **energy distribution**=
CONSTANT *real*|RANDOM *real* [*real*]
[**angle distribution**=NORMAL|CONSTANT, X=*real* Y=*real* Z=*rea*l|
RANDOM [*real real*]|COSINE] [**normal tolerance** *real*] **height distribution**= FIXED *real*|RANDOM *real* [*real*] **breakdown** *real*]
[**qpmin_fun** *function-set*] [**qpmin_floor** *real*]

Options common to **beam emission** and **field emission** commands:

The **sideset** keyword is used to provide the number of the sideset in the Genesis input file that provides the set of faces that comprise the emission surface for this beam. The **species** keyword is used to provide 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. The **cycle interval** parameter specifies the emission frequency in timesteps, with a default value of 1. **Count** specifies the number of particles emitted from each face in the emission per timestep, with a default value of 1. The **emit probability** keyword 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.

The **energy distribution** keyword 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 three additional options; see below.

The **angle distribution** keyword 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 °) 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., $dN/d\theta = \cos\theta$) 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.

The **normal tolerance** keyword 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 $\left|\hat{n}_1 - \hat{n}_2\right| \leq \varepsilon$, they are considered to be equal. If not specified, it 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.

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:

qpmin(t) = qpmin_floor, or

qpmin(t) = max{*scale*fun(t), qpmin_floor*}

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 *num* [SCALE *scale*]" 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*, *Qcre*, is a running sum of *I(t)*A*dt_emit/segment_area*, where *I(t)* is the time function defined by the **temporal** keyword, , and *dt_emit = cycle_interval*timestep*. Once this sum exceeds *qpmin(t)*, *N* particles are created, where *N = min(int(Qcre/qpmin),count)* and *Qcre* 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 *Qcre-min*, *Qcre-mean*, and *Qcre-max* for the faces. The actual runtime values of these quantities are scaled by *I(t)*cycle_interval*.

Options specific to the **beam emission** command:

The **spatial distribution** keyword 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.

The **beam emission** command keyword has two keyword options that provide a means to specify the amplitude of the beam current. The **amplitude** keyword provides a scalar multiplier for the beam amplitude,

in amperes, with a default value of 1.0. The **temporal** keyword provides a means to specify non-constant time dependence for the beam amplitude. If specified, the amplitude at any instant in time is the product of the scalar multiplier provided by the **amplitude** keyword and the value of the function-set specified by **temporal** 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 three additional options for the **energy distribution**: MAXWELLIAN, ITS, and ITS_SURFACE. 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

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

where *kT* is the supplied temperature of the distribution.

The two ITS options indicate that the energies for emitted particles are to be obtained by randomly sampling from data computed by the ITS Radiation Transport code (or any other source that follows the ITS/Emphasis convention for writing energy-angle (E,θ,ϕ) emission distributions to PFF datasets). These data are transferred from ITS to Emphasis via PFF datasets. The datasets are read from the PFF file whose name is specified using the **its file** command keyword.The usage of the ITS and ITS_SURFACE keywords depends on the type of PFF file being used. The original "version 0" PFF file only contained emission distribution datasets for ITS emission subsurfaces. More recently, a new "version 1" PFF file has been developed, which organizes data by the ITS emission surfaces defined with the "ELECTRON-EMISSION" command in the ITS simulation. For each surface, there is a header dataset with geometry information of the surface, followed by one or more emission datasets for each subsurface.

For *version-0* PFF files, the ITS keyword must be used. There are two types of *version-0* PFF files: archaic files (earlier than ~2008) that have no geometry information, and later files in which each dataset includes a bounding box for the spatial domain on which the distribution was collected. For archaic files, the user must explicitly provide the dataset

number for the distribution. For later *version-0* files with the bounding box information, the integer is optional. The dataset index is computed internally by finding the best match to the bounding box of the sideset. If it cannot find a match, an error will be generated, and the dataset index will have to be explicitly provided.

For *version-1* PFF files, two syntax options are supported. First, with the new ITS_SURFACE keyword, the integer parameter is now required. If this value is greater than zero, it is the index of a single ITS surface in the PFF file. All faces in the sideset must be on the specified emission surface. Internally, the subsurface in which each face is located is determined automatically, and the corresponding emission distribution dataset is associated with the face. More recently, the ability to directly auto-fit sidesets to one or more emission surfaces has been added. This is enabled with either "ITS_SURFACE 0" or simply ITS. There is no ambiguity with the latter option – the type of PFF file makes the context clear.

In an ITS emission dataset, the angles θ and ϕ are computed in a local, right-handed basis, ($\hat{n}, \hat{\phi}_0, \hat{n} \times \hat{\phi}_0$), where $\hat{n}$ is the normal to the surface, $\hat{\phi}_0$ defines the ϕ = 0 axis, and $\hat{n} \times \hat{\phi}_0$ defines the ϕ = 90^o axis. Emphasis needs to use the same local basis for each face. Information to compute $\hat{\phi}_0$ is encoded in the ITS dataset, except for archaic *version*-0 PFF files. For these files, the user must use the **phi0** keyword to explicitly provide the X, Y, and Z components of vector not parallel to $\hat{n}$ to compute $\hat{\phi}_0$.

Finally, both ITS options have 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 [0,2π].

Options specific to the **field emission** command:

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.

The **height distribution** keyword is used to specify 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.

The **breakdown** keyword is used to specify the electric field intensity (**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.

courant factor

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

Courant factor *real*, [**relative type** = MINIMUM EDGE LENGTH | AVERAGE EDGE LENGTH | MINIMUM ELEMENT HEIGHT]

The specific scale length used can be selected using the **relative type** keyword. Currently, this length scale can be the minimum or average length of any edge in the problem domain, or 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, the minimum length of any edge in the problem domain will be used.

If the **constant time step** keyword (see [4]) 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 timestep will be used (not recommended practice). It is important to note that the timestep due to the electromagnetic field solver formulation being used should not be exceeded regardless of the method chosen for setting the timestep.

electron surface database

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.

electron surface database, **type**=*string*, **name**=*string*, **file=***string* [, **logarithmic**]

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

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 above. 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:

qpmin(t) = max{*scale*fun(t),qpmin_floor*}

By default, qpmin(t) = 0, and *qpmin_floor* = 0 if only the **qpmin_fun** keyword is specified. This keyword uses the standard Nevada function syntax "FUNCTION *int* [SCALE *scale*]" 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

The **particle balance** command keyword controls the dynamic load balancing of the particle workload in parallel

particle balance [, **trigger=***real*] [, **target**=*real*] [, **min_tol_incr**=*real*] [, **min_part_per_cell=***real*] [, **max_step_count**=*int*] [, **zoltan**=*parameter_name "parameter_value"*]

The **trigger** keyword controls when load balancing is attempted by comparing the specified value to the current imbalance. The **target** keyword specifies the imbalance that the load-balancing algorithm attempts to achieve. A perfectly balanced particle workload has an imbalance of 1.0. The **min_tol_incr** keyword specifies the fractional amount above the imbalance obtained at the last balance before rebalancing is attempted. The **min_part_per_cell** keyword is the minimum average number of particles per cell required before load balancing is attempted. The **max_step_count** keyword specifies the maximum number of time steps after balancing occurs before rebalancing will be attempted again. The **zoltan** keyword 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

The **particle merge** command keyword controls the merging of particles.

particle merge, [species=name1 [… name_n]], [exclude=name1 [… name_n]], **cycle interval=***int*, **trigger**=*int* [*int*], [, **target**=*int*]

[, **block=*int*** [*int* …]] [, **subgrids**=*int*] [, **max_iterations**=*int*]
[, **vth_frac**=*real*] [, **qtot_frac**=*real*] [, **qmax_over_qmin**=*real*]
[, **spread**=*real*] [, **qerr_tol**=*real*] [, **qlow_tol**=*real*] [, **emax_tol**=*real*]

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. The **target** keyword specifies the target number of particles to remain in the cell after a successful merge; its default value is 50% of the **trigger** value. 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. The **subgrids** keyword (default value: 3) provides a scale factor for sub-gridding each element. The number of actual sub-elements used will be the cube of this value. The **max_iterations** keyword (default value: 1) specifies the number of iterations used in testing for particle rejection due to thermal velocities much larger than the mean thermal velocity. The **vth_frac** keyword (default value: 8.0) specifies the multiple of the thermal velocity above which a particle will be rejected due to large thermal velocity. The **qtot_frac** keyword (default value: 5.0) is used to specify the rejection of heavyweight particles. Any particle whose charge is more than this value times the mean particle charge will be rejected. The **qmax_over_qmin** keyword (default value: 2.0) specifies the nominal ratio of the maximum and minimum charge of any particle in an element after merging has been performed. The **spread** keyword (default value: 1.0) is a scale factor determining the size of the volume into which merged particles associated with a single sub-grid region will be positioned. The **qerr_tol** keyword (default value: 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. The **qlow_tol** keyword (default value: 0.2) controls merge rejection due to low-weight merged particles. The merge will be accepted only if minimum charge of any merged particle is greater than this value times the mean charge of all merged particles in the element. The **emax_tol** keyword (default value: 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.

particle preload

This command preloads the specified **blocks** with the specified particle distribution. 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. The **count** keyword is specified by 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 the **count** keyword is not supplied, it defaults to "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 specified by the **count** keyword. The particle **density** is specified in units of m^{-3}. The **momentum** keyword specifies the mass normalized momentum of preloaded particles in units of m/s. The optional **temperature** keyword specifies the mean temperature of the Maxwellian distribution in the rest frame in units of eV.

particle preload, **block** *int* [*int* …] **species**=*string*
[**count** *int int int*] **density** *real* **momentum x** *real* **y** *real* **z** *real*
[**temperature** *real*]

particle sort

The **particle sort** command keyword controls the sorting of particles

particle sort [, **cycle interval=***int*] [, **cell**] [, **species**]
[, **block=*int*** [*int* …]]

The **cycle interval** keyword controls the frequency of sorting. The default is to sort every cycle. The remaining keywords control how the particles are sorted. By default particles are sorted by cell which can also be specified explicitly with the **cell** keyword. Particles within a cell may also be sorted by species by specifying the **species** keyword. By default, particles in all element blocks will be sorted. However, if one or more element blocks are specified with the **block** keyword, only particles in elements of the specified element blocks will be sorted.

Framework Keywords

These keywords are framework keywords [4] which are required for a successful UTDEM simulation and appear outside the "unstructured td electromagnetics" block.

Edgeset keyword

exodus edge sets

Define specific exodus sideset(s) existing in the model which have been pre-coded by the user to be converted to edgeset(s) by the Nevada framework. This is necessary because standard ExodusII has no notion of a list of edges.

exodus edge sets (*int, int, …***)**

Specifies a list of sideset ids existing in the Genesis file that are to be converted to Nevada edgesets. 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.

Material keywords

material

Define a material model or models for a material

material *int*
    model *int*
    model *int*
    …
end

Relates the material **material** to material **model**(s). For UTDEM, only one model applies to each material.

model

Define a material model

model *int string*
    [parameter real]
    [parameter real]
    …
end

Relates the material model **model** to a specific model type defined by the name *string* and defines parameters specific to that model.

For UTDEM, options for *string* and *parameter*(s) are:

Simple Electrical
eps *real*
mu *real*
sigma *real*
end

or

Breakdown Electrical
eps *real*
mu *real*
sigma *real*
threshold *real*
sigma_bkdn *real*
end

or

HP Gas Electrical
eps *real*
mu *real*
sigma0 *real*
density *real*
water_fraction *real*
gasname "*string***"**
xiev *real*
ndof *int*
ntimestates 3
end

or

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

or

Kinetic Gas Electrical

eps *real*
mu *real*
sigma0 *real*
p_torr *real*
mixture "*string"*
no_attachment
fractional ionization *fractional_ionization_specification*
primary *string*
secondary *string*
emin *real*
emax *real*
nebins *int*

end

or

RIC Electrical

eps *real*
mu *real*
sigma0 *real*
coefficient *real*
exponent *real*
ndof *int*
ntimestates 3

end

where **eps** is the relative permittivity of the medium, **mu** is the relative permeability of the medium, and **sigma** (or **sigma0)** is the conductivity of the material.

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 "HP Gas Electrical" model is a high-pressure gas chemistry model containing the additional parameters **density**, **water_fraction**, **xiev**, and **ndof**. 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 making the model variables 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 by "itt_air".

The “HP Foam Electrical” model is an *experimental* foam model based on the field-exclusion foam model of Stringer and Dumcum [8]. 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.

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 be 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 above 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, $|q_{sec}|$, is less than q_{min}, an ionization fraction F_{ion} is computed using

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

subject to the additional constraint that

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

The probability of the collision is then reduced by F_{ion}, and if the collision still occurs, q_{sec} is increased by $1/F_{ion}$. There are two forms available for *fractional_ionization_specification*:

function *int*, [**scale**=*real*,] **baseline**=*real*
or
real, [**scale**=*real*,] **baseline**=*real*.

The **function** parameter, combined with the optional multiplicative **scale** parameter provides the time-dependent specification of q_{min} (in Coulombs). If not provided, the **scale** parameter is set to 1.0. In the second form of the specification, the function can be replaced by a single real value, which is interpreted to the amplitude of a constant function, and the minimum charge in this case is simply the product of that value and the **scale** parameter. The optional **baseline** parameter is used to specify the value of F_{base}, and defaults to 0.0. Finally, if the **fractional ionization** parameter 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 above for how this file is located. 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.

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 Farad(F)/m, K (**coefficient**) is in ((Mho/F)/(Rad/s), $\dot{\gamma}$ is the dose rate in Rad/s, and *e* is the **exponent** parameter. Typical values for kapton are $\varepsilon = 3.5\varepsilon_0$, $K = 3.23 \times 10^{-6}$, and $e = 0.95$.

The **ntimestates** parameter in the HP Gas and RIC models *must* be set to 3 for proper UTDEM functionality. If not, the code will halt and notify the user.

Example input file fragments for these model specifications are shown in Fig. 3.

```
Model 1 Simple Electrical
  eps 2.
  mu 1.
  sigma 1.e-3
end

Model 2 Breakdown Electrical
  eps 2.
  mu 1.
  sigma 0.
  threshold 2000.0
  sigma_bkdn 1.e7
end

Model 3 HP Gas Electrical
  eps 2.
  mu 1.
  sigma0 0.
  density 1.23
  water_fraction 0.02
  ndof 1
  ntimestates 3
end

Model 4 RIC Electrical
  eps 2.
  mu 1.
  sigma0 0.
  coefficient 3.23e-6
  exponent 0.95
  ntimestates 3
end
```

Figure 3. Typical material model descriptions.

Simulation time and output control keywords

Typical simulation time and output control keywords are shown in Fig. 4. These are described briefly below.

```
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 4. Typical simulation and output control keywords

termination time = *real*
Total time for which to run the simulation

termination cycle = *int*
Total cycles for which to run the simulation

emit screen, cycle interval = *int*
Print status line to standard out every **cycle interval** cycles

emit screen, time interval = *float*
Print status line to standard out every **time interval**

emit plot, cycle interval = *int*
Write **plot variable**s to exodus file every **cycle interval** cycles

emit plot, time interval = *float*
Write **plot variable**s to exodus file every **time interval**

emit hisplot, cycle interval = *int*
Write global variables to hisplt file every **cycle interval** cycles

emit hisplot, time interval = *float*
Write global variables to hisplt file every **time interval**

plot variable
registered-variable name
registered-variable name
…
end

history plot variable

registered-variable name
registered-variable name
…

end

Valid plot variables for UTDEM are **electric_field**, **magnetic_field**, **magnetic_flux_density**, **cur_den** (current density for external J-sources) and **econ** (conductivity, from Simple Electrical, RIC Electrical, and Breakdown Electrical material models, and for external J-sources), and **electron_concentration**, **negative_ion_concentration**, **avalanche_rate**, **attachment_rate** (from HP Gas material model).

UTDEM_PIC adds the variables **pic_current, ave_pic_current, ave_rho_*string***, **rho_*string*** (where ***string*** is the name of a defined particle species), **ave_electric_field, ave_magnetic_field, charge_density**, and **divd_m_rho** ($\nabla \cdot \overline{D} - \rho$). The ave variables are the quantities averaged over cycle intervals. An individual particle species' charge density is separately computed only if it is referenced using the **rho_*string*** or ave_rho_*string* plot variable, or explicitly defined with the **register_density** option of the **define species** input command (see on page 37). If the **ave_rho**_*string* plot variable is listed then EMPHASIS calculates the **rho**_*string* plot variable, but the non averaged charge density is only output if the **rho**_*string* variable is added to 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. The **pic_current** plot variable is the particle current applied to the mesh. Note that multiple PMC symmetry planes must be defined by separate side sets for the **pic_current** variable to be accurate.

Valid history plot variables are the same, and for those selected, time-history data for all spatial locations specified in the **tracer points** keyword will be written to the hisplt database *problem_name*.**his.**

Restart keywords

Restart keywords are described briefly below.

emit restart, cycle interval = *int*
Emit a restart dump every *cycle interval* cycles

emit restart, time interval = *real*
Emit a restart dump every *time interval* seconds

restart dumps *int*
Retain the last *restart dumps* dumps, default 2

A restart dump is also created at the end of the simulation. By default only the last two restart dumps generated are retained, changed by the **restart dumps** keyword.

read restart dump *int*
Restart simulation by reading *restart dump*

read restart time *real*
Restart simulation

For the first form the actual *restart dump* is read if it exists, otherwise the restart fails. If *restart dump* is -1, the latest available restart dump is used by consulting the generated file *problem_name*.dpl. If this file is empty or does not exist, a new simulation is started. For the second form the closest restart dump to time *restart dump* is selected. If *restart time* is negative a new simulation is started.

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.

Linear solver keywords

Typical linear solver keywords, in this case for AZTEC [9], are shown in Fig. 5. In the first case, conjugate gradient (cg) is specified 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.

For the second case, cg is again specified but with jacobi preconditioning 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.

In the third case, the Multi-Level (ML) preconditioner is specified to achieve convergence for very large time steps. The settings shown are typical to achieve a successful ML solution with limited testing. Others are possible and perhaps even desirable.

The fourth case mimics as closely as possible using Aztec the solver technology used in the legacy unstructured code.

Debug Mode Keyword

debug mode

Specifies specific code debug information at run time.

debug mode: *string*

Some options are “location”, “signals”, “fpe”.

Note that “signals” or “fpe” must be specified if events such as floating-point exceptions are to be caught and reported.

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

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

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
```

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

Figure 5. Typical solver control keywords.

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

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.

Running UTDEM Simulations

As mentioned previously, running a UTDEM simulation requires a Genesis mesh geometry file, *problem_name*.gen, and an input file, *problem_name*.inp (example in Appendix II). Assuming a standard Nevada user's environment or a subset thereof exists in the user's operating environment, two scripts are used to run simulations:

alegrabal –p*int problem_name*

and

Alegra *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.

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 [3]. 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 III. 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 IV. 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 [10].

4. Invoke **UTDEM** physics in Emphasis (3D) to complete the full 3D solution, specifying *field dist* in the port source keyword descriptor.

Conclusion

This document, along with the user guides for a modeling and meshing tool such as I-DEAS should allow the user to successfully utilize EMPHASIS/Nevada UTDEM to

simulate a wide variety of EM effects. To gain experience, the UTDEM regression suite contains several simulations exercising most if not all of the code options. These should be examined and well understood before attempting more involved simulations.

References

1. C. D. Turner, T. D. Pointon, K. L. Cartwright, *Emphasis/NEVADA Unstructured FEM Implementation Version 2.1.1,* Sandia National Laboratories Report SAND2014-16737, Aug. 2014.

2. L. A. Schoof, V. R. Yarberry, *EXODUSII: A Finite Element Data Model,* Sandia National Laboratories Report SAND92-2137, reprinted Sep. 1996.

3. M. F. Pasik, et.al., *VOLMAX User's Guide,* unpublished.

4. S. K. Carroll, et. al., *ALEGRA: Version 4.6,* Sandia National Laboratories Report SAND2004-6541, Jan. 2005.

5. A. Friedman, *A Second-Order Implicit Particle Mover with Adjustable Damping,* J. Comput. Phys **90**, 292-312 (1990).

6. A. D. Greenwood, K. L. Cartwright, J.W Luginsland, E. A. Baca, *On the elimination of numerical Cerenkov radiation in PIC simulations*, J. Comput. Phys 201, 665-684 (2004).

7. R. S. Coats, M. F. Pasik, and D. B. Seidel, *Emphasis/Nevada STDEM User's Guide Version 1.0,* Sandia National Laboratories Report SAND2005-1024, April 2005.

8. T. A. Stringer and N. S. Dumcum, *Comprehensive Report on Gas/Foam RIC Test and Analyses,* ITT Industries Report, Sep. 2002.

9. R. S. Tuminaro, M. Heroux, S. A. Hutchinson, J. N. Shadid, *Official Aztec User's Guide Version 2.1,* Sandia National Laboratories Report SAND99-8801J, Nov. 1999.

10. C. D. Turner, W. J. Bohnhoff, J. L. Powell, *EMPHASIS™/Nevada CABANA User Guide Version 2.1.1*, Sandia National Laboratories Report SAND2014-16736, Aug. 2014.

Appendix I. 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 [3] 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.

Appendix II. 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

aztec 2
  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
```

Appendix III. Sideset Extractor input file

```
$-----------------------------------BEGIN_QA---------------------------
$ CVS: $Id$
$-----------------------------------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
```

Appendix IV. Poisson Solution input file

```
$-----------------------------------BEGIN_QA---------------------------
$ CVS: $Id$
$-----------------------------------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
```

Distribution

1	MS1152	K. L. Cartwright, 01352
1	MS1152	R. S. Coats, 01352
1	MS1152	J. D. Kotulski, 01352
1	MS1152	T. D. Pointon, 01352
1	MS1152	C. D. Turner, 01352
1	MS0899	Technical Library, 9536 (electronic copy)


Sandia National Laboratories