User's Guide for spiffe Version 4.5

Michael Borland
Advanced Photon Source

September 2, 2015

1 Introduction

This manual describes use of the program spiffe (SPace charge and Integration of Forces For Electrons). Those wishing to understand the algorithms used by spiffe should consult reference [2].

2 Run Organization

A typical spiffe run consists of the following steps, most of which are optional. Although the steps need not be executed exactly in the order shown here, doing so will prevent the occasional error message. Each step is performed by invoking a namelist command, as indicated:

1. Define the simulation region and the geometry of a cavity. This includes defining the grid sizes, the boundary conditions, which types of fields to include (TE or TM), and how to interpolate on the grid. This is done with the define_geometry command, which requires existence of a boundary definition file in a POISSON-like format. This file specifies not only the location of the metal surfaces in the problem, but optionally the potential of each.

2. Optional: define the time-dependent fields and/or a method of generating fields. This is done with the load_fields, define_antenna, and/or add_on_axis_fields commands. The former allows loading fields from a previous spiffe run, whereas the latter allows generating new fields using a modulated sine-wave current source.

3. Optional: define solenoid coils, from which static magnetic fields will be computed and imposed on the particle motion. The define_solenoid command permits defining a solenoid with a specified length, radius, symmetry, and current/field.

4. Optional: define constant field components to be imposed on the simulation. These field components, defined with the constant_fields command, are not necessarily physical in that they may not satisfy Maxwell's equations.

5. Optional: load a particle distribution and/or define a cathode for generation of particles. The former is not presently implemented in version 2.0 of spiffe. The define_cathode namelist permits specification of the size, current density, time profile, and other parameters of particle emission from an annulus. Starting in version 2.3, multiple cathodes may be defined.

6. Optional: invoke and control Poisson correction. This is controlled by the `poisson_correction` command. It is recommended in order to increase the accuracy of the field solutions. Without it, numerical errors tend to build up that correspond to fictitious concentrations of static charge on the grid.

7. Optional: define resistive elements in the cavity interior, which simulates the effect of walls that are less than perfectly conducting. This is done with one or more `define_resistor` commands. It slows the simulation tremendously, due to the reduced step size needed to obtain numerical stability.

8. Optional: define a series of diagnostic "screens" for placement in the beam path. A single `define_screen` command can be used to produce output of beam parameters at a series of equispaced longitudinal positions.

9. Optional: request beam snapshots at equispaced time intervals. This is done with the `define_snapshots` command.

10. Optional: request field output and/or field saving. Field output is in a format convenient for plotting and analysis, in that values are interpolated onto the same grid for all field components. Field saves are intended for use by a subsequent spiffe run. These operations are set up by the `define_field_output` and `define_field_saving` commands.

11. Optional: request sampling of the fields vs time or space coordinates. With the command `define_field_sampling`, one may set up sampling of a specific field component along a line in z or r, with output vs the distance along the line at specified times or else output of the average value along the line vs time.

12. Optional: request reduction and translation of the simulation grid so that it moves along with the particles.

13. Optional: request simulation of secondary emission, using the `define_secondary_emission` command. This won't result in generation of any secondary particles unless a cathode is defined or particles are loaded from a file.

14. Define integration parameters and begin integration. The `integrate` command allows specifying the integration time step, the total time to integrate, and other conditions of integration. (Strictly speaking, it is optional, but only if you don't want to do anything.)

3 Manual Pages

The main input file for a `spiffe` run consists of a series of namelists, which function as commands. Most of the namelists direct `spiffe` to set up to run in a certain way. A few are "action" commands that begin the actual simulation. FORTRAN programmers should note that, unlike FORTRAN namelists, these namelists need not come in a predefined order; `spiffe` is able to detect which namelist is next in the file and process appropriately.

Each namelist has a number of variables associated with it, which are used to control details of the run. These variables come in three data types: (1) `long`, for the C long integer type, (2) `double`, for the C double-precision floating point type, and (3) `STRING`, for a character string enclosed in double quotation marks. All variables have default values, which are listed on the following pages. `STRING` variables often have a default value listed as `NULL`, which means no data; this is quite different from the value “”, which is a zero-length character string. `long` variables are often used as logical flags, with a zero value indicating false and a non-zero value indicating true.

On the following pages the reader will find individual descriptions of each of the namelist commands and their variables. Each description contains a sequence of the form

```
&<namelist-name>
    <variable-type> <variable-name> = <default-value>;
    .
    .
    .
&end
```

This summarizes the parameters of the namelist. Note, however, that the namelists are invoked in the form

```
&<namelist-name>
    [<variable-name> = <value> ,]
    [<array-name>[<index>] = <value> [,<value> ...] ,]
        .
        .
        .
&end
```

The square-brackets enclose an optional component. Not all namelists require variables to be given–the defaults may be sufficient. However, if a variable name is given, it must have a value. Values for `STRING` variables must be enclosed in double quotation marks. Values for `double` variables may be in floating-point, exponential, or integer format (exponential format uses the ‘e’ character to introduce the exponent).

3.1 define_geometry

- **description:** Define the simulation region and the geometry of a cavity. This includes defining the grid sizes, the boundary conditions, which types of fields to include (TE or TM), and how to interpolate on the grid. This is done with the `define_geometry` command, which requires existence of a boundary definition file in a POISSON-like format. This file specifies not only the location of the metal surfaces in the problem, but optionally the potential of each.

- **example:**

```
&define_geometry
    nz = 165, zmin = 0.0, zmax = 0.0959148653920,
    nr = 165, rmax=0.04289777777,
    boundary = "mg6mm-3.geo",
    boundary_output = "mg6mm-3.bnd",
    interior_points = "mg6mm-3.pts"
&end
```

  This command defines a problem with an extent of about 9.6cm in the longitudinal direction and about 4.3cm in the radial direction. The number of grid lines in each dimension is 165. Boundary input is taken from file `mg6mm-3.geo`. In addition, output of the boundary coordinates is to be placed in file `mg6mm-3.bnd` while output of the coordinates of all interior grid points is to be placed in `mg6mm-3.pts`.

- **synopsis and defaults:**

```
&define_geometry
    long nz = 0;
    long nr = 0;
    double zmin = 0;
    double zmax = 0;
    double rmax = 0;
    double zr_factor = 1;
    STRING rootname = NULL;
    STRING boundary = NULL;
    STRING boundary_output = NULL;
    STRING urmel_boundary_output = NULL;
    STRING discrete_boundary_output = NULL;
    STRING interior_points = NULL;
    STRING lower = "Dirichlet";
    STRING upper = "Neumann";
    STRING right = "Neumann";
    STRING left  = "Neumann";
    long include_TE_fields = 0;
    long exclude_TM_fields = 0;
    long turn_off_Er = 0;
    long turn_off_Ez = 0;
    long turn_off_Ephi = 0;
```

```
    long turn_off_Br = 0;
    long turn_off_Bz = 0;
    long turn_off_Bphi = 0;
    long print_grids = 0;
    long radial_interpolation = 1;
    long longitudinal_interpolation = 1;
    long radial_smearing = 0;
    long longitudinal_smearing = 0;
&end
```

- **details:**
  - `nz, nr`: number of grid lines in z and r dimensions, respectively.
  - `zmin, zmax`: starting and ending longitudinal coordinate, respectively.
  - `rmax`: maximum radial coordinate.
  - `zr_factor`: a factor by which to multiply the z and r values in the boundary input file to convert the values to meters. For example, `zr_factor=0.01` would be used if the boundary input file values were in centimeters.
  - `rootname`: rootname for construction of output filenames. Defaults to the rootname of the input file.
  - `boundary`: name of input file containing POISSON-like specification of the cavity boundary and surface potentials.
  - `boundary_output`: (incomplete) name of output file to which SDDS-protocol data will be sent containing the coordinates of points on the *ideal* boundary, i.e., the boundary you would get if your grid spacing was zero. Recommended value: “
  - `discrete_boundary_output`: (incomplete) name of output file to which SDDS-protocol data will be sent containing the coordinates of points on the actual boundary used in the simulation. This differs from the ideal boundary because every point on the actual boundary must be a grid point. Recommended value: “
  - `interior_points`: (incomplete) name of output file to which SDDS-protocol data will be sent containing the coordinates of all interior points of the cavity. May be used together with boundary output files and `sddsplot` to manually confirm the interpretation of the cavity specification by `spiffe`.
  - `lower, upper, right, left`: boundary conditions for the edges of the simulation region. The defaults are listed above. “Dirichlet” boundary conditions means that electric field lines are parallel to the boundary. “Neumann” boundary conditions means that electric field lines are perpendicular to the boundary.
  - `include_TE_fields`: flag indicating whether to include transverse-electric fields, i.e., fields having no longitudinal electric field components. If you include space-charge and the beam is rotating, you should set this to 1.
  - `exclude_TM_fields`: flag indicating whether to exclude transverse-magnetic fields, i.e., fields having no longitudinal magnetic field components.
  - `turn_off_...`: flags indicating that the specified fields should be “turned off,” which means that the don’t affect particles. Used for testing purposes.
  - `print_grids`: flag requesting a text-based picture of the simulation grids.

– `radial_interpolation, longitudinal_interpolation`: flags requesting that field components be interpolated in the radial and longitudinal direction when fields are applied to particles. If 0, then fields will change abruptly as particles move from one grid square to the next.

– `radial_smearing, longitudinal_smearing`: flags requesting that charge and current from simulation macro particles be smeared over the grid points surrounding each particle. If 0, charge and current are assigned to the nearest grid point.

3.2 Geometry File

- **description:** This page describes the structure of the geometry file used with the `define_geometry` namelist. The file is similar to those used with the program POISSON.
- **example:**

```
! new RF gun, first cell
 &po x=0.000, y=0.000 &end
 &po x=0.000, y=0.006 &end
 &po x=0.0016, y=0.0135 &end
 &po x=0.0016, y=0.0427755  &end
 &po x=0.0060, y=0.0427755   &end
 &po nt=2, x0=0.006, y0=0.0181355,
          r=0.02464, theta=0.0  &end
 &po x=0.03064  , y=0.014818    &end
 &po nt=2, x0=0.02861, y0=0.014818,
          r=0.00203, theta=-90.0  &end
 &po x=0.02836, y=0.012788   &end
 &po nt=2, x0=0.02836, y0=0.010588,
          r=0.0022, theta=180.0  &end
 &po x=0.02616, y=0.01008     &end
 &po nt=2, x0=0.03116, y0=0.01008,
          r=0.005, theta=270.0  &end
 &po x=0.0348, y=0.00508  &end
 &po x=0.0348, y=0.00  &end
 &po x=0.000000, y=0.00  &end
```

- **synopsis and defaults:**

```
&point
    int nt = 1;
    double x = 0;
    double y = 0;
    double x0 = 0;
    double y0 = 0;
    double r = 0;
    double a = 0;
    double b = 0;
    double aovrb = 0;
    double theta = 0;
    int change_direction = 0;
    double potential = 0;
    int ramp_potential = 0;
&end
```

- **details:**
  - `nt`: The segment type, where 1 (the default) indicates a line segment; 2 indicates an arc of a circle; 3 indicates the start of a separate structure; and 4 indicates a definition of an in-vacuum point.

- `x, y`: For `nt=1`, the endpoint of the line. x corresponds to z (the longitudinal coordinate) and y corresponds to r (the radial coordinate). For `nt=2`, the endpoint of the circular or elliptical arc relative to (`x0`, `y0`). For `nt=3`, the first point on the new shape. For `nt=4`, the coordinates of the in-vacuum point.
- `x0, y0`: For `nt=2`, the center of the circular or elliptical arc.
- `r`: For `nt=2`, the radius of the circular arc. The equation of the arc is $(x-x_0)^2+(y-y_0)^2 = r^2$.
- `a`, `b`, `aovrb`: For `nt=2`, the parameters of the elliptical arc. The equation of the arc is $(x-x_0)^2/a^2+(y-y_0)^2/b^2 = 1$. If `a` or `b` is zero, then the value is determined using the ratio `aovrb` (a over b).
- `theta`: For `nt=2`, the angle in degrees of the end of the arc as seen from the center of the arc. If this angle is less (greater) than the angle of the starting point (which is on $[-180, 180]$), then the sense of the arc is clockwise (counter-clockwise).
- `change_direction`: For `nt=2` when `theta` is not given but `x` and `y` are, `spiffe` may have trouble determining the direction of the arc. This flag can be used to change the direction.
- `potential`: The potential of the segment, in volts.
- `ramp_potential`: If non-zero, the potential along a line is ramped linearly between the given value and the end value for the previous segment. Only implemented for `nt=1`.

3.3 define_antenna

- **description:** Allows generating time-varying fields using a modulated sine-wave current source.
- **example:**

```
&define_antenna
    start = 0.01, end = 0.02, position = 0.03,
    direction = "z",
    current = 1,
    frequency = 2856e6,
    waveform = "spline.wf"
&end
```

  This defines a current source in the longitudinal direction extending from z of 1cm to 2cm at a radius of 3cm. The amplitude of the current is 1A with a frequency of 2856MHz, modulated by the envelope in SDDS file `spline.wf`.

- **synopsis and defaults:**

```
&define_antenna
    double start = 0;
    double end = 0;
    double position = 0;
    STRING direction = "z";
    double current = 0;
    double frequency = 0;
    double phase = 0;
    STRING waveform = NULL;
    double time_offset = 0;
&end
```

- **details:**
  - `direction`: may take values "z" and "r", indicating an antenna extending in the longitudinal or radial direction, respectively.
  - `start, end`: starting and ending limits of the antenna in the `direction` direction.
  - `position`: position of the antenna in the "other" direction. I.e., it is the r position if `direction` is "z", and the z position if `direction` is "r".
  - `current, frequency, phase`: basic parameters of the antenna waveform.
  - `waveform, time_offset`: specifies an envelope function for the antenna drive. The SDDS file `waveform` must contain at least two columns, named `t` (for the time in seconds) and `W`, specifying the envelope `W(t)`. The antenna is driven by the function $I * W(t - t_o) * \sin(2 * \pi * f + \phi)$, where I is the `current` in Amperes, t is the time in seconds, t_o is `time_offset` in seconds, f is `frequency` in Hertz, and ϕ is `phase` in radians.

3.4 load_fields

- **description:**

  Allows loading fields from a previous `spiffe` run. The fields are stored in a file created with the `save_fields` command.

- **example:**

```
&load_fields
    filename = "fields.saved",
    factor = 1.5;
&end
```

  This loads fields from the file `fields.saved`, applying a factor of 1.5 to the values. These fields become the only time-varying fields in the problem. (Others may be superimposed in subsequent operations.)

- **synopsis and defaults:**

```
#namelist load_fields
    STRING filename = NULL;
    double Ez_peak = 0;
    double factor = 1;
    double time_threshold = 0;
    long overlay = 0;
#end
```

- **details:**

  - `filename`: Name of the file from which to take field data. Normally created with the `save_fields` command. The file is in SDDS-protocol, typically with multiple data pages. Normally, the first page is used. This may be modified with the `time_threshold` parameter.
  - `Ez_peak`: Desired maximum value of on-axis longitudinal electric field. The fields from the data file are scaled to obtain this value. Note that this option cannot be used if TM fields are disabled. By default, no scaling occurs.
  - `factor`: Factor by which to multiply the fields before use. If `Ez_peak` is 0, then this value is ignored. One of `factor` or `Ez_peak` must be nonzero.
  - `time_threshold`: Minimum simulation time, in seconds, at which the fields may have been created in order to be used. For example, if `time_threshold` is `1e-9`, then `spiffe` will advance through the pages of fields until it finds one from 1ps or more (in simulation time) after the start of the simulation that created the file `filename`.
  - `overlay`: Normally, the time-varying fields in the simulation are set equal to those in the file, within a scale factor. If you wish to simply add the new fields to those already in force, then set `overlay` to a nonzero value.

3.5 set_constant_fields

- **description:** Defines constant field components to be imposed on the simulation. These field components are not necessarily physical in that they may not satisfy Maxwell's equations.

- **example:**

```
&set_constant_fields
    Ez = 1e3,
    Bz = 1,
&end
```

  This command specifies a constant longitudinal electric field of 1 kV/m and a constant longitudinal magnetic field of 1 T. Note that these are physically possible, at least in the absence of metallic or magnetic materials.

- **synopsis and defaults:**

```
&constant_fields
    double Ez = 0;
    double Er = 0;
    double Ephi = 0;
    double Bz = 0;
    double Br = 0;
    double Bphi = 0;
&end
```

- **details:**
  - `Ez`: Specifies longitudinal electric field in V/m.
  - `Er`: Specifies radial electric field in V/m.
  - `Ephi`: Specifies azimuthal electric field in V/m.
  - `Bz`: Specifies longitudinal magnetic field in T.
  - `Br`: Specifies radial magnetic field in T.
  - `Bphi`: Specifies azimuthal magnetic field in T.

3.6 add_on_axis_fields

- **description:**

  Adds on-axis field data to the simulation. The user must provide an SDDS file giving $E_z(z, r = 0)$. This data is used to compute $E_z(z, r, t)$, $E_r(z, r, t)$, and $B_\phi(z, r, t)$ using an off-axis expansion in r and assuming $E \sim \sin(\omega t + \phi)$ and $B \sim \cos(\omega t + \phi)$.

  Any number of `add_on_axis_fields` commands may be given.

- **example:**

```
&add_on_axis_fields
  filename = fieldProfile.sdds,
  z_name = z,
  Ez_name = Ez,
  Ez_peak = 30e6,
  phase = 180,
  expansion_order = 2
  fields_used = fieldUsed.sdds,
&end
```

  This command loads on-axis field data from columns `z` and `Ez` in `fieldProfile.sdds`, and scales it so that the peak field is 30 MV/m. The phase, ϕ, is set to 180 degrees. Because `spiffe` simulates electrons, if $E(z, r)$ is positive, the phase factor must be negative to provide acceleration. I.e., $\phi = 270°$ is the accelerating phase.

- **synopsis and defaults:**

```
#namelist add_on_axis_fields
        STRING filename = NULL;
        STRING z_name = NULL;
        STRING Ez_name = NULL;
        double Ez_peak = 0;
        double frequency = 0;
        double z_offset = 0;
        long expansion_order = 3;
        STRING fields_used = NULL;
#end
```

- **details:**

  - `filename`: Name of the SDDS file containing the data.
  - `z_name`: Name of the column containing z values, which must be monotonically increasing and equispaced.
  - `Ez_name`: Name of the column containing E_z values.
  - `Ez_peak`: Absolute value, in V/m, of the maximum on-axis electric field due to this field profile. The E_z values are scaled to obtain this maximum, but the signs are unchanged.
  - `frequency`: Frequency of the fields, in Hz.
  - `z_offset`: Offset, in meters, to be added to the z values.

- `expansion_order`: Order of the off-axis expansion:
  - 0 E_z is constant in r, while E_r and B_ϕ are zero.
  - 1 E_z is constant in r, while E_r and B_ϕ vary linearly with r.
  - 2 Adds a quadratic variation with r to E_z .
  - 3 Adds a cubic variation with r to E_r and B_ϕ.
- `fields_used`: Name of an SDDS file to which field profile data will be written for all on-axis fields specified up to and including this command.

3.7 define_solenoid

- **description:** Define solenoid coils, from which static magnetic fields will be computed and imposed on the particle motion. The `define_solenoid` command permits defining a solenoid with a specified length, radius, symmetry, and current/field.
- **example:**

```
&define_solenoid
   radius = 0.05,
   z_start = 0.01, z_end = 0.02,
   current = 1,
   Bz_peak = 1,
   turns = 100
&end
```

  This command defines a 100-turn solenoid extending longitudinally between coordinates 1cm and 2cm, at a radius of 5cm. The current, initial 1A, is adjusted to obtain a peak on-axis longitudinal B field of 1Tesla.

- **synopsis and defaults:**

```
#namelist define_solenoid
        double radius = 0;
        double evaluation_radius_limit = 0;
        double z_start = 0;
        double z_end = 0;
        double current = 0;
        double Bz_peak = 0;
        long turns = 1;
        long symmetry = 0;
        STRING field_output = NULL;
        long bucking = 0;
        double z_buck = 0;
#end
```

- **details:**
  - `radius`: The radius of the coils in meters.
  - `evaluation_radius_limit`: The maximum radius in meters at which the solenoidal fields should be computed. This can save considerable CPU time if you are not interested in particles that go beyond a certain radius (e.g., you know they'll be lost).
  - `z_start, z_end`: The starting and ending longitudinal coordinate of the coils, in meters.
  - `current`: The current in each coil in Amperes (**not** Ampere-turns).
  - `Bz_peak`: The peak on-axis longitudinal magnetic field in Tesla desired from this solenoid. The current is scaled to achieve this value. Even if you give this value, you must give an initial value for `current`.
  - `turns`: The number of turns (or coils) in the solenoid.

- symmetry: Either 0, 1, or -1 for no symmetry, even symmetry, or odd symmetry. For codes of ± 1, the resulting fields are those produced by the combination of the specified solenoid and another identical solenoid extending from -z_start to -z_end. If symmetry is 1 (-1), the field from the mirror solenoid adds to (subtracts from) the field due to the specified solenoid.
- field_output: Requests output of the solenoid field to an SDDS file. This output is accumulated field from all solenoids defined up to this point, including the solenoid presently being defined.
- bucking: If nonzero, indicates that this is a bucking solenoid. The solenoid current is adjusted to zero the on-axis value of B_z at z_buck.
- z_buck: If bucking is nonzero, the location at which the field is bucked.

Here is an example of using three define_solenoid commands to make a solenoid field with a peak on-axis B_z value of 0.3T that is bucked to zero at $z = 0$. The main solenoid field is produced by two sets of windings with a ratio of 10:1 between the current:

```
&define_solenoid
        radius = 0.06,
        evaluation_radius_limit = 0.1,
        z_start = 0.0,
        z_end = 0.15,
        current = 1,
        turns = 100,
&end

&define_solenoid
        radius = 0.06,
        evaluation_radius_limit = 0.1,
        z_start = 0.20,
        z_end = 0.30,
        current = 0.1,
        turns = 67,
&end

&define_solenoid
        radius = 0.02,
        evaluation_radius_limit = 0.1,
        z_start = -0.04,
        z_end = -0.02,
        current = 1,  ! any nonzero value will do
        turns = 100,
        bucking = 1,
        Bz_peak = 0.3,
        field_output = solenoid.sdds
&end
```

3.8 define_cathode

- **description:** Permits specification of the size, current density, time profile, and other parameters of particle emission from an annulus. Starting in version 2.3, multiple cathodes may be defined.

- **example:**

```
&define_cathode
    z_position = 0, outer_radius = 0.003,
    current_density = 20e4,
    start_time = 0, stop_time = 200e-12,
    number_per_step = 8,
&end
```

- **synopsis and defaults:**

```
&define_cathode
    double z_position = 0;
    double inner_radius = 0;
    double outer_radius = 0;
    double current_density = 0;
    double temperature = 0;
    double work_function = 0;
    long field_emission = 0;
    double field_emission_beta = 1;
    long determine_temperature = 0;
    double electrons_per_macroparticle = 0;
    double start_time = 0;
    double stop_time = 0;
    long autophase = 0;
    double time_offset = 0;
    double number_per_step = 0;
    double initial_pz = 0;
    double initial_omega = 0;
    double stiffness = 1;
    long discretize_radii = 0;
    long random_number_seed = 987654321;
    long distribution_correction_interval = 0;
    long spread_over_dt = 0;
    long zoned_emission = 1;
    long halton_radix_dt = 0;
    long halton_radix_r = 0;
    STRING profile = NULL;
    STRING profile_time_name = NULL;
    STRING profile_factor_name = NULL;
&end
```

- **details:**

- `z_position`: Longitudinal position of the cathode in meters.
- `inner_radius, outer_radius`: Inner and outer radius of the edges of the cathode, in meters.
- `current_density`: Base current density, in Amperes/m^2.
- `temperature`: Temperature of the cathode in degrees Kelvin. If zero, then emission is constant at the rate given by `current_density`. Otherwise, used together with the work function (given by the `work_function` parameter) and the Richardson-Schottky emission model to determine emission at each time step based on the electric field.
- `work_function`: Work function of the cathode material in eV. Must be nonzero if the temperature is nonzero.
- `determine_temperature`: If nonzero, then attempts to determine the temperature required to give the current density given by `current_density`. You must give the `work_function`. Results are approximate because of the Richardson-Schottky law.
- `field_emission`: If nonzero, then the cathode emits only by field emission. The treatment of field emission is from section 6.13 of *The Handbook of Accelerator Physics and Engineering.*. In field emission mode, `spiffe` splits the cathode into many subcathodes, each one radial grid space in extent. The field emission current density is computed for each subcathode separately, so that the results are correct in the case where the field varies over the cathode.
- `field_emission_beta`: Gives the field enhancement factor for computing field emission current density. The value of the electric field is multiplied by this factor before being used to compute the field emission current density. Typical values are between 1 and 100. In this mode, you must specify `electrons_per_macroparticle`.
- `electrons_per_macroparticle`: Number of electrons represented by each macroparticle.
- `number_per_step`: How many macroparticles to emit per step. Incompatible with specifying `electrons_per_macroparticle`.
- `start_time, stop_time`: Start and stop time for emission, in seconds.
- `autophase`: Flag requesting that cathode emission start only when the field at the cathode has the proper phase to accelerate the beam. The total time for emission is still determined by the difference between the start and stop time.
- `time_offset`: Only relevant when `autophase = 1`. Specifies a time offset relative to the emission start time determined by autophasing.
- `initial_pz`: The initial longitudinal momentum of emitted particles, in dimensionless units (i.e., normalized to $m_e c$).
- `initial_omega`: The initial angular velocity of particles, in radians per second.
- `stiffness`: The beam stiffness, i.e., the particle mass, in electron masses.
- `discretize_radii`: Flag requesting that particles be emitted only from radii $n * \Delta r$, where n is an integer and Δr is the radial grid spacing. This can be useful for certain types of diagnostic runs, but should not be used with space charge.
- `random_number_seed`: The seed for the particle emission random number generator. A large, odd integer is recommended. If 0 is given, the seed is picked based on the computer clock.

- `distribution_correction_interval`: The number of steps between corrections to the emitted particle distribution. Can be used to compensate for nonuniform emission that occurs due to use of random numbers in the emission algorithm. If used, it should be set to 1. Cannot be used when the temperature is nonzero (Richardson-Schottky emission law).
- `spread_over_dt`: Flag requesting that emitted particles have their effective emission times spread out over the simulation time step, Δt. The particle velocities are adjusted appropriately using the instantaneous E_z and E_r fields *only*.
- `zoned_emission`: Flag requesting that emission calculations take place separately for each annular zone of width Δr (the radial grid spacing). Reduces the possibility that random number effects will result in a nonuniform current density.
- `halton_radix_t`: Halton radix (a small prime number) to be used for quiet-start generation of time values.
- `halton_radix_r`: Halton radix (a small prime number) to be used for quiet-start generation of radius values.
- `profile`: The name of an SDDS-protocol file containing a time-profile with which to modulate the base current density.
- `profile_factor_name, profile_time_name`: The columns giving the current-density adjustment factor and the corresponding time when it is valid for the file named by `profile`. The adjustment factor should be on $[0, 1]$.

3.9 load_particles

- **description:**

  Allows loading particles from an SDDS file. This is an alternative to using a cathode and can provide essentially arbitrary particle distributions. Since `spiffe` is a 2.5 dimensional code, the "particles" are really rings at a given radius and longitudinal position.

- **example:**

```
&load_particles
    filename = "particles.sdds",
&end
```

  This loads particles from the file `particles.sdds`.

- **synopsis and defaults:**

```
#namelist load_particles
        STRING filename = NULL;
        long sample_interval = 1;
        double stiffness = 1;
#end
```

- **details:**

  - `filename`: Name of the SDDS file from which to take particle data. The file must have the following columns with the following units:

    z : longitudinal position in meters.

    r : radial poition in meters. If the particle position is initially outside the problem region or inside a metal volume, it will move ballistically until it enters the problem region or emerges from the metal.

    pz : longitudinal momentum, $\beta_z \gamma$.

    pr : radial momentum, $\beta_r \gamma$.

    pphi : azimuthal momentum, $\beta_\phi \gamma$.

    q : charge, in Coulombs.

  - `sample_interval`: Causes `spiffe` to take only every `sample_interval`th particle from the file.
  - `stiffness`: Allows making the beam artificially stiff. Equivalent to increasing the particle mass by the given factor.

3.10 poission_correction

- **description:** This command requests correction of the electric fields so that they satisfy Poisson's equation. It is recommended in order to increase the accuracy of the field solutions. Without it, numerical errors tend to build up that correspond to fictitious concentrations of static charge on the grid.

- **example:**

```
&poisson_correction
    start_time = 1e-9,
    step_interval = 32,
    accuracy = 1e-4,
    error_charge_threshold = 1e-15
&end
```

  This requests that Poisson correction be performed every 32 simulation steps starting 1ns after the start of the simulation. The fractional accuracy of the Poisson solver is 10^{-4}. It is invoked only when the amount of fictitious charge exceeds 0.001 pC.

- **synopsis and defaults:**

```
&poisson_correction
    double start_time = 0;
    long step_interval = 0;
    double accuracy = 1e-6;
    double error_charge_threshold = 0;
    long maximum_iterations = 1000;
    long verbosity = 0;
    double test_charge = 0;
    double z_test_charge = 0;
    double r_test_charge = 0;
    STRING guess_type = "none";
&end
```

- **details:**
  - `start_time`: Time in seconds at which to begin Poisson correction.
  - `step_interval`: Interval between corrections in units of the simulation step.
  - `accuracy`: Fractional accuracy of the Poisson solutions.
  - `error_charge_threshold`: Amount of net error charge that must be present for Poisson correction to actually take place.
  - `maximum_iterations`: Maximum number of iterations of the relaxation loop for the Poisson solver.
  - `verbosity`: Flag requesting informational output about Poisson correction.
  - `test_charge, z_test_charge, r_test_charge`: For developmental use only.
  - `guess_type`: The model to use for the initial guess to start the Poisson iteration. Possibilities are, "none", "line-charge", "point-charge", and "zero." It is recommended that "none" be used.

3.11 translate

- **description:** Sets up repeated translation of the memory used for the mesh in the +z direction to follow the beam. This can be used to simulate a long drift while using less computing time and less memory. There has been little testing of this feature!

- **example:**

```
&translate
        z_trigger = 0.05,
        z_lower = 0.02,
        z_upper = 0.055
&end
```

  This specifies that when the first particle passes $z = 5$cm, the mesh will be contracted to cover the region from $z = 2$cm to $z = 5.5$cm. Thereafter, each time a particle advances by one longitudinal mesh spacing, the grid will be translated that same distance in the +z direction.

  The boundary conditions are changed to Neumann on the upper, right, and left boundaries and Dirichlet on the lower boundary.

- **synopsis and defaults:**

```
&translate
        z_lower = <zMin>,
        z_upper = <zMax>,
        z_trigger = <z0>,
&end
```

  where `<zMin>` and `<zMax>` are the minimum and maximum longitudinal coordinates for the original problem mesh, and `<z0>` is `<zMax>`-$5 * \Delta z$, where Δz is the mesh spacing.

- **details:**

  – `z_trigger`: The first time any particle passes the longitudinal position given for this variable, mesh reduction is triggered and translation is enabled. Prior to this time, the `translate` command has no effect. After this time, translation by Δz occurs every time the maximum longitudinal particle position increases by Δz.

  – `z_lower`: The lower limit of the longitudinal extent of the reduced mesh.

  – `z_upper`: The upper limit of the longitudinal extent of the reduced mesh.

3.12 define_resistor

- **description:** Defines a resistive element in the cavity interior. These simulate the effect of walls that are less than perfectly conducting. It slows the simulation tremendously, due to the reduced step size needed to obtain numerical stability.

- **example:**

```
&define_resistor
    start = 0.02, end = 0.03,
    position = 0.04,
    direction = "z",
    conductivity = 6e3
&end
```

  This specifies a resistor extending in the longitudinal (z) direction from 2cm to 3cm, at a radius of 4cm. The conductivity is $6 * 10^3 (\mathrm{m} * \mathrm{ohms})^{-1}$, the value for copper.

- **synopsis and defaults:**

```
&define_resistor
    double start = 0;
    double end = 0;
    double position = 0;
    STRING direction = "z";
    double conductivity = 1e154;
&end
```

- **details:**
  - `direction`: The direction in which the resistor extends. May be "z" or "r".
  - `start, end`: The starting and ending coordinates of the resistor in the `direction` dimension.
  - `position`: The position of the resistor in the "other" dimension. E.g., the radial position if `direction` is "z".
  - `conductivity`: The conductivity of the metal in $(\mathrm{m} * \mathrm{ohms})^{-1}$.

3.13 define_screen

- **description:** Defines a series of diagnostic "screens" for placement in the beam path. A single `define_screen` command can be used to produce output of beam parameters at a series of equispaced longitudinal positions.

- **example:**

```
&define_screen
    template = "gunRun-%03ld.sdds",
    z_position = 0.01,
    delta_z = 0.01,
    number_of_screens = 5,
    start_time = 1e-9,
    direction = "forward"
&end
```

  Defines a series of 5 screens for detection of forward-moving particles, The screens are positioned at z coordinates of 1, 2, 3, 4, and 5cm, with data going to files named `gunRun-000.sdds`, `gunRun-001.sdds`, etc. Detection begins only after 1ns of simulated time.

- **synopsis and defaults:**

```
&define_screen
    STRING filename = NULL;
    STRING template = NULL;
    double z_position = 0;
    double delta_z = 0;
    long number_of_screens = 1;
    double start_time = 0;
    STRING direction = "forward";
&end
```

- **details:**
  - `filename`: Name of an SDDS file to which to write the data. Used only if `number_of_screens = 1`.
  - `template`: Template for creating the names of SDDS files to which to write data. The template must contain a C-style format specifier for a long integer. The template is used as an argument to the C function `sprintf` to create each filename in turn.
  - `z_position`: The position (or starting position) of the screen (or series of screens), in meters.
  - `delta_z`: In `number_of_screens` is greater than 1, gives the distance in meters between successive screens.
  - `number_of_screens`: The number of screens in the series.
  - `start_time`: The starting time in seconds for accumulation of data.
  - `direction`: The direction in which particles must be traveling in order to be recorded by the screen. May be "forward" or "backward".

3.14 define_secondary_emission

- **description:** Permits specification of secondary emission yield function for the material surface. This refers to emission of one or more new particles when a particle impacts the surface. By default, no secondary emission is done.

- **example:**

```
&define_secondary_emission
    input_file = ``secondary.sdds'',
    kinetic_energy_column = ``K'',
    yield_column = ``Yield''
&end
```

- **synopsis and defaults:**

```
&namelist secondary_emission
        STRING input_file = NULL;
        STRING kinetic_energy_column = NULL;
        STRING yield_column = NULL;
        long yield_limit = 0;
        double emitted_momentum = 0;
        long verbosity = 1;
        long material_id = 1;
        STRING log_file = NULL;
&end
```

- **details:**
  - `input_file` — Name of an SDDS file from which the secondary emission yield curve will be read.
  - `kinetic_energy_column` — Name of the column in `input_file` giving values of the particle kinetic energy, in eV.
  - `yield_column` — Name of the column in `input_file` giving values of the mean yield. This is a ratio, giving the mean number of new electrons per incident electron.
  - `yield_limit` — If non-zero, this parameter limits from above the number of secondaries that can be emitted per primary particle. It can be helpful in preventing runaway, wherein the number of low-energy secondaries grows exponentially.
  - `emitted_momentum` — $\beta\gamma$ value for newly-emitted particles. The orientation of the momentum is random.
  - `verbosity` — Larger positive values result in more detailed printouts during the run.
  - `material_id` — A positive integer giving the material for which this command specifies secondary emission properties. This will be used together with the `material_id` parameter of the `point` namelists in the geometry file to determine the appropriate secondary emission properties for each segment of the cavity boundary.
  - `log_file` — A possibly incomplete name of an SDDS file to which secondary emission records will be written.

The algorithm is a simple one suggested by J. Lewellen (APS). We assume that the secondary emission yield is a function only of the incident particle's kinetic energy. Each time a particle is lost, the code determines where the particle intersected the metal boundary. The mean secondary yield is computed from the kinetic energy at the time of loss. The number of secondary particles emitted is chosen using a Poisson distribution with that mean. The secondary particles are placed "slightly" ($\Delta r/10^6$ or $\Delta z/10^6$) outside the metal surface.

To prevent runaway, the secondary yield curve should fall to zero for low energies. If you have problems with runaway, try setting the `yield_limit` parameter to a small positive integer. Runaway appears to be associated (at times) with the occasional production of large numbers of secondaries due to the tails of the Poisson distribution.

3.15 define_snapshots

- **description:** Requests beam snapshots at equispaced time intervals. These snapshots contain the spatial and momentum coordinates of all particles.

- **example:**

```
&define_snapshots
    filename = "gunRun.snap",
    time_interval = 0.1e-9
    start_time = 2e-9
&end
```

  This command results in snapshots being taken every 100ps starting at time 2ns, with data going to `gunRun.snap`.

- **synopsis and defaults:**

```
&define_snapshots
    STRING filename = NULL;
    double time_interval = 0;
    double start_time = 0;
&end
```

- **details:**
  - `filename`: The name of an SDDS file to which to write the snapshots.
  - `time_interval`: The interval in seconds between successive snapshots.
  - `start_time`: The time in seconds at which to make the first snapshot.

3.16 define_field_output

- **description:** Requests output of the field map into an SDDS file. A separate SDDS data page is made for successive maps. The interleaved simulation field grids are interpolated to give all field values on the same grid.

- **example:**

```
&define_field_output
    filename = "gunRun.fields",
    time_interval = 0.1e-9,
    start_time = 1e-9,
    z_interval = 2,
    r_interval = 2
&end
```

  This command results in output of the fields on a grid that is twice as coarse as the simulation grid in both z and r, starting at time 1ns and continuing at 100ps intervals thereafter. The data is put in an SDDS file named `gunRun.fields`.

- **synopsis and defaults:**

```
&define_field_output
    STRING filename = NULL;
    double time_interval = 0;
    double start_time = 0;
    long z_interval = 1;
    long r_interval = 1;
    long exclude_imposed_field = 0;
    long separate_imposed_field = 0;
&end
```

- **details:**
  - `filename`: Name of an SDDS file in which to put the data.
  - `time_interval`: Simulated time interval in seconds between successive maps.
  - `start_time`: Starting simulated time for data output.
  - `z_interval`: Interval at which grid points are spaced longitudinally in units of the simulation longitudinal grid size.
  - `r_interval`: Interval at which grid points are spaced radially in units of the simulation radial grid size.
  - `exclude_imposed_field`: Flag requesting that any fields specified with the `constant_fields` command should be excluded from the field output.
  - `separate_imposed_field`: Flag requesting that any fields specified with the `constant_fields` command should be included in the output as separate data elements.

3.17 define_field_saving

- **description:** Field saving refers to writing the simulation fields to disk in such a way that they can be reloaded in subsequent runs. At this time, it saves only the time-dependent field components, not the external fields defined with `define_solenoid` or `constant_fields`.

- **example:**

```
&define_field_saving
    filename = "gunRun.fsave",
    time_interval = 1e-9,
    double start_time = 5e-9
&end
```

  This command results in saving of the fields at 1ns intervals starting at simulated time of 5ns to file "gunRun.fsave". Successive saves are placed in successive SDDS pages.

- **synopsis and defaults:**

```
&define_field_saving
    STRING filename = NULL;
    double time_interval = 0;
    double start_time = 0;
    long save_before_exiting = 0;
&end
```

- **details:**
  - `filename`: Name of an SDDS file to which to write the field saves.
  - `time_interval`: Simulated time interval in seconds between successive saves.
  - `start_time`: Simulated time at which to make the first save.
  - `save_before_exiting`: Flag requesting that a save be made just prior to program termination.

3.18 define_field_sampling

- **description:** Requests sampling of the fields vs time or space coordinates. One may set up sampling of a specific field component along a line in z or r, with output vs the distance along the line at specified times or else output of the average value along the line vs time.

- **example:**

```
&define_field_sampling
    filename = "gunRun.Ez_t,
    component = "Ez",
    time_sequence = 1
    direction = "z",
    min_coord = 0, max_coord = 0.01,
    position = 0,
    time_interval = 10e-12,
    start_time = 0,
&end
```

  Requests logging of E_z as a function of time, with samples made a 10ps simulated time intervals starting with the start of the simulation. The data output is the average of E_z along a line from $z = 0$ to $z = 1$cm.

- **synopsis and defaults:**

```
&define_field_sampling
    STRING filename = NULL;
    STRING component = NULL;
    STRING direction = NULL;
    double min_coord = 0;
    double max_coord = 0;
    double position = 0;
    double time_interval = 0;
    double start_time = 0;
    long time_sequence = 0;
&end
```

- **details:**
  - `filename`: Name of an SDDS file to which to write the data.
  - `component`: Field component to sample. Must be one of "Ez", "Er", "Jz", "Jr", "Bphi", "Phi", "Ephi", "Bz", "Br", and "Q". E is the electric field, B is the magnetic field, J is the current density, Phi is the scalar potential, and Q charge assigned to a grid point.
  - `direction`: The direction of the line along which samples are taken. Must be "z" or "r".
  - `min_coord, max_coord`: The minimum and maximum coordinates of the sample line.
  - `position`: The position of the sample line in the direction orthogonal to `direction`.

– `time_sequence`: Flag requesting that instead of writing the selected component as a function of the coordinate `direction`, `spiffe` instead write the average value of the component along the sample line as a function of time. It zero, then `spiffe` creates a new SDDS page for each sample time.

– `start_time`: The simulated time in seconds at which to start sampling.

– `time_interval`: The simulated time interval in seconds at which to make samples.

3.19 integrate

- **description:** Defines integration parameters and begins integration. Allows specifying the integration time step, the total time to integrate, and other conditions of integration.
- **example:**

```
&integrate
    dt_integration = 1e-12,
    start_time = 0,
    finish_time = 5e-9,
    status_interval = 128,
    space_charge = 1
&end
```

Starts integration of equations for particles and fields at a simulated time of 0, taking steps of 1ps, until reaching 5ns. Every 128 steps, status information is printed to the screen. Space charge is included.

- **synopsis and defaults:**

```
&integrate
    double dt_integration = 0;
    double start_time = 0;
    double finish_time = 0;
    long status_interval = -1;
    long space_charge = 0;
    long check_divergence = 0;
    double smoothing_parameter = 0;
    double J_filter_multiplier = 0;
    long terminate_on_total_loss = 0;
    STRING status_output = NULL;
    STRING lost_particles = NULL;
&end
```

- **details:**
  - `dt_integration`: Simulation step size in seconds.
  - `start_time`: Simulation start time. Typically 0 for new runs. Ignored for runs that involve fields loaded from other simulations.
  - `finish_time`: Simulation stop time.
  - `status_interval`: Interval in units of a simulation step between status printouts.
  - `space_charge`: Flag requesting inclusion of space-charge in the simulation.
  - `check_divergence`: Flag requesting that status printouts include a check of the field values using the divergence equation.
  - `smoothing_parameter`: Specifies a simple spatial filter for the current density. The smoothing parameter, s, is used to compute two new quantities, $c_1 = 1-s$ and $c_2 = s/2$. The program smooths longitudinal variation for constant radius, using $A_o \rightarrow (A_- + A_+)c_2 + A_o c_1$, where A_o is the central value and $A_\pm$ are the adjacent values to a grid point. This function is rarely used and I do not recommend it.

– `J_filter_multiplier`: Specifies a simple time-domain filter for the current density. For each point on the grid, the new current density value $J(1)$ is replaced by $J(1) * (1 - f) + J(0) * f$. This is an infinite impulse response filter. This function is rarely used and I do not recommend it.

– `terminate_on_total_loss`: Flag requesting that when all simulation particles are lost (e.g., by hitting a wall or exiting the simulation), the simulation should terminate.

– `status_output`: Provide the name of a file to which to write status information, including statistics on the beams and fields. File is in SDDS format.

– `lost_particles`: Provide the name of a file to which to write information about particles that get lost. File is in SDDS format.

References

[1] H. G. Kirk *et al.* in *Handbook of Accelerator Physics and Engineering*, A. W. Chao and M. Tigner eds., section 2.4.2.1, World Scientific, 1999, page 99.

[2] M. Borland, *Summary of Equations and Methods Used in* `spiffe`, APS/IN/LINAC/92-2, 29 June 1992.