

CSRtrack Version 1.1 User's Manual

1. Introduction

The CSRtrack code tracks particle ensembles through beam lines with arbitrary geometry. The field calculation in CSRtrack offers different algorithms to choose from, from the fast 'projected' 1-D method to the most rigorous, the three-dimensional integration over 3D Gaussian sub-bunch distributions (see section '[forces](#)').

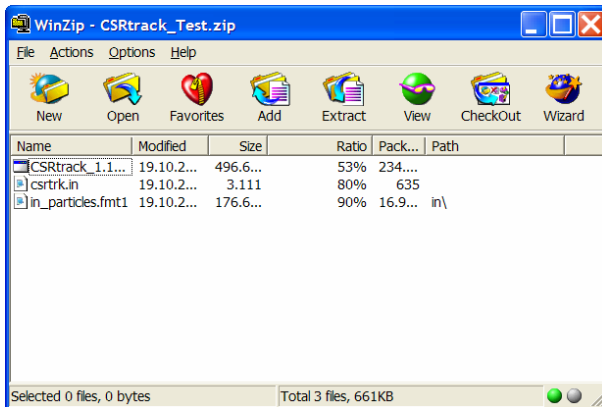
The 3D field calculations take Coherent Synchrotron Radiation fields into account as well as intra-bunch fields, similar to the well-known space-charge fields on straight trajectories, but, on curved paths, not cleanly separable from radiative fields any longer.

Tracking is done in absolute coordinates through a magnet lattice defined by magnet field boundaries (see section '[lattice](#)'). CSRtrack handles dipole, quadrupole and multipole magnets.

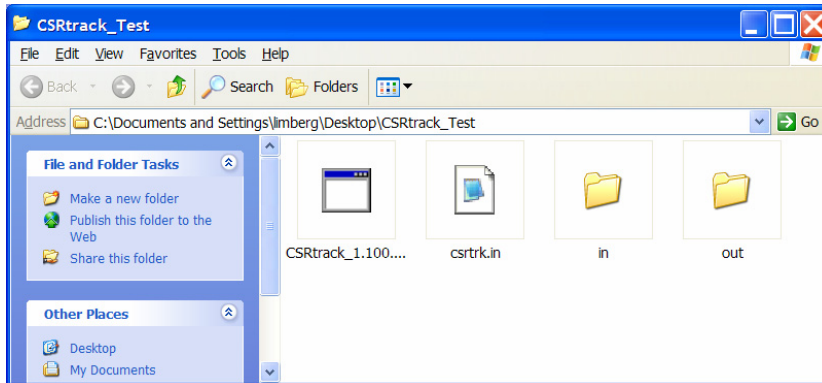
RF sections are not implemented yet. Tracking through long straight RF sections is better left to codes like elegant or ASTRA, depending on the importance of space charge force.

2. First Steps to run CSRtrack on a WINDOWS XP Computer

Download the file 'CSRtrack_example1.zip' and open it. You should see the following:

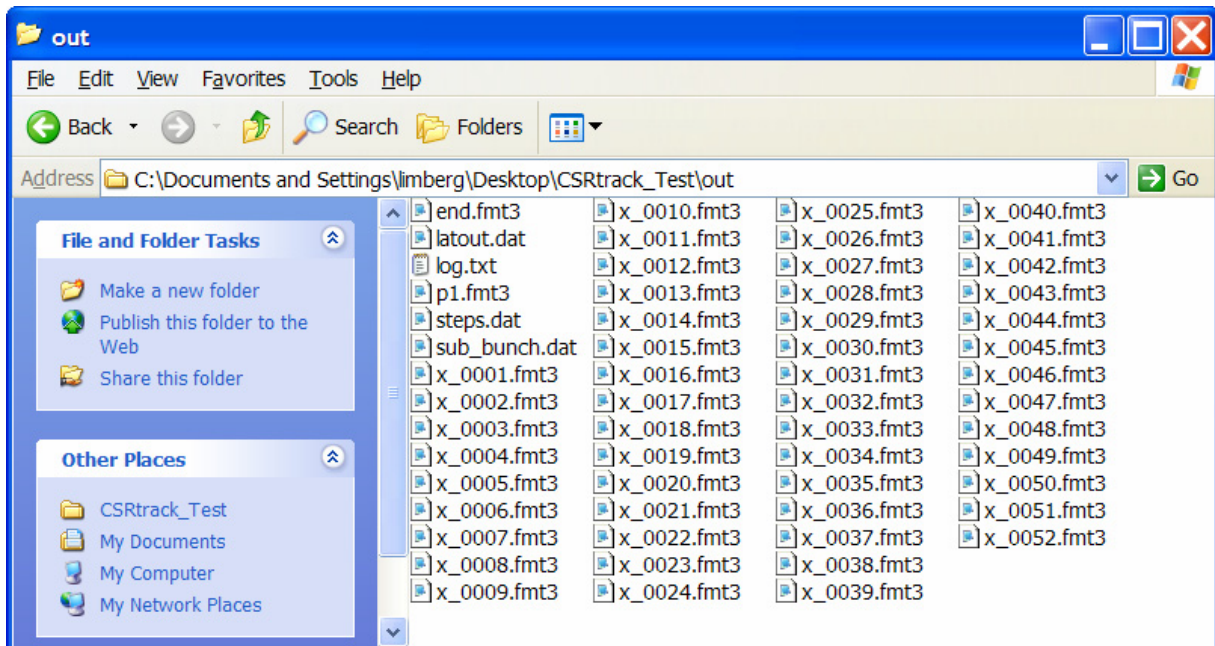


Extracting all files into a new folder (called 'CSRtrack_Test' here) should lead to the following if you explore the folder:



The folder 'in' contains the particle distribution file 'in_particles.fmt1' (see section 'particles'), the 'out' folder is empty. The CSRtrack executable expects to find a 'csrtrk.in' file in the same folder it is located. That file can be a complete description of the run or it may refer to other input files (see '[Command File](#)').

Double-clicking the executable runs CSRtrack and the following files should appear in the 'out' folder:



Now you ran CSRtrack successfully for the first time. For a better understanding what you calculated and how the output is generated, let's look into the 'csrtrk.in' file. It starts with the specification of in- and output paths (see section [io_path](#)), starting the path in the folder where the executable is located.

```
!-----
io_path{input =in,output=out,logfile=log.txt}
```

So our folders ‘in’ and ‘out’ are specified as targets for in- and output.

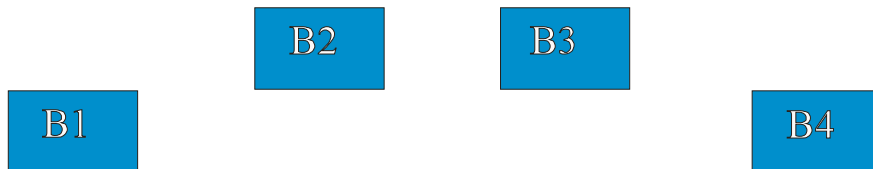
Now the lattice is defined (see section ‘[lattice](#)’):

```

!-----
! 4 magnet bunch compressor
!-----
lattice{
  dipole ! 1st dipole
    {position{rho=0.0,psi=0.0,marker=d1a}
     properties{r=-8.4}
     position{rho=0.5,psi=0.0,marker=d1b}
    }
  dipole ! 2nd dipole
    {position{rho=1.0,psi=0.0,marker=d2a}
     properties{r=8.4}
     position{rho=1.5,psi=0.0,marker=d2b}
    }
  dipole ! 3rd dipole
    {position{rho=2.5,psi=0.0,marker=d3a}
     properties{r=8.4}
     position{rho=3.0,psi=0.0,marker=d3b}
    }
  dipole ! 4th dipole
    {position{rho=3.5,psi=0.0,marker=d4a}
     properties{r=-8.4}
     position{rho=4.0,psi=0.0,marker=d4b}
    }
}

```

And describes a four dipole magnet chicane



with the following parameters:

chicane		
bend magnet length (projected)	0.5 m	
drift length,(proj.) B1->B2 and B3->B4	0.5 m	
drift length, B2->B3	0.5 m	
bend radius	8.4 m	
momentum compaction	6 mm	

The particles distribution to be tracked is specified next:

```

!-----
! particle distribution
!-----
particles{reference momentum =reference particle
          reference point x   =0.0
          reference point y   =0.0
          reference point phi =0.0
          format=fmt1,array=#file{name=in particles.fmt1}
        }

```

After some options to change the reference system, the distribution is referenced to the file we have already seen in the ‘in’ folder. The format is ‘fmt1’ in this case (see section ‘[particles](#)’).

The example consists of a Gaussian particle distribution (~1000 particles). Beam parameters are:

bunch		
energy	511 MeV	
charge	0.833 nC	
bunch length (in)	80 μm	
bunch length (out)	20 μm	
peak current (out)	5 kA	
horizontal twiss parameters (in):		
normalized emittance	1 mm mrad	
alpha	2.2	
beta	10 m	
particle distribution		
number of particles	997	
number of slices	83	
particles per slice	12	
reference particle without charge		
sub-bunch length	5.3 μm	
sub-bunch width (horizontal)	33 μm	
sub-bunch width (vertical)	50 μm	

The output of CSRtrack is prompted by so-called monitors: ‘Online monitors’ write data to file during the tracking, ‘offline monitors’ save data at the beginning or the end (see ‘[monitors](#)’)

```

!-----
! online monitors
!-----
online_monitor{name=sub_bunch.dat,type=subbunch

```

```

        start time c0=now
        end time marker=d4b,end time shift c0=2.0
        time step c0=all
    }
online monitor{name=steps.dat,type=steps
               start time c0=now
               end time marker=d4b,end time shift c0=2.0
               time step c0=all
            }
online monitor{name=p1.fmt3,type=phase,format=fmt3,particle=1
               start time c0=now
               end time marker=d4b,end time shift c0=1.0
               time step c0=all
            }
online monitor{name=x.fmt3,type=phase,format=fmt3,particle=all
               start time c0=now
               end time marker=d4b,end time shift c0=1.0
               time step c0=0.10
            }
!-----

```

where name specifies the file name(s) to write the data to, accordingly the files named 'sub_bunch.dat' etc. appear in our 'out' folder. Details can be found in the section '[monitors](#)'; in the example above data describing sub-bunch length, time-step-width and the phase-space position of the reference particle along the beam-line are dumped in single files while the x_n files contain the phase-space coordinates for all particles at positions 10 cm apart from start to end_time_marker+end_time_shift_c0 [m].

The method to calculate CSR fields and the parameters of the sub-bunches are chosen in the '[forces](#)' command:

```

! force definition
!-----
forces{type=projected
       sigma long= 5.3e-6
      }

```

In this example, the 1-D projected field calculation method is used with a longitudinal size of 5.3 μ m. See chapter '[forces](#)' for details and other field solvers.

Finally, the range for the tracking calculation and the numerical parameters for the self-consistent, iterative particle tracking are specified:

```

!-----
! tracking
!-----
track step{precondition=yes
           iterative=2
           error per ct=0.001
           error weight momentum=0.0

           ct step min=0.02
           ct_step_max=0.10
          }

```

```

ct step first=0.10
increase factor=1.5
arc factor=0.3
duty steps=yes
}

```

```

tracker{end_time_marker=d4b,end_time_shift_c0=1.00}

```

Here, the ‘tracker’ command tells the code to track to the marker ‘d4b’, which is associated with the end of the last bend (see the lattice definition above) plus an additional time interval which corresponds to the reference particle traveling the length of 1 m with speed c_0 (end_time_shift_c0=1.00).

The ‘track_step’ command specifies that iterative tracking will be performed until two iterations are done or the error criterion is reached. The parameter `ct_step_min` and the ones below control the step-width algorithm. For details see [‘tracking’](#).

Finally, another monitor command writes the phase space at the end of the tracking to the file ‘end.fmt3’:

```

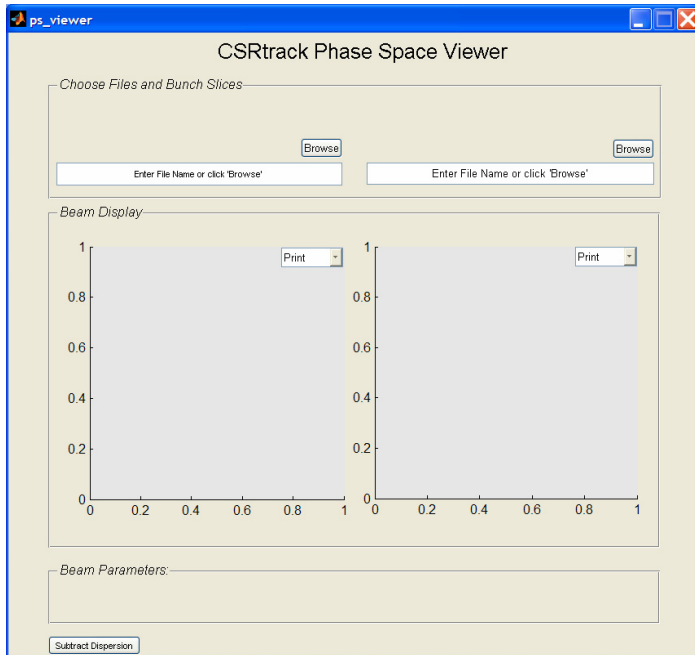
!-----
! offline monitors
!-----
monitor{format=fmt3,name=end.fmt3}

```

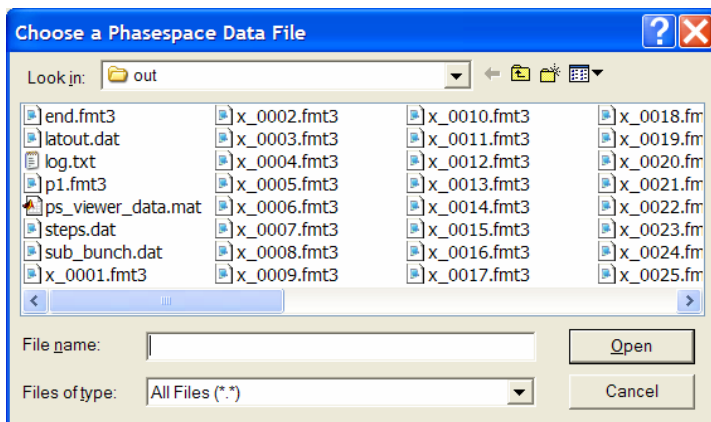
The calculation is started by running CSRtrack in the directory with the input file. The particle distribution is read from `in/particles_in.fmt1` and all output files are written to the directory `out`. This should create the same files as in the folder `out_solved`.

2.1. Plotting Results

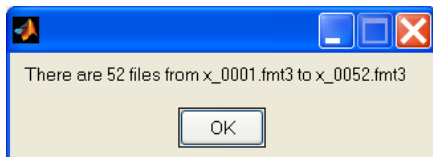
A MATLAB GUI called ‘CSRtrack_ps_viewer’ is available to plot CSRtrack results. Download the file ‘ps_viewer.zip’ and extract the files in a folder ‘ps_viewer’. Open MATLAB, choose that folder as working directory and enter ‘ps_viewer’ on the MATLAB command line. You should see the following GUI:



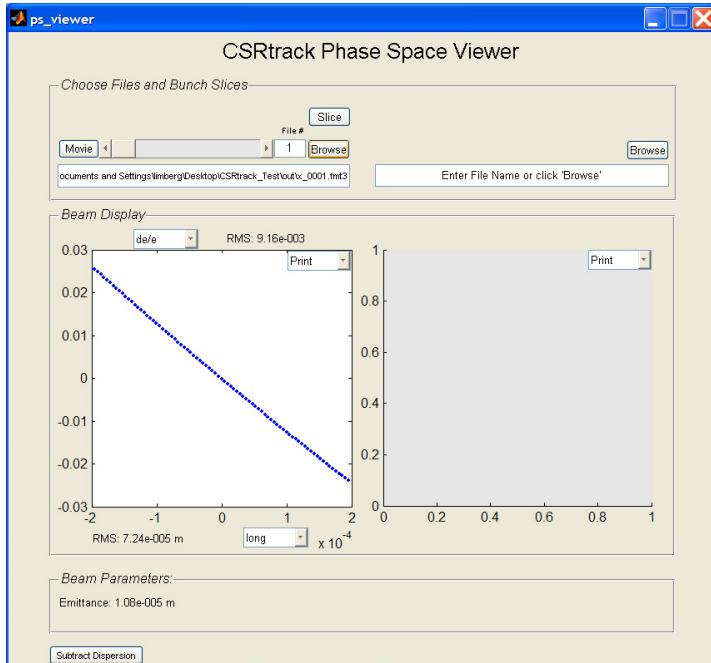
Basically there are two axes to plot phase space projections for comparison. Use the ‘Browse’ button to open one of the result files you have in the ‘out’ folder in your CSRtrack_test folder. The browse window should show the following files:



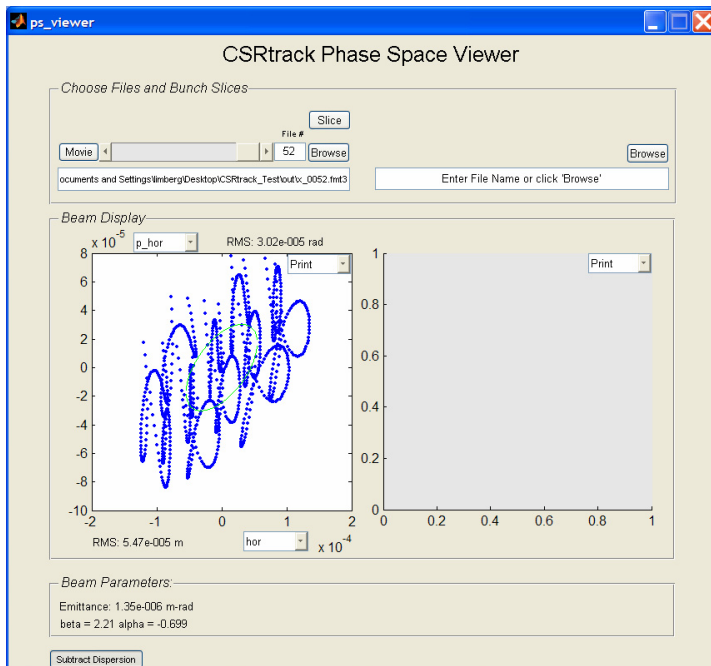
The ps_viewer reads phase space distributions saved in the CSRtrack formats .fmt1 (like the input file) or .fmt3. Open the x_0001.fmt3 file and the longitudinal phase space at this position will appear, after a message box



has informed you that there are 52 files along the beam line. A 'Movie' button pops up when there are more than 3 files of such type and you can push it to see a movie of how the longitudinal phase space evolves along the chicane.



Other phase space projections can be plotted by using the pull-down menus at the axis' of the plot. The following shows the horizontal phase space at the end of the chicane (the slider or the eiting field can be used to go back and forth between the different x_nnnn files):



When ‘normal’ transverse phase space (like $x-x'$, here hor for horizontal and p_{hor} for normalized horizontal momentum) is plotted, the emittance and the optics is calculated, else only the RMS values.

2.2. Calculation with force type = g_to_p

The command file is the same as for force type = **projected** with exception of the force definition:

```
!-----
! force definition
!-----
forces{type=csr g to p,shape=ellipsoid
      sigma long= 5.3e-6
      sigma rad =33.0e-6
      sigma vert=50.0e-6
}
```

2.3. Calculation with force type = p_to_p

The command file is the same as for force type = **projected** with exception of the force definition:

```
!-----
! force definition
!-----
forces{type=csr p to p,shape=ellipsoid
      sigma long= 5.3e-6
      sigma rad =33.0e-6
      sigma vert=50.0e-6
}
```

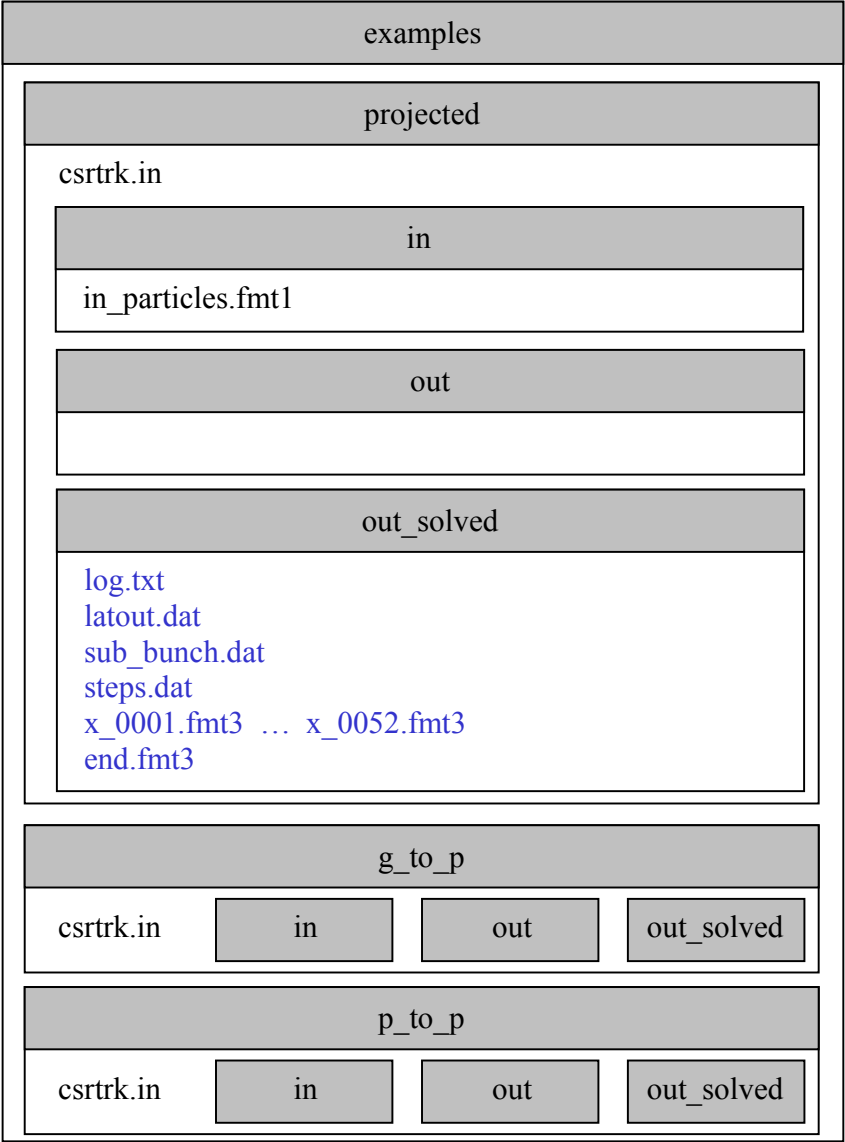
The dimensions of the sub-bunches are summarized in the following table.

Sub-Bunch Dimensions		
sub-bunch length	5.3 μm	
sub-bunch width (horizontal)	33 μm	
sub-bunch width (vertical)	50 μm	

Results for the different calculations types are plotted in the appendix.

2.4. CPU Time and File Structure

Calculation time (on a PC from 2004) for the **projected** method is less than 1minute, for the **g_to_p** method 30 minutes and for the ‘direct’ **p_to_p** method 8.5 hours.



3. Command Structure

3.1. Command File

CSRtrack reads all commands from the ascii input file 'csrtrk.in'. It has the following structure:

```
commands exit <CR>
```

with

```
commands = command [commands]  
command = comment / global_command / section  
comment = ! text <CR>  
separator = <blank> / , / <CR>
```

The input file can be used to specify and open further input files (see **#file**). The length of command lines in input files is limited to 400 characters.

3.2. Sections

```
section_name { section_body }
```

A section-call causes three activities:

- 1) The section is initialized after the opening bracket '{',
- 2) Global commands and specific section commands are valid in the section body. Section specific commands are either assignment statements or nested sections. The assignment of section parameters can be done in any succession. The only exception is the particles definition (see section **particles**).
- 3) The section action is started after the closing bracket '}'.

3.3. Global Commands

There is only one global command in version 1.0:

```
global_command = #file{name = filename}  
filename = < name of nested input file >
```

The **#file** command opens a nested input file. The commands in this file are processed in the same way as that in the command file 'csrtrk.in'. Maximal 10 nested files can be opened at once. Each nested file has to end with <CR>. After the processing of a nested file, CSRtrack continues processing of commands in files with higher level or of commands in 'csrtrk.in'.

CSRtrack searches the input file *filename* either in the root directory (with 'csrtrk.in') or, if specified, in the input directory that is defined by the section **iopath**.

3.4. CSRtrack Section Menu

section name	
io_path	file io
lattice	lattice definition
particles	definition of particle distribution
track_step	Time grid, iterative tracking
tracker	tracking
forces	model for self forces
monitor	monitor
online_monitor	online monitor

3.4.1. Section: io_path

The **io_path** section is used to specify the input- and output-directories. If these directories are unspecified, input- and output-files are read from or written to the root directory (with 'csrtrk.in').

identifier	argument	unit / type
input	input directory	character string
output	output directory	character string
logfile	file name	character string

Example:

```
io path{input  =data/bc2/in,
        output =data/bc2/out,
        logfile=log.txt}
```

has the same effect as:

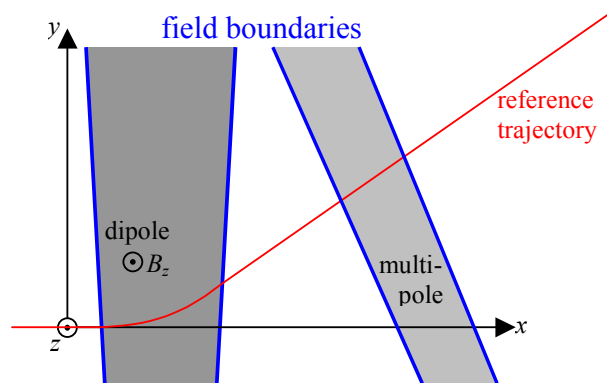
```
io path{logfile=log.txt,
        input  =data/bc2/in,
        output =data/bc2/out}
```

The logfile is written to 'data/bc2_100/out/log.txt'.

3.4.2. Branch Section: lattice

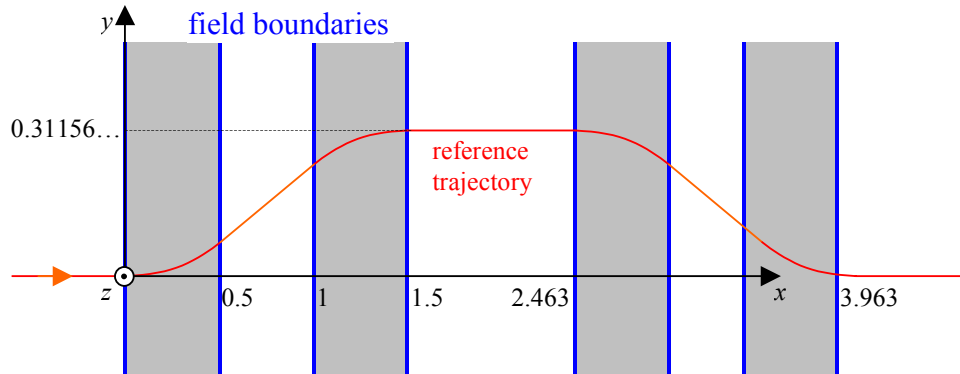
subsection name	
dipole	definition of dipoles
quadrupole	definition of quadrupoles
multipole	definition of multipoles

a) Concept: CSRtrack supports magnetic dipole- and multipole-fields that are defined in specified (x,y,z) coordinates. The range of these fields is defined by two **field-boundaries** that are perpendicular to the xy -plane. The dipole field between field boundaries is constant and parallel to the z -axis. Together with the dipoles a **reference trajectory** is defined that is composed by arcs and lines and lies in the xy -plane. The first part of the reference trajectory coincides with the x -axis. The definition of each lattice element has three parts: the definition of the first field-boundary, the definition of element properties and the definition of the second field boundary.



b) Example:

```
lattice{
  dipole ! 1st dipole
    {position{rho=0.0 ,psi=0.0,marker=d1a}
     properties{r=-1.66275}
     position{rho=0.5 ,psi=0.0,marker=d1b}
    }
  dipole ! 2nd dipole
    {position{rho=1.0 ,psi=0.0,marker=d2a}
     properties{r=1.66275}
     position{rho=1.5 ,psi=0.0,marker=d2b}
    }
  dipole ! 3rd dipole
    {position{rho=2.463,psi=0.0,marker=d3a}
     properties{r=1.66275}
     position{rho=2.963,psi=0.0,marker=d3b}
    }
  dipole ! 4th dipole
    {position{rho=3.463,psi=0.0,marker=d4a}
     properties{r=-1.66275}
     position{rho=3.963,psi=0.0,marker=d4b}
    }
}
```

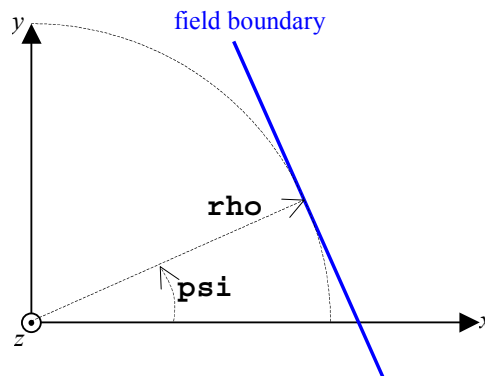


3.4.2.1. Definition of Field-Boundaries (Subsection: position)

identifier	argument	unit / type
rho	value	length / number
psi	value	angle / number
delta_s	value	length / number
delta_psi	value	angle / number
marker	marker name	character string
duty	yes / no	

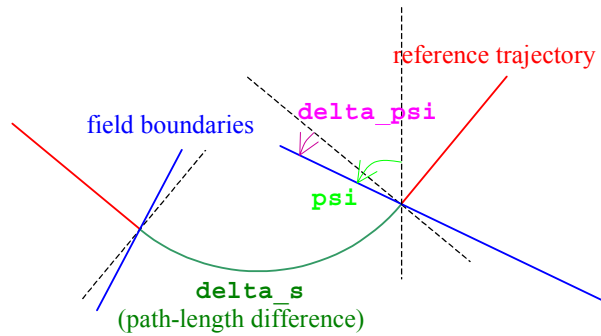
The position section is used to define field-boundaries. They can be defined absolute or relative.

a) Absolute definition of field boundaries: The field boundary is defined in polar coordinates by the parameters **rho** and **psi**. The definition of the first field boundary has to be absolute. The reference trajectory before the first field boundary is identical to the x -axis. The first reference point is the intersection of the first reference plane and the x -axis. The rest of the reference trajectory (and all later reference points) are recursively defined by dipoles and their curvature radii.



b) Relative definition of field boundaries: If the position and direction of the reference trajectory are defined for one field boundary, the position and direction of the intersection with the next field boundary is uniquely defined by the path-length difference **delta_s** and the curvature

radius r (if a dipole is bounded). The orientation of the field boundary is either specified by ψ or $\Delta\psi$ or it is perpendicular to the reference trajectory. ψ defines the absolute orientation (in the same way as for the absolute definition) and $\Delta\psi$ defines the angle between field boundary and the plane perpendicular to the trajectory at the intersection point.



c) **marker**: The identifier **marker** is used to assign a name to a field boundary. The marker names can be used to specify time events eg. The instantaneous time when the reference particle (see particle definition) passes a field boundary. This identifier is optional, its argument is a character string.

d) **duty**: The identifier **duty** can be used to affect the step width control of the tracking algorithm (see **track_step**). This identifier is optional, its argument is **yes** or **no**. If duty is not specified, it is set to **yes** for field boundaries of dipoles and to **no** for the rest.

e) Example

```
position{delta_s=0.2,marker=quad_in,duty=no}
```

3.4.2.2. Branch Section: Dipole

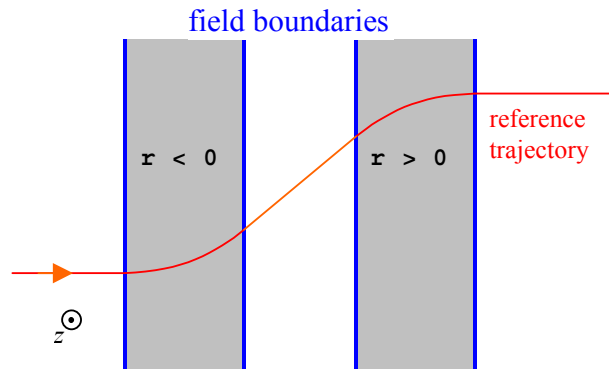
subsection name	
position	definition of field boundaries
properties	dipole properties

a) Subsection **position**: See ‘Definition of Field Boundaries’.

b) Subsection **properties**:

identifiers	argument	unit / type
r	value	length / number

r sets the curvature radius of the reference trajectory. In combination with the reference momentum (see **particles**) the strength of the dipole field is uniquely determined. To take into account vertical effects (z -direction) by edge \square pecify, the tracking algorithm applies a vertical kick proportional to the offset from x - y -plane at the field boundaries.



c) Example:

```
dipole ! 1st dipole
  {position{rho=0.0 ,psi=0.0,marker=d1a}
   properties{r=-1.66275}
   position{rho=0.5 ,psi=0.0,marker=d1b}
  }
```

3.4.2.3. Branch Section: quadrupole

subsection name	
position	definition of field boundaries
properties	quadrupole properties

The definition of quadrupoles is identical to that of multipoles that is described in the next subsection. The only difference is the parameter **poles** that obsolete. Example:

```
quadrupole{ position{delta s=0.2,marker=quad in,duty=no}
  properties{strength=0.100,alpha=0
             horizontal offset=0,vertical offset=0}
  position{delta s=0.5,marker=quad out,duty=no}
}
```

3.4.2.4. Branch Section: multipole

subsection name	
position	definition of field boundaries
properties	multipole properties

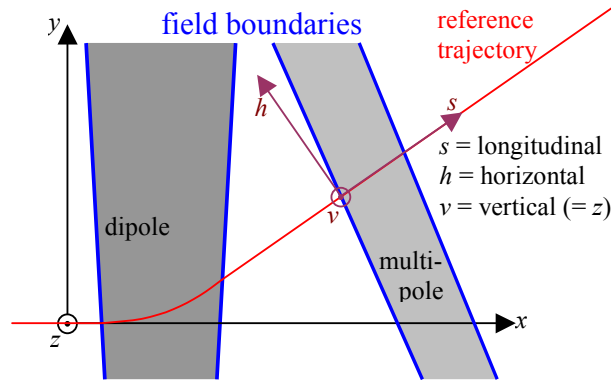
a) Subsection **position**: See ‘Definition of Field Boundaries’.

b) Subsection **properties**:

identifier	argument	unit / type
strength	value (a)	length ⁽⁻ⁿ⁾ / number
alpha	value (α)	angle / number

horizontal_offset	value (h_0)	length / number
vertical_offset	value (v_0)	length / number
poles	value ($2n$)	integer

The magnetic multipole field is defined in a local specified coordinate system (v, h, s) with its origin in the intersection point of the reference trajectory and the field boundary:



The magnetic field between the field boundaries is independent on the longitudinal coordinate. In complex notation the horizontal and vertical components B_v and B_h of the field are:

$$B_v + jB_h = \frac{p_r}{q} \frac{a}{(n-1)!} ([h-h_0] + j[v-v_0])^{n-1} e^{-jn\alpha}$$

with q the particle charge, p_r the reference momentum, n the azimuthal order, a the multipole strength, h_0 , v_0 the horizontal and vertical offset and α the skew angle. These parameters are related to the **properties** identifiers by:

poles = $2n$ (= 4 for quadrupoles)
strength = a
horizontal_offset = h_0
vertical_offset = v_0
alpha = α

The parameters **horizontal_offset**, **vertical_offset** and **alpha** need not to be specified. Their default is zero.

c) Example:

```

multipole{ position{delta s=0.2,marker=quad in,duty=no}
            properties{poles=4
                        strength=0.100,alpha=0
                        horizontal offset=0,vertical offset=0}
            position{delta s=0.5,marker=quad out,duty=no}
            }

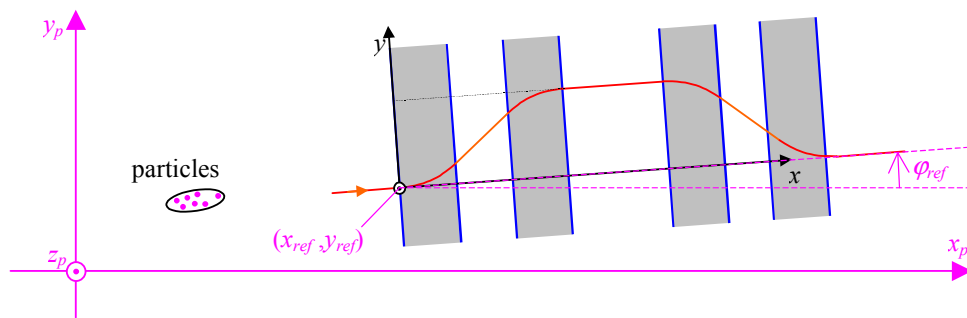
```

3.4.3. Assignment Section: particles

identifier	argument	unit / type
format	fmt1 / fmt2 / astra	
reference_momentum	value (p_r) / reference_partice / average	momentum / number
reference_point_x	value (x_{ref})	length / number
reference_point_y	value (y_{ref})	length / number
reference_point_phi	value (φ)	angle / number
array		

a) Concept: The **particles** section is used to define the position, momentum and charge of a particle distribution in a absolute specified coordinate system (x_p, y_p, z_p). The first particle of this distribution is called **reference particle**. The reference particle is treated as all other particles, but some input and output parameters are defined relative to the reference particle. Eg. A time event may be defined by the transition of the reference particle through a field boundary, or the coordinates and momenta of other particles are given as increment to that of the reference particle. The **reference momentum** relates the **lattice** settings to absolute field strengths. (The reference momentum is not necessarily the momentum of the reference particle.) The properties of the particle distribution as well as the absolute time are defined in an array that is assigned to **array**. This assignment is terminated by the closing bracket of the section. Therefore **array** has to be the last assignment.

b) (x_p, y_p, z_p)-coordinates: The (x_p, y_p, z_p) coordinate system of the particle definition is related to the coordinate system of the lattice definition by:



$$x = (x_p - x_{ref}) \cos \varphi + (y_p - y_{ref}) \sin \varphi$$

$$y = (x_{ref} - x_p) \sin \varphi + (y_p - y_{ref}) \cos \varphi$$

$$z = z_p$$

c) **reference_momentum**: The reference momentum is used to relate the curvature radii and strengths settings (**r**, **strength**) in **lattice** to magnetic field strengths:

$$\text{reference_momentum} = \text{value}(p_r) / \frac{\text{reference_particle}}{\text{average}}$$

Either the reference momentum is specified directly by value, or it is set to the momentum of the reference particle or to the average momentum of all particles.

d) **format** and **array**: Three different input formats are available to define particle distributions:

Format 1:

```
format = fmt1
array = t   r1   r2   r3   r4   r5   r6
        xr  yr  zr  pxr  pyr  pzr  q1
        δx2 δy2 δz2 δpx2 δpy2 δpz2 q2
        δx3 δy3 δz3 δpx3 δpy3 δpz3 q3
        Λ
        δxn δyn δzn δpxn δpyn δpzn qn
```

The arguments of the **array** command are processed in the same way as the rest of the command file. Therefore the input can be directed to an other input file by the global command

```
#file{name = filename}
```

and comments as well as all CSRtrack separators are valid. The end of the argument list is defined by the closing bracket of the **particles** section. In format 1 CSRtrack expects $7(N+1)$ numerical arguments. The first number t defines the time of the distribution. The next six values ($r_1, r_2, r_3, r_4, r_5, r_6$) have no meaning and do not affect the result of the calculation. The triplets (x_r, y_r, z_r), (px_r, py_r, pz_r) define the position and momentum of the reference particle in (x_p, y_p, z_p)-coordinates. Q_1 is the charge of the reference particle. Each further particle is defined by an additional set of numbers ($\delta x_i, \delta y_i, \delta z_i, \delta p x_i, \delta p y_i, \delta p z_i, q_i$) with the position ($x_r + \delta x_i, y_r + \delta y_i, z_r + \delta z_i$), the momentum ($px_r + \delta p x_i, py_r + \delta p y_i, pz_r + \delta p z_i$) and the charge q_i .

Format 2:

```

format = fmt2
array = t    r1  r2  r3  r4  r5  r6
        xr  yr  zr  pxr  pyr  pzr  q1
        δ2  δh2  δv2  δps2  δph2  δpv2  q2
        δ3  δh3  δv3  δps3  δph3  δpv3  q3
        Λ
        δn  δhn  δvn  δpsn  δphn  δpvn  qn

```

The structure of format 2 is the same as for format 1. Position and momentum of the reference particle are defined as before. For all other particles the position- and momentum-differences to the reference particle are given in (s, h, v) coordinates. The direction of the s -axis is defined by the direction of the reference particle. The orthogonal h - and v -directions follow from:

$$\mathbf{u}_s = \mathbf{p}_{reference_particle} / p_{reference_particle}$$

$$\mathbf{u}_h = \mathbf{u}_z \times \mathbf{u}_s$$

$$\mathbf{u}_v = \mathbf{u}_s \times \mathbf{u}_h$$

$$\mathbf{r}_i = \mathbf{r}_{ref} + \delta s_i \mathbf{u}_s + \delta h_i \mathbf{u}_h + \delta v_i \mathbf{u}_v$$

$$\mathbf{p}_i = \mathbf{p}_{ref} + \delta p s_i \mathbf{u}_s + \delta p h_i \mathbf{u}_h + \delta p v_i \mathbf{u}_v$$

Astra-format:

```
format = astra
```

Astra coordinates are converted to CSRtrack coordinates by the permutation: $(z, x, y)_{Astra} \rightarrow (x_p, y_p, z_p)$.

e) Example:

```

particles{reference momentum =reference particle
reference point x =0.0
reference point y =0.0
reference point phi =0.0
format=fmt1,array=#file{name=particles bc2.fmt1}
}

```

3.4.4. Section: track_step

identifier	argument	unit / type
precondition	yes / no	
iterative	value (<i>it</i>)	integer ≥ 1
error_per_ct	value (<i>err</i>)	time ⁻¹ / number > 0

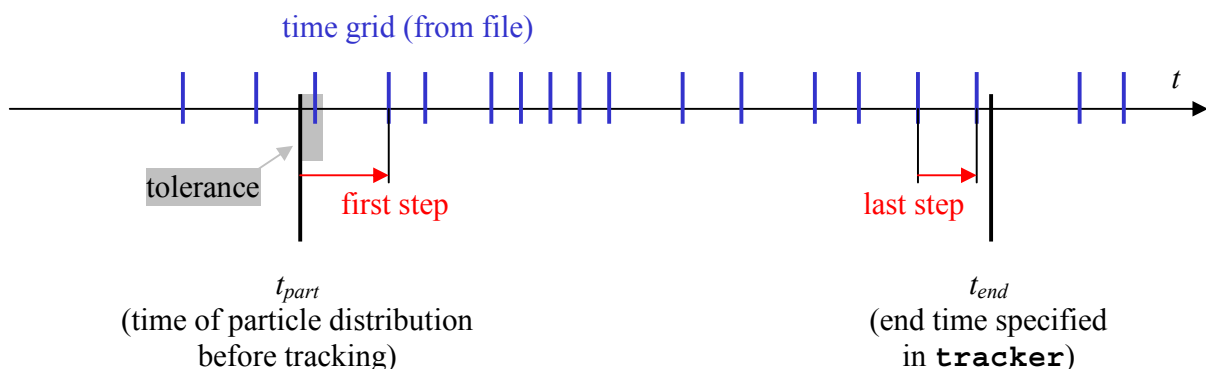
error_weight_momentum	value (w_{mom})	number ≥ 0
ct_step_min	value (t_{min})	time / number ≥ 0
ct_step_max	value (t_{max})	time / number ≥ 0
ct_step_first	value (t_{first})	time / number ≥ 0
duty_steps	yes / no	
increase_factor	value (f_{inc})	number ≥ 1
arc_factor	value (f_{arc})	number ≥ 1
time_grid_file	file name	character string
steps_tolerance	value (tol)	time / number > 0

a) Concept: CSRtrack calculates self-forces on a **time grid** and interpolates them linearly for particle tracking. The section **track_step** is used to control the time grid as well as the tracking from one grid point to the next. There are two possibilities to determine the time grid or the widths of time steps: either an external file defines the grid directly or the step widths are set recursively. For a new force calculation the phase space coordinates all particles have to be known, but they depend on the force that has to be determined. This implicit problem is solved recursively (**iterative tracking**). The time steps and the parameters for iterative tracking are crucial for the accuracy of the calculation. The **track_step** settings can be redefined eg. Before a consecutive call of the **tracker** section.

b) **time_grid_file**: The time grid can be defined directly by the uses of a file with time grid values. The filename and a tolerance parameter are set by the following commands:

```
time_grid_file = file name
steps_tolerance = value (tol)
```

CSRtrack expects an ascii input file with the specified name in the input- or root-directory (see **io_path**). It reads a list of time grid values, one value per input line with the time unit (1 m / c_0).



CSRtrack processes time grid values between the actual time associated to the particle distribution (t_{part}) to the end time (t_{end}) that is defined in the section **tracker**. Time grid values are ignored if they need steps smaller than the tolerance parameter. The tolerance parameter (tol) is used to avoid a first step of zero length or to avoid extremely small steps.

For the preparation of the time grid, the user has to take care about the position of the particles distribution in the lattice. The steps-monitor (see **online_monitor**, **type=steps**) can be used to support this task. E.g. a new time grid can be derived from an old one that was generated by a calculation with recursive step widths control.

c) Recursive calculation of time steps: The automatic time stepping is active if no time grid file is specified. It is controlled by the parameters

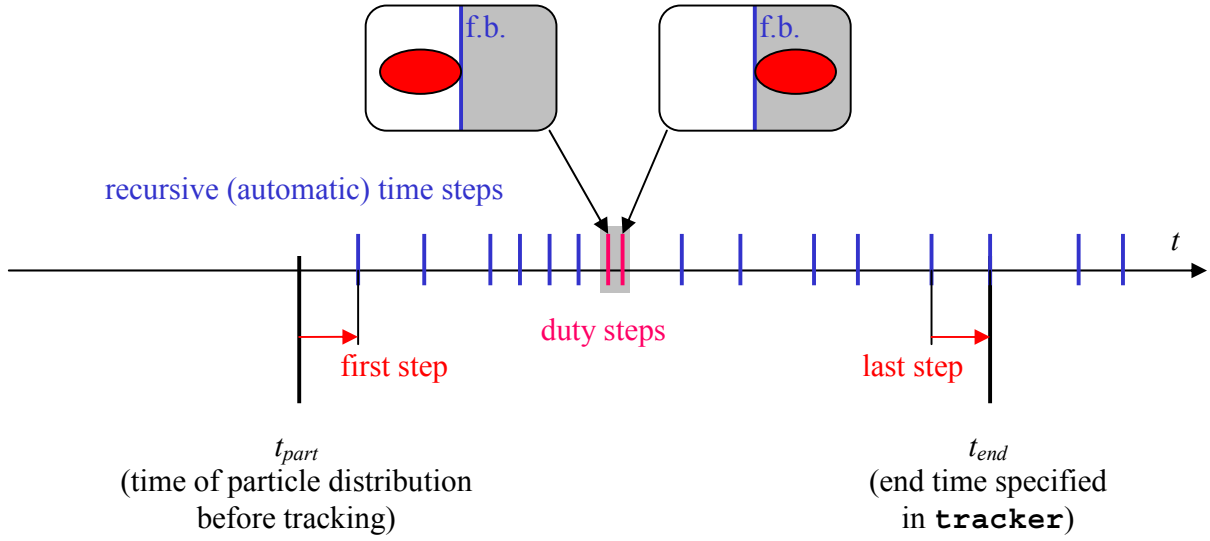
```

ct_step_max      =  $t_{max}$ 
ct_step_min      =  $t_{min}$ 
ct_step_first    =  $t_{first}$ 
duty_steps       = yes / no
increase_factor  =  $f_{inc}$ 
arc_factor       =  $f_{arc}$ 

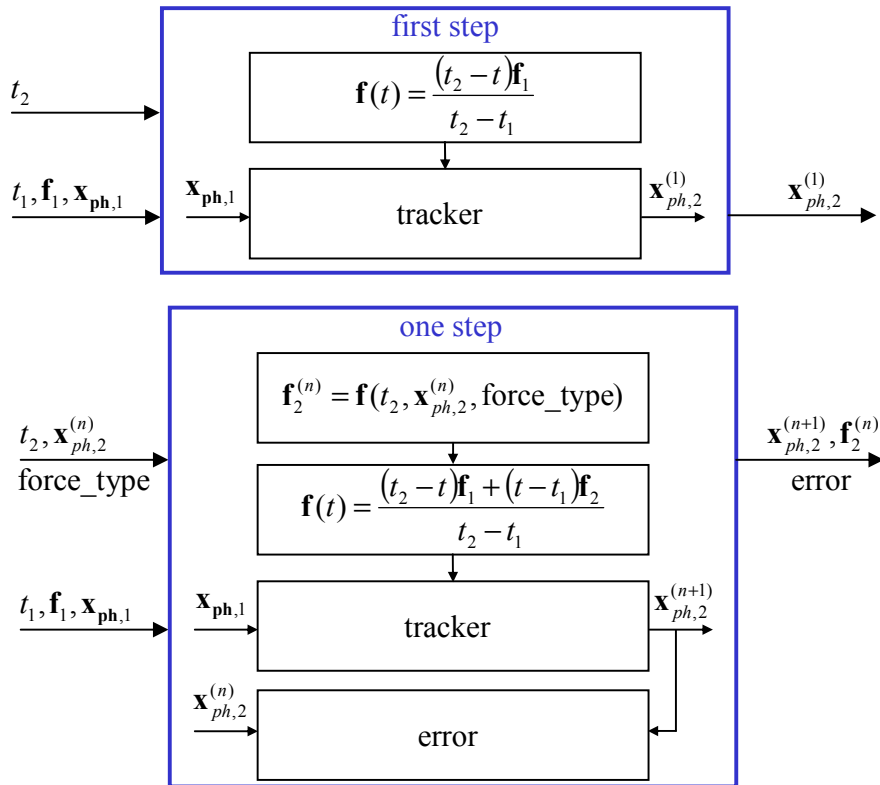
```

If **duty_steps** is set to ‘**no**’ or if no field boundary is passed during the track step CSRtrack uses time steps longer or equal t_{min} . The maximal time step is limited by $\min(t_{min}, t_{arc})$ with $t_{arc} = f_{arc} \sqrt[3]{24R_c\sigma_{rms}}$ with R_c the actual curvature radius of the trajectory and σ_{rms} the (actual) rms length of the particle distribution. It starts with a step of the length t_{first} and increases it for each new step by the factor f_{arc} until it is limited by $\min(t_{min}, t_{arc})$.

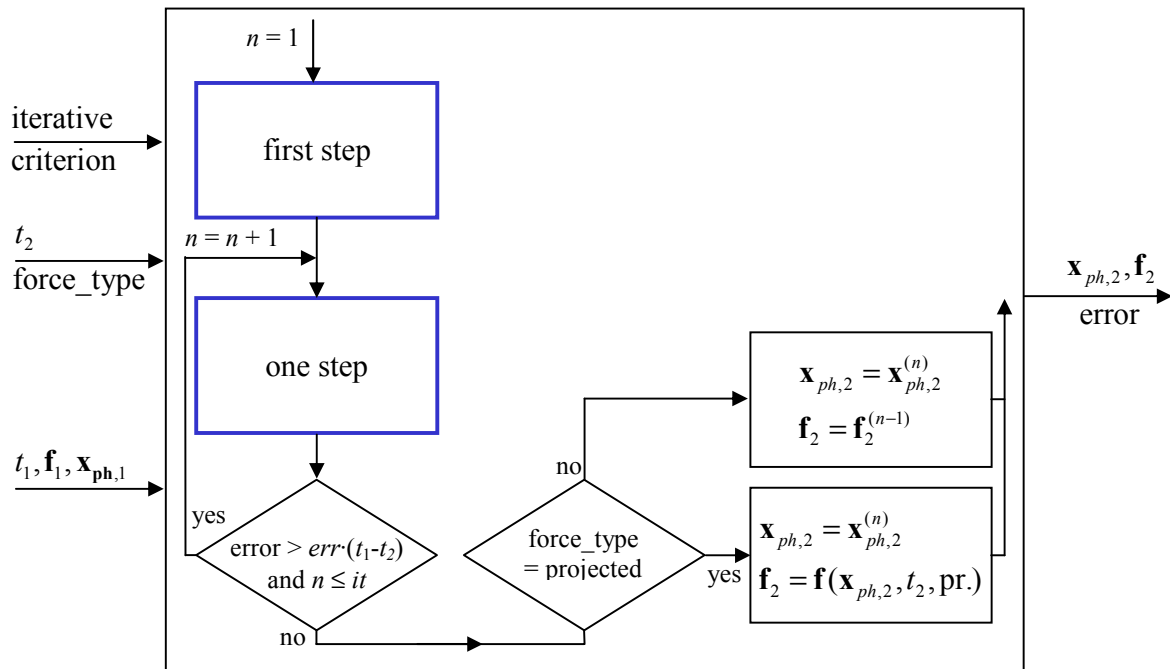
The time dependency of forces is usually slowly compared to the bunch length. Transition processes are typically of the order of the formation time $c^{-1} \cdot \sqrt[3]{24R_c\sigma_{rms}}$. This is different if the bunch shape or the curvature radius change rapidly. The **duty_steps** command is used to consider fast transient processes (especially of radial forces) at field boundaries. CSRtrack uses extra grid points for the transition of field boundaries if **duty_steps** is set to ‘**yes**’. At these points the complete particle distribution is either directly before or directly after a field boundary. (Duty steps at a particular field boundary can be disabled by the **duty** command in the **position** section.) The recursive step algorithm is forward looking: it uses steps that are shorter than allowed if this helps to avoid a very short step before the end point (defined in section **tracker**) or a duty point.



d) Iterative tracking: To integrate the equation of motion from on grid point (t_1) to the next (t_2) CSRtrack needs the self forces $\mathbf{f}(t)$ to all particles in the complete time interval $t_1 \leq t \leq t_2$. They are approximated by a linear interpolation between $\mathbf{f}_1 = \mathbf{f}(t_1)$ and $\mathbf{f}_2 = \mathbf{f}(t_2)$. As \mathbf{f}_2 depends on unknown phase space coordinates $\mathbf{x}_{ph,2} = \mathbf{x}_{ph}(t_2)$ they are estimated by $\mathbf{x}_{ph,2}^{(n)}$ and are improved to $\mathbf{x}_{ph,2}^{(n+1)}$ by iterative tracking. The first estimation $\mathbf{x}_{ph,2}^{(1)}$ is calculated with force $\mathbf{f}_2^{(0)} = \mathbf{0}$.



CSRtrack repeats the iterative tracking until the error criterion (set by **error_per_ct**) is fulfilled or the maximal number of iterations (set by **iterative**) is reached.



As the field computation for `force_type = projected` is very efficient, CSRtrack calculates $\mathbf{x}_{ph,2} = \mathbf{x}_{ph,2}^{(n)}$, $\mathbf{f}_2 = \mathbf{f}(t_2, \mathbf{x}_{ph,2}^{(n)}, \text{projected})$ for this type and uses $\mathbf{x}_{ph,2} = \mathbf{x}_{ph,2}^{(n)}$, $\mathbf{f}_2 = \mathbf{f}(t_2, \mathbf{x}_{ph,2}^{(n-1)}, \Lambda)$ otherwise. (The `force_type` is set by the command **type** in the **forces** section.)

e) Iterative tracking with precondition: The **forces** section provides several models for the calculation of self forces, e.g. **type = projected / csr_p_to_p / csr_g_to_p**. As the ‘projected’ model needs much less numerical effort than the other models, it could be helpful to use phase space coordinates $\mathbf{x}_{ph,2}$ that have been computed by iterative tracking with this method to improve the start estimation for other methods. This option is activated or deactivated by the command

precondition = yes / no

f) Error criterion: The accuracy parameter is set by the command

error_per_ct = err
error_weight_momentum = w_{mom}

The (relative) error criterion is

$$error(\mathbf{x}_{ph,2}^{(n)}, \mathbf{x}_{ph,2}^{(n+1)}) < (t_2 - t_1) \cdot err$$

$$error(\mathbf{x}_{ph,2}^{(n)}, \mathbf{x}_{ph,2}^{(n+1)}) = \sqrt{\frac{\langle (s^{(n+1)} - s^{(n)})^2 \rangle}{\langle (s^{(n+1)} - \langle s^{(n+1)} \rangle)^2 \rangle} + \frac{\langle (h^{(n+1)} - h^{(n)})^2 \rangle}{\langle (h^{(n+1)} - \langle h^{(n+1)} \rangle)^2 \rangle}} + w_{mom} \sqrt{\frac{\langle (p_s^{(n+1)} - p_s^{(n)})^2 \rangle}{\langle (p_s^{(n+1)} - \langle p_s^{(n+1)} \rangle)^2 \rangle}}$$

with s , h and p_s the longitudinal-, horizontal- and momentum offset of all particles with respect to the reference particle. The operator $\langle x \rangle$ averages phase space coordinates without weighting by particle charges.

g) Example:

```
track step{ct step min=0.02
           ct step max=0.20
           ct step first=0.20
           increase factor=2.0
           arc factor=0.3
           duty steps=yes

           iterative=2
           error per ct=0.001
           error weight momentum=0.1
           precondition=yes
}
```

The sum of the relative errors of all track steps from t_{part} (start time) to t_{end} is below $err \cdot (t_{end} - t_{part})$. For a chicane with a path length of about 5m the simulated time interval will be similar ($\cong 5$ m/c) . Therefore the sum of relative errors is below 0.005 for $err = 0.001$. Note that the parameter err controls errors due to iterative tracking and not the error related to the quality of the time grid.

3.4.5. Section: tracker

identifier	argument	unit / type
end_time_c0	value (t_{end})	time / number
delta_time_c0	value (dt_{end})	time / number
end_time_marker	marker name	character string
end_time_shift_c0	value (t_{shift})	time / number

The particle tracking is activated by the **tracker** section. There are three possibilities to define the time end (t_{end}) of the calculation: either directly by the command

end_time_c0 = t_{end}

or incremental by

delta_time_c0 = $dt_{end} = t_{end} - t_{part}$

or by reference to a field boundary with marker (see **position** section)

```

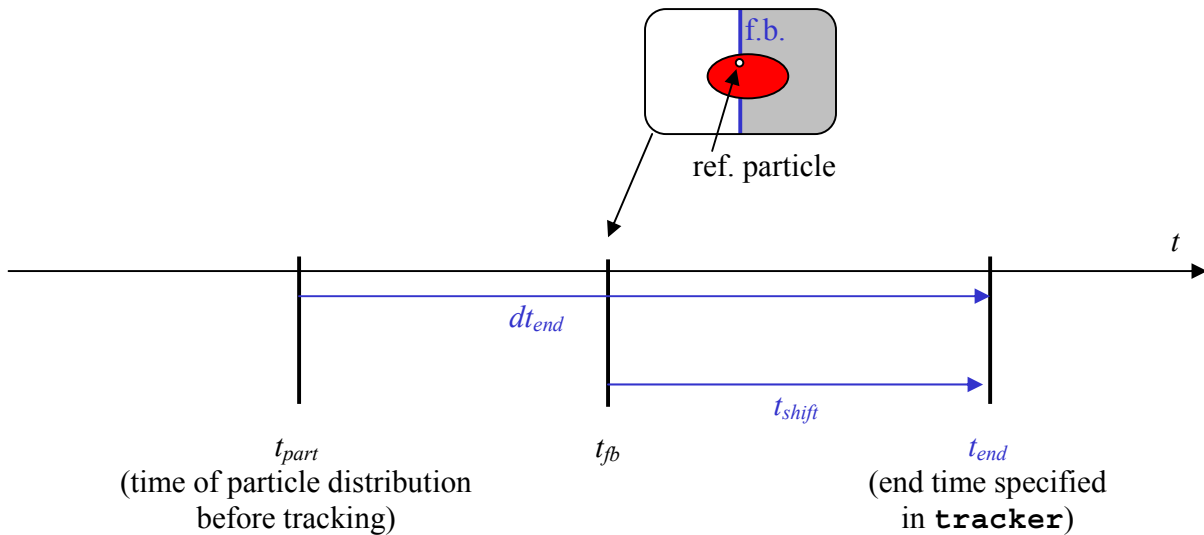
end_time_marker    = marker name
end_time_shift_c0 =  $t_{shift}$ 

```

The last option defines the time end relative to the time t_{fb} when the reference particle travels through the field boundary which is specified by marker name:

$$t_{end} = t_{fb} + t_{shift}$$

If the time grid is defined by an external file (see section **track_step**, command **time_grid_file**) the tracking is executed until the last grid value before or equal to the end time is reached. Otherwise the recursive time step algorithm generates time steps that end exactly at t_{end} . The **tracker** section can be called more than once. Before a consecutive call of the **tracker** section it is for example possible to modify **track_step** parameters or to call the **monitor** section.



Example:

```

tracker{end_time_marker=d4b,end_time_shift_c0=0.10}

```

3.4.6. Section: forces

identifier	argument	unit / type
type	none / projected / csr_p_to_p / csr_g_to_p / csr_p_to_m / csr_g_to_m	
shape	sphere / ellipsoid	

sigma_long	value ($\sigma_{ }$) / relative / file	length / number > 0
relative_long	value ($\sigma_{ rel}$)	number > 0
sigma_rad	value (σ_r) / file	length / number > 0
sigma_vert	value (σ_v) / relative / file	length / number > 0
relative_vert	value (σ_{vrel})	number > 0
sigma_file	file name	character string
shield		
shield_max		
use_old_mesh	yes / no	
test		
rintmi		

a) Concept: CSRtrack provides several models for the calculation of self-forces. The type of the model is specified by the command

type = none / projected / csr_p_to_p / csr_g_to_p

The parameter **none** causes particle tracking without self forces. If **type** is set to **projected**, CSRtrack uses a simple and very efficient model that neglects transverse dimensions of the source distribution, transverse forces, the transverse dependency of longitudinal forces as well as space charge effects. This model is based on a gaussian sub-bunch approach: all source particles are replaced by sub-bunches with corresponding strengths and longitudinal offsets. Therefore the three dimensional distribution of point particles is approximated by a smooth line charge density. The calculation of longitudinal fields neglects deformations of the *retarded* density function. The method '**csr_p_to_p**' replaces all source particles by three dimensional gaussian sub-bunches with individual strength and trajectory but with the same shape. It neglects vertical offsets and vertical particle motion. As all 'point' to point (or more precise sub-bunch to point) interactions have to be calculated, the numerical effort increases quadratically with the number of particles. The method '**csr_g_to_p**' is based on the same sub-bunch approach and the same 'point' to point interactions, but it uses a pseudo green's function for the field of a sub-bunch. Before each computation of 'point' to point interactions, the electromagnetic field is calculated on a mesh in the horizontal plane for a 'typical' sub-bunch. The trajectory of other sub-bunches can be fitted to the trajectory of the 'typical' sub-bunch by a coordinate transformation. The same transformation is used to calculate the electromagnetic fields of other sub-bunches from the meshed field.

b) Sub-bunch properties of **projected** force: This model uses one dimensional Gaussian sub-bunches. There are three possibilities to define their longitudinal rms length. Either the longitudinal size $\sigma_{||}$ is set directly by the command

```
sigma_long =  $\sigma_{||}$ 
```

or it is defined relative to the rms length of the particle distribution by

```
sigma_long    = relative  
relative_long =  $\sigma_{||rel}$ 
```

or it is set by an input file with

```
sigma_file = file name  
sigma_long = file
```

The last two possibilities allow to use sub-bunch dimensions that depend on time. The calculation of fields with time dependent sub-bunches neglects the change of the bunch length at retarded times. The use of bunch dimension files is described below.

c) Sub-bunch properties of **csr_p_to_p** and **csr_g_to_p** forces: This model uses Gaussian sub-bunches that are either spheres or ellipsoids:

```
shape = sphere / ellipsoid
```

The radius of spherical sub-bunches is defined in the same way as the longitudinal dimension of sub-bunches of the **projected** force. Ellipsoidal bunches have three rms parameters: the longitudinal size $\sigma_{||}$, the horizontal (or radial) size σ_r and the vertical size σ_v . $\sigma_{||}$ and σ_v are either set directly, or relative to the corresponding rms dimension of the particle distribution, or by an input file with sub-bunch dimensions. σ_r is either defined directly or by file. The calculation of fields with time dependent sub-bunches neglects the change of the bunch length at retarded times. The use of bunch dimension files is described below.

d) Sub-bunch dimension file: Sub-bunch sizes can be defined directly by file. The file name is set with the command

```
sigma_file = file name
```

CSRtrack expects an ascii input file with the specified name in the input- or root-directory (see **io_path**). It reads a list of values for time and dimensions (units = time, length, length, length):

```
 $t_1$    $\sigma_{||1}$    $\sigma_r1$    $\sigma_v1$   
 $t_2$    $\sigma_{||2}$    $\sigma_r2$    $\sigma_v2$   
 $t_3$    $\sigma_{||3}$    $\sigma_r3$    $\sigma_v3$   
 $t_4$    $\sigma_{||4}$    $\sigma_r4$    $\sigma_v4$   
...  
 $t_n$    $\sigma_{||n}$    $\sigma_{rn}$    $\sigma_{vn}$ 
```

(Each input line has to start with four numbers, the rest is ignored.) The sub-bunch dimensions are calculated by linear interpolation along the time axis (see **track_step**, time grid). The interpolated longitudinal dimension is used if the argument **file** is assigned to **sigma_long**. Corresponding assignments can be used for the transverse dimensions. The output file created by

the steps-monitor (see `online_monitor`, `type=steps`) can be used as sub-bunch dimension file, e.g. to perform calculations with exactly the same dimensions as in an earlier run.

e) Example:

```
forces{type=csr g to p,
       shape=ellipsoid
       sigma file=my file.dat
       sigma long=file
       sigma rad =0.0003
       sigma vert=relative,relative vert=1.0
}
```

3.5. Version 1.0 ---> Version 1.1

```
forces{type = ...,
       sbox = ..., rbox=...,
       use old mesh = ...,
       shield= ..., shield_max=... }
```

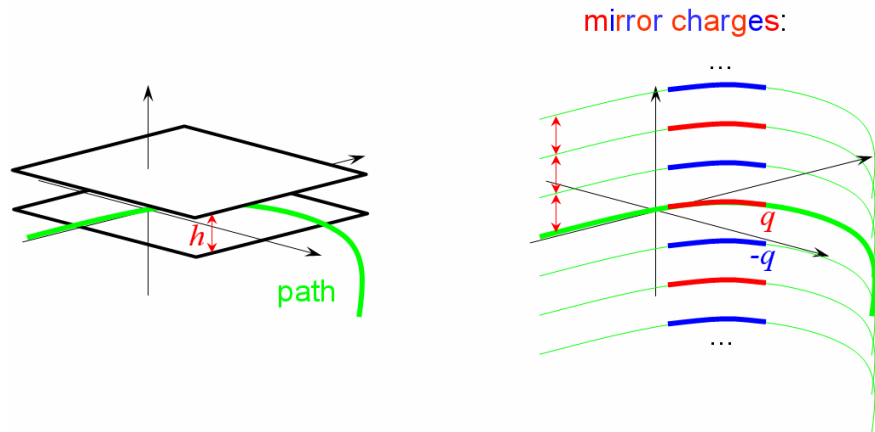
the new types are: `csr_p_to_m`, `csr_g_to_m`; they have the same effect as `csr_p_to_p`, `csr_g_to_p` but the forces are calculated from meshed electromagnetic field values; `sbox` and `rbox` can be used to modify the longitudinal and radial step width of the (equidistant) mesh; the step widths are: $\Delta s = \mathbf{sbox} \cdot \sigma_{||}$, $\Delta r = \mathbf{rbox} \cdot \sigma_r$ with $\sigma_{||}$, σ_r the longitudinal and radial rms dimension of the sub-bunches; the default values are `sbox = 0.5`, `rbox=0.5`; mesh field values are used only for cells with at least four particles (observers) per cell; otherwise the field values are calculated directly; additional output per calculation step:

```
ctstep= 0.03000 -> ct= 10.77925
*** step_error/step_time= 4.98611017E-05
boxes = 1961 e_points= 5027 p_points= 7886
step_error/step_time= 0.00198496122
boxes = 1960 e_points= 5026 p_points= 7887
step_error/step_time= 6.39205063E-07
ctstep= 0.02581 -> ct= 10.80506
*** step_error/step_time= 5.22485114E-05
boxes = 1992 e_points= 5070 p_points= 7712
step_error/step_time= 0.00203230079
```

‘`boxes`’ is the number of mesh cells (with at least four particles) that are calculated; ‘`e_points`’ is the number of edge points of these cells; ‘`p_points`’ is the number of particles (observers) that are calculated directly;

calculation of the meshed Green’s function (`csr_g_to_p`, `csr_g_to_m`) during iterative tracking: sometimes it is possible to reuse the Green’s function for successive tracking iterations (e.g. if sufficient memory space is available to store the function on the complete mesh); the command `use_old_mesh = yes/no` can be used to enable/disable this possibility;

the commands `shield= ...` and `shield_max=...` are used to specify shielding by perfect conducting horizontal planes; `shield` defines the distance h between the conducting planes, `shield_max` defines the maximal vertical distance to which mirror charges are taken into account;



work in progress: the mesh of the Green's function is not now optimized for wave fronts that are created by mirror charges; therefore the global mesh density is increased; this solution is not optimal concerning mesh resolution and computational efficiency;

attention: shielding causes a dispersive propagation of electromagnetic fields; a time grid (see `track_step`) that works for calculations without shielding is not necessarily sufficient with shielding (and otherwise); it is recommended to observe the particle coordinates and forces at few test particles for all time steps (see `online_monitor, type = phase, particle = m`) and to adjust the grid if required;

```
track_step{precondition=yes}
```

different preconditioning methods are used for different force types (`forces, type`):

Sections: `monitor` and `online_monitor`

a) Concept: The `monitor` section is used to write particles properties and forces before or after a tracking calculation to file while monitors defined in the `online_monitor` section are served during the tracking. As fields and forces are calculated during the tracking, which is activated in the `tracker` section, they are undefined before the `tracker` section has been called.

b) `monitor` section:

identifier	argument	unit / type
<code>name</code>	file name	character string
<code>format</code>	<code>fmt1 / fmt2 / fmt3</code>	

The name of the output file is defined by `name = file name`. The file formats are described below.

c) `online_monitor` section:

identifier	argument	unit / type
<code>name</code>	file name	character string
<code>type</code>	<code>phase / subbunch / steps</code>	
<code>format</code>	<code>fmt1 / fmt2 / fmt3</code>	
<code>particle</code>	value (m)	integer
<code>start_time_c0</code>	value (t_{start}) / <code>now</code>	time / number
<code>start_time_marker</code>	marker name	character string
<code>start_time_shift_c0</code>	value (dt_{start})	time / number
<code>end_time_c0</code>	value (t_{end})	time / number
<code>end_time_marker</code>	marker name	character string
<code>end_time_shift</code>	value (dt_{end})	time / number
<code>time_step_c0</code>	value (t_{step}) / <code>all</code>	time / number

There are three different monitor types:

`type = phase / subbunch / steps`

The type `phase` is used to monitor properties of an individual particle or of the complete distribution.

d) Monitor Events: Online monitors are served during the tracking each time when a monitor event is fulfilled. In principle monitor events are defined by the start time t_{start} , end time t_{end} and time step t_{step} , but the so defined monitor grid does usually not coincide with the `time grid` defined in the section `track_step`. Therefore monitor events are related always to the next following point on the time grid. t_{start} and t_{end} can be specified either directly by

```
start_time_c0 =  $t_{start}$  / now
end_time_c0   =  $t_{end}$ 
```

or by reference to a field boundary with marker:

```
start_time_marker = marker name
start_time_shift_c0 =  $dt_{start}$ 
end_time_marker   = marker name
end_time_shift_c0 =  $dt_{end}$ 
```

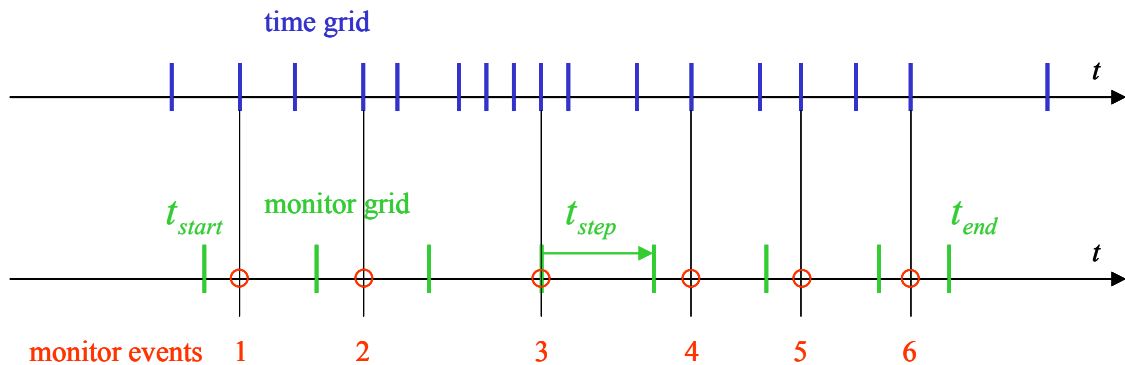
The second possibility defines the time relative to the time t_{fb} when the reference particle travels through the field boundary which is specified by marker name:

$$t_{start} = t_{fb} + dt_{start} \quad \text{and/or} \quad t_{end} = t_{fb} + dt_{end}$$

The time step is defined by

$$\text{time_step_c0} = t_{\text{step}} / \text{all}$$

The argument **all** has the effect that all points on the time grid are monitored.



e) Name convention for output files: Phase monitors of the full particle distribution (**type=phase, particle=all**) create for each monitor event an individual file. The file name is derived from the argument of the **name** command and the event number as follows:

name=<string>.<extension>

1st monitor event → file name = **<string>0001.<extension>**
 2nd monitor event → file name = **<string>0002.<extension>**
 3rd monitor event → file name = **<string>0003.<extension>**

...

The numbering of monitor events depends on the definition of t_{start} and t_{step} that can be defined individually for different monitors. Therefore events with the same number that have been recorded by different monitors do not necessarily coincide.

f) Example:

```
online monitor{name=x.fmt3,type=phase,format=fmt3,particle=all
  start time c0=now
  end time marker=d4b,end time shift c0=1.0
  time step c0=0.02
}
```

3.5.1.1. Particle monitors

a) **format = fmt1, particle = all**: This format is compatible to the corresponding input format of the **particles** section. The output file can be used as input file for a later (restarted) calculation. Each line of the output file is written with the fortran format (7(1x,e22.15)). The last six numbers in the first line ($r_1, r_2, r_3, r_4, r_5, r_6$) are set to zero.

b) **format = fmt2, particle = all**: This format is compatible to the corresponding input format of the **particles** section. The output file can be used as input file for a later

(restarted) calculation. Each line of the output file is written with the fortran format (7(1x,e22.15)). The last six numbers in the first line ($r_1, r_2, r_3, r_4, r_5, r_6$) are set to zero.

c) **format = fmt3, particle = all**: Each line of the output file is written with the fortran format (10(1x,e22.15),2(1x,i1)). The output file has the following structure:

```

t   γr  0  0  0  0  0  σs  σr  σz  0  0
hr h'r vr v'r 0  δer q1 fs,1 fh,1 fv,1 Ls,1 Lt,1
h2 h'2 v2 v'2 s2 δe2 q2 fs,2 fh,2 fv,2 Ls,2 Lt,2
h3 h'3 v3 v'3 s3 δe3 q3 fs,3 fh,3 fv,3 Ls,3 Lt,3
Λ
hn h'n vn v'n sn δen qn fs,n fh,n fv,n Ls,n Lt,n

```

t is the time of the distribution. γ is the Lorentz factor γ that corresponds to the reference momentum (see **particles** section). σ_s , σ_r , and σ_z are the longitudinal, radial and vertical rms sub-bunch dimensions that were used for the force calculation (see **forces** section). The particle coordinates (h_i, v_i, s_i) and slopes (h'_i, v'_i) are defined relative to the reference trajectory and the reference particle: (h_i, h'_i) are horizontal parameters, (v_i, v'_i) are the offset and slope in vertical- or z-direction and s_i is the path-length difference with respect to the reference particle. δe_i is the relative energy deviation normalized to the energy that corresponds to the reference momentum. ($f_{s,i}, f_{h,i}, f_{v,i}$) are the force components with:

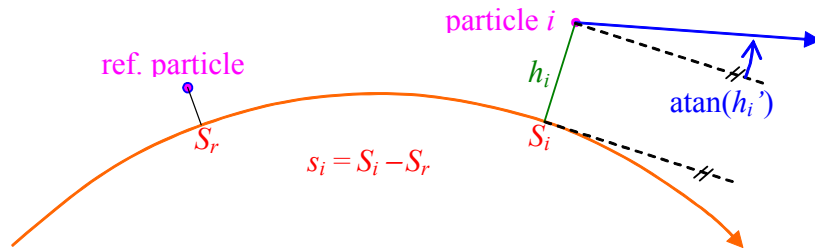
$$\mathbf{u}_{s,i} = \mathbf{p}_i / p_i$$

$$\mathbf{u}_{h,i} = \mathbf{u}_z \times \mathbf{u}_{s,i}$$

$$\mathbf{u}_{v,i} = \mathbf{u}_{s,i} \times \mathbf{u}_{h,i}$$

$$\mathbf{f}_i = f_{s,i} \mathbf{u}_{s,i} + f_{h,i} \mathbf{u}_{h,i} + f_{v,i} \mathbf{u}_{v,i}$$

In the present version of CSRtrack there is no input format available that allows to set or reset the source and test flags ($L_{s,i}, L_{t,i}$). Therefore they are always set (=1). The source flag determines if a certain particle contributes to the generation of self-fields or not. A particle is tracked with self and external forces if the test flag is set, otherwise only external fields affect the motion of this particle.



d) **format = fmt1, particle = m**: For each monitor event, a new line with the actual phase space information of particle m is added to the output file. The file is of the following structure:

$$\begin{array}{cccccccc}
 t_1 & \delta x_m^{(t1)} & \delta y_m^{(t1)} & \delta z_m^{(t1)} & \delta p x_m^{(t1)} & \delta p y_m^{(t1)} & \delta p z_m^{(t1)} & q_m \\
 t_2 & \delta x_m^{(t2)} & \delta y_m^{(t2)} & \delta z_m^{(t2)} & \delta p x_m^{(t2)} & \delta p y_m^{(t2)} & \delta p z_m^{(t2)} & q_m \\
 t_3 & \delta x_m^{(t3)} & \delta y_m^{(t3)} & \delta z_m^{(t3)} & \delta p x_m^{(t3)} & \delta p y_m^{(t3)} & \delta p z_m^{(t3)} & q_m \\
 \Lambda
 \end{array}$$

The first column ($t_1, t_2, t_3 \dots$) is the time of the monitor event (unit = time). The next seven rows describe the position, momentum and charge in the same way as for the corresponding input format (**fmt1**, x_p, y_p, z_p -coordinates). This means the coordinates of the reference particle ($m = 1$) are absolute, the coordinates of all other particles ($m > 1$) are relative to the reference particle.

e) **format = fmt2, particle = m**: For each monitor event, a new line with the actual phase space information of particle m is added to the output file. The file is of the following structure:

$$\begin{array}{cccccccc}
 t_1 & \delta s_m^{(t1)} & \delta h_m^{(t1)} & \delta v_m^{(t1)} & \delta p s_m^{(t1)} & \delta p h_m^{(t1)} & \delta p v_m^{(t1)} & q_m \\
 t_2 & \delta s_m^{(t2)} & \delta h_m^{(t2)} & \delta v_m^{(t2)} & \delta p s_m^{(t2)} & \delta p h_m^{(t2)} & \delta p v_m^{(t2)} & q_m \\
 t_3 & \delta s_m^{(t3)} & \delta h_m^{(t3)} & \delta v_m^{(t3)} & \delta p s_m^{(t3)} & \delta p h_m^{(t3)} & \delta p v_m^{(t3)} & q_m \\
 \Lambda
 \end{array}$$

The first column ($t_1, t_2, t_3 \dots$) is the time of the monitor event (unit = time). The next seven rows describe the position, momentum and charge in the same way as for the corresponding input format (**fmt2**). This means the coordinates of the reference particle ($m = 1$) are absolute, the coordinates of all other particles ($m > 1$) are relative to the reference particle ($\mathbf{u}_s, \mathbf{u}_v, \mathbf{u}_h$ -base).

f) **format = fmt3, particle = m**: For each monitor event, a new line with the actual phase space coordinates and forces of particle m is added to the output file. The file is of the following structure:

$$\begin{array}{cccccccccccccc}
 t_1 & h_m^{(t1)} & h'_m{}^{(t1)} & v_m^{(t1)} & v'_m{}^{(t1)} & s_m^{(t1)} & e_m^{(t1)} & q_m & f_{s,m}^{(t1)} & f_{h,m}^{(t1)} & f_{v,m}^{(t1)} & L_{s,m} & L_{t,m} \\
 t_2 & h_m^{(t2)} & h'_m{}^{(t2)} & v_m^{(t2)} & v'_m{}^{(t2)} & s_m^{(t2)} & e_m^{(t2)} & q_m & f_{s,m}^{(t2)} & f_{h,m}^{(t2)} & f_{v,m}^{(t2)} & L_{s,m} & L_{t,m} \\
 t_3 & h_m^{(t3)} & h'_m{}^{(t3)} & v_m^{(t3)} & v'_m{}^{(t3)} & s_m^{(t3)} & e_m^{(t3)} & q_m & f_{s,m}^{(t3)} & f_{h,m}^{(t3)} & f_{v,m}^{(t3)} & L_{s,m} & L_{t,m} \\
 \Lambda
 \end{array}$$

The first column ($t_1, t_2, t_3 \dots$) is the time of the monitor event (unit = time). The next eleven rows describe the position, momentum, charge and force in the same way as for the corresponding input format (**fmt3**).

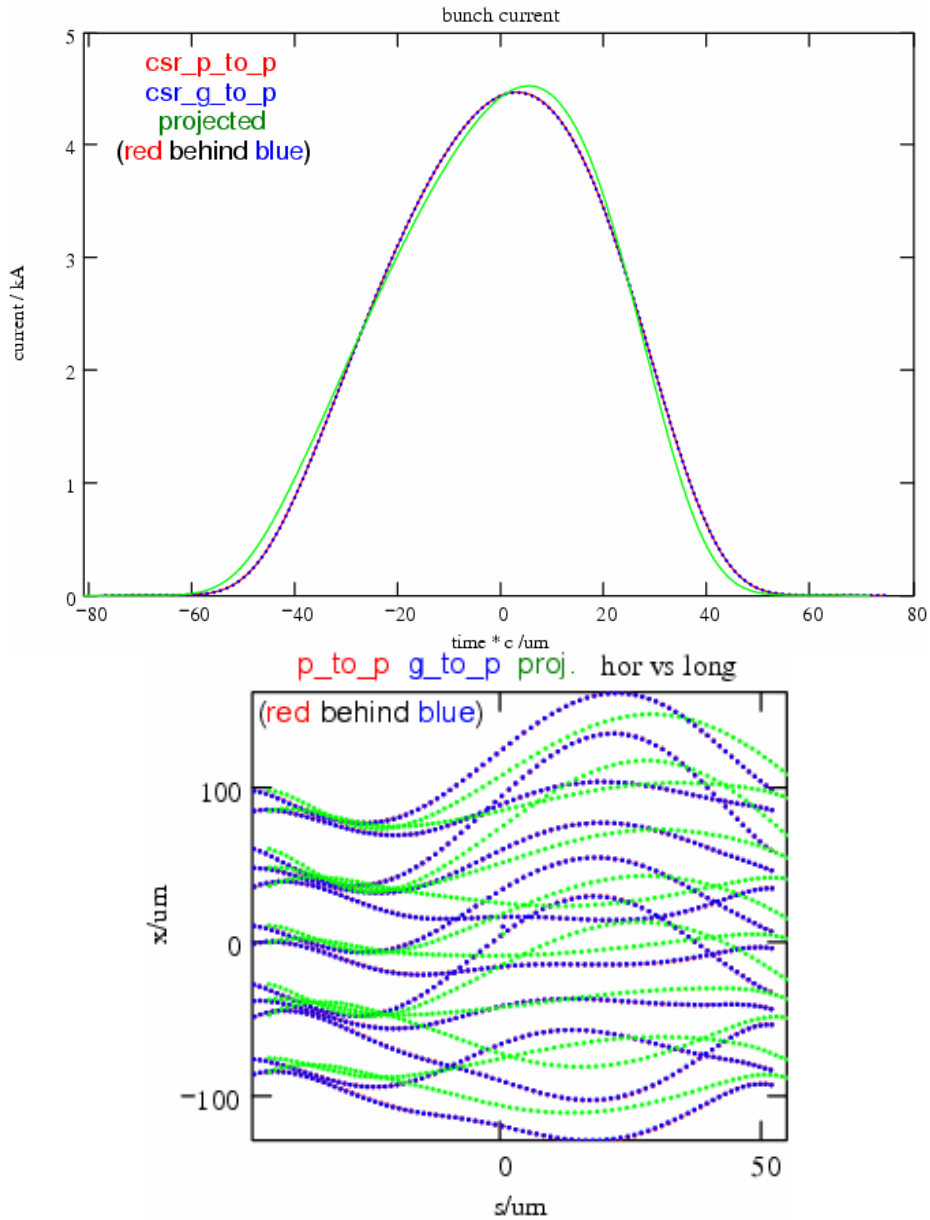
3.5.1.2. online_monitor: type = subbunch

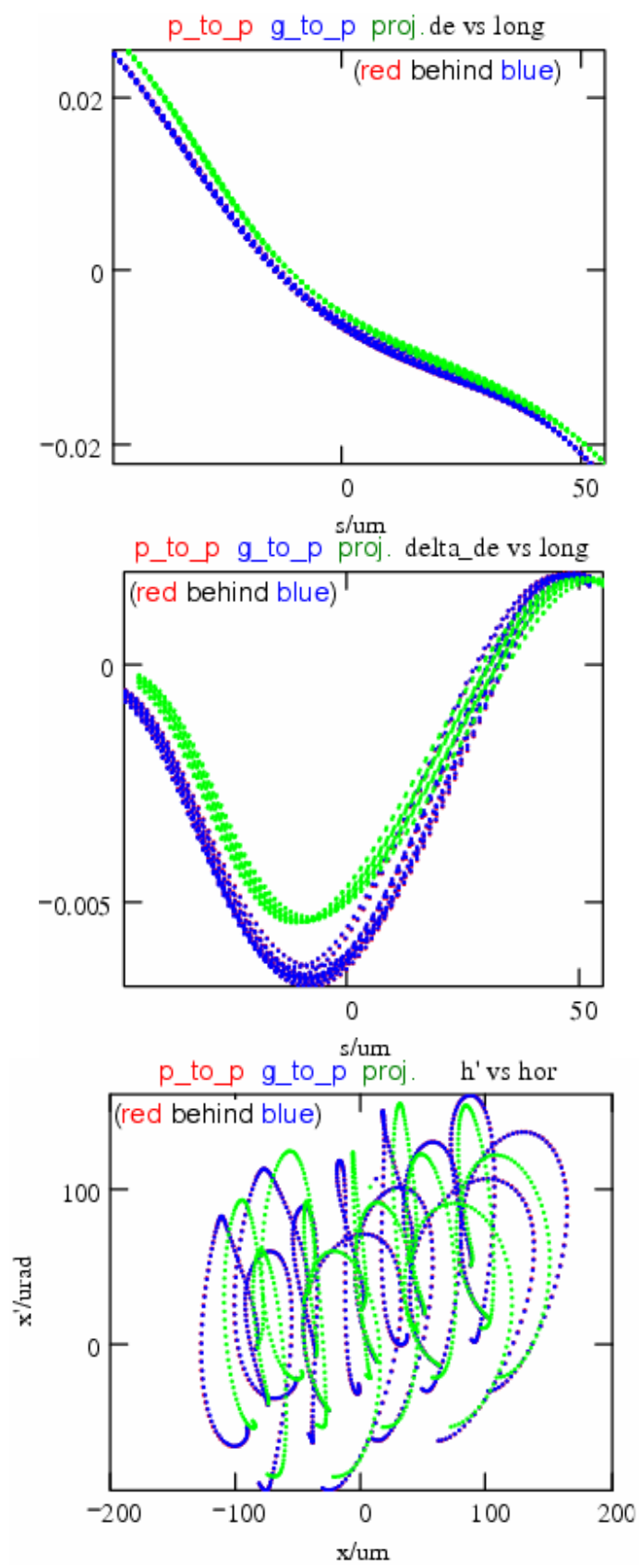
3.5.1.3. online_monitor: type = steps

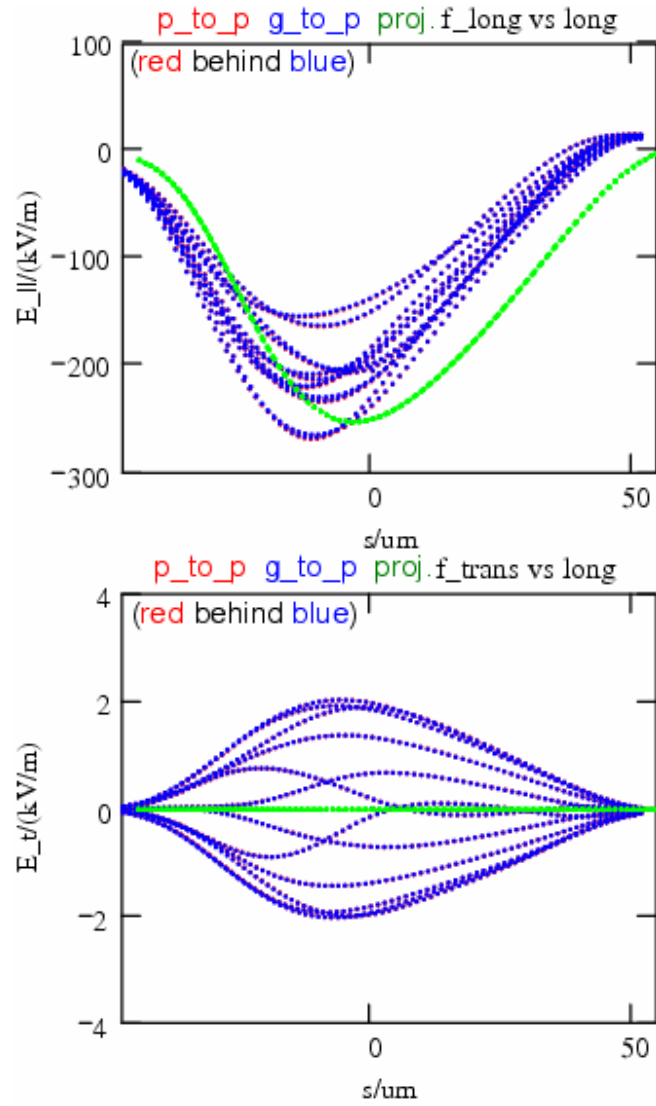
Appendix

3.6. Plotted Results for Example Case

Comparison of calculations with force types **p_to_p**, **g_to_p** and **projected**. The data points for force type **p_to_p** are plotted in red, for **g_to_p** in blue and for **projected** in green. Usually the red points are hidden by the blue ones.







3.7. Units

dimension	normalization
length	1 m
time	1 m / c_0
momentum	1 eV / c_0
charge	1 C
angle	1 deg

4. Bibliography

- [1] E. Saldin, E.Schneidmiller, M.Yurkov: Radiative Interaction of Electrons in a Bunch Moving in an Undulator. NIM A417 (1998) 158-168.
- [2] M. Dohlus: Two Methods for the Calculation of CSR Fields. TESLA-FEL-2003-05
- [3] M. Dohlus, A. Kabel, T. Limberg: Efficient Field Calculation of 3D Bunches on General Trajectories. NIM A445 (2000) 338-342.