The OPALX Universe

User guide, physics manual, reference, and developer handbook

1 OPALX Universe

Main Developers & Institutions

Andreas Adelmann (PSI); Pedro Calvo (CIEMAT); Achim Gsell (PSI); Sriramkrishnan Muralikrishnan (JSC); Philippe Piot (NIU); Chris Rogers (RAL); Mohsen Sadr (PSI/MIT); Jochem Snuverink (PSI); Jonathan Thompson (RAL); Daniel Winklehner (MIT); Yuanjie Bi (Sun Yat-sen University); John Biddiscombe (CSCS/ETH)

Simulate charged-particle accelerators across CPUs and GPUs

OPALX is the performance-portable accelerator simulation framework for high-statistics, three-dimensional beam dynamics. This manual brings the user guide, physics models, command reference, and developer handbook into one searchable source.

Install OPALX Start a simulation

1.1 Find your path

Start here

1.1.1 Getting Started

Build OPALX, choose a platform backend, and run a first input file.

Get started

Run simulations

1.1.2 User Guide

Learn the input language, beam definitions, runtime options, tracking workflow, elements, reusable structures, field solvers, and outputs.

Read the user guide

Understand the models

1.1.3 Physics

Follow coordinate conventions, equations of motion, element fields, collective effects, decay, spin, and photon physics.

Explore the physics

Look something up

1.1.4 Reference

See the commands and elements registered by the current executable, build configuration, file formats, and external API documentation.

Open the reference

Contribute

1.1.5 Developer Guide

Understand the architecture, build and test workflows, and how to add features without losing GPU portability.

Read the developer guide

Resolve a problem

1.1.6 Troubleshooting

Diagnose build and runtime failures, prepare reproducible bug reports, and check known limitations.

Troubleshoot OPALX

Live resources OPALX API documentation Regression-test dashboard OPALX source

1.2 Documentation status

This redesign is an actively curated foundation. Pages marked **current** have been checked against the present OPALX source. **Experimental** pages describe implemented work whose interface may still change. **Planned** pages make documentation gaps visible instead of filling them with unverified OPAL text.

1.3 Acknowledgments and citation

OPALX builds on the original OPAL project and is developed across PSI and its international accelerator-science partners. Use the publications and citation guidance when citing the software. The original OPAL manual remains available as a separate historical resource.

Part I

Getting Started

2 Install and Build OPALX

OPALX uses CMake and Kokkos to select the execution backend. The following toolchain is recommended for current development and production builds:

Dependency	Recommended version	Notes
GCC	13	Use the same host compiler throughout the dependency stack.
OpenMPI	4 or 5	Either major version is suitable; use the MPI wrappers from the selected installation.
CUDA	12.9	Required only for NVIDIA GPU builds. Match the architecture setting to the target GPU.
CMake	3.24 or newer	OPALX currently declares CMake 3.24 as its minimum version.
Git	Recent release	Used to clone OPALX and fetched dependencies.

These are recommended versions, not a promise that every other combination is unsupported. Accelerator builds need the matching CUDA or HIP toolchain. Other core dependencies are obtained by the build unless installed-dependency options are selected.

On a module-based system, a typical environment is:

```
module load gcc/13
module load openmpi/5  # openmpi/4 is also supported
module load cuda/12.9  # CUDA builds only
```

Module names vary between clusters. After loading them, check that `gcc`, `mpicxx`, and `nvcc` resolve to the intended installations before configuring the build.

2.1 Clone the source

```
git clone https://github.com/OPALX-project/OPALX.git
cd OPALX
```

2.2 Configure a CPU build

For a portable single-threaded backend:

```
cmake -S . -B build_serial \
  -DBUILD_TYPE=Release \
  -DPLATFORMS=SERIAL
cmake --build build_serial -j 8
```

For OpenMP, use a separate build directory:

```
cmake -S . -B build_openmp \
  -DBUILD_TYPE=Release \
  -DPLATFORMS=OPENMP
cmake --build build_openmp -j 8
```

2.3 Configure a GPU build

An NVIDIA A100 build uses the CUDA backend and Ampere architecture:

```
cmake -S . -B build_cuda \
  -DBUILD_TYPE=Release \
  -DPLATFORMS=CUDA \
  -DARCH=AMPERE80
cmake --build build_cuda -j 8
```

Select the architecture that matches the target GPU. HIP builds use `-DPLATFORMS=HIP` and the corresponding AMD architecture.

2.4 Enable tests

Add `-DOPALX_ENABLE_UNIT_TESTS=ON` when configuring, then run:

```
ctest --test-dir build_serial --output-on-failure
```

See Build, Test, and CI for the full developer workflow and Configuration Reference for the maintained build options.

3 Cluster-Specific Setup

Cluster modules, queues, accounts, and scheduler policies change independently of OPALX. Treat the examples here as starting points: confirm the current site documentation and replace paths, input files, time limits, and resource counts before submitting a job.

3.1 Common Slurm workflow

1. Load one consistent compiler, MPI, and accelerator-toolkit stack.
2. Configure a new build directory for that stack and target architecture.
3. Build on a login or build node according to local policy.
4. Submit simulations through Slurm rather than running them on a login node.
5. For GPU jobs, normally allocate one MPI rank per GPU and pass `--kokkos-map-device-id-by=mpi_rank` to OPALX.

The recommended baseline is GCC 13, OpenMPI 4 or 5, and CUDA 12.9. Module names are site-specific; see Install and Build OPALX for the general toolchain requirements.

3.2 PSI Merlin GPU

The Merlin GPU example targets Pascal hardware:

```
cmake -S . -B build_merlin_cuda \
  -DBUILD_TYPE=Release \
  -DPLATFORMS=CUDA \
  -DARCH=PASCAL61
cmake --build build_merlin_cuda -j 8
```

A minimal two-GPU Slurm script is:

```
#!/bin/bash
#SBATCH --job-name=opalx-merlin
#SBATCH --output=opalx-merlin-%j.out
#SBATCH --error=opalx-merlin-%j.err
#SBATCH --time=00:10:00
#SBATCH --nodes=1
#SBATCH --ntasks=2
#SBATCH --clusters=gmerlin6
#SBATCH --partition=gpu-short
#SBATCH --account=merlin
#SBATCH --gpus=2

ulimit -c unlimited

srun /path/to/build_merlin_cuda/src/opalx /path/to/example.in \
  --info 10 --kokkos-map-device-id-by=mpi_rank
```

3.3 PSI Gwendolen GPU

Gwendolen uses the CUDA backend with the Ampere A100 target:

```
cmake -S . -B build_gwendolen_cuda \
  -DBUILD_TYPE=Release \
  -DPLATFORMS=CUDA \
  -DARCH=AMPERE80
cmake --build build_gwendolen_cuda -j 8
```

The corresponding two-GPU Slurm starting point is:

```
#!/bin/bash
#SBATCH --job-name=opalx-gwendolen
#SBATCH --output=opalx-gwendolen-%j.out
#SBATCH --error=opalx-gwendolen-%j.err
#SBATCH --time=00:10:00
#SBATCH --nodes=1
#SBATCH --ntasks=2
#SBATCH --clusters=gmerlin6
#SBATCH --partition=gwendolen
#SBATCH --account=gwendolen
#SBATCH --gpus=2

ulimit -c unlimited

srun /path/to/build_gwendolen_cuda/src/opalx /path/to/example.in \
  --info 10 --kokkos-map-device-id-by=mpi_rank
```

3.4 Add another cluster

New entries should record the toolchain modules, OPALX backend and architecture, configure command, scheduler resource model, MPI-to-device mapping, and a small verified launch script. As this section grows, each cluster can move to its own page under `getting-started/clusters/` without mixing site-specific commands into the general installation instructions.

The IPPL Slurm guide contains additional PSI and LUMI examples that can inform future entries.

4 Run a First Simulation

Start with an input file from the maintained OPALX regression suite. Those inputs are executable specifications and are checked continuously against the current parser and physics implementation.

1. Choose a small serial regression case.
2. Copy its `.in` file and required supporting files into a new working directory.
3. Run the executable from that directory.

```
mpirun -np 1 /path/to/build_serial/src/opalx example.in --info 2
```

For a multi-GPU run, map each MPI rank to a device:

```
srun /path/to/build_cuda/src/opalx example.in \
  --info 2 --kokkos-map-device-id-by=mpi_rank
```

4.1 What to check

- The startup banner identifies the build and execution backend.
- Parser errors name the statement and location that could not be read.
- Statistics and particle outputs depend on the input-file options and output commands.
- A nonzero exit status means the run did not complete successfully.

The regression-test dashboard shows which inputs currently pass. Input-file compatibility is still being cataloged, so a regression input is safer than copying an example from the historical OPAL manual. Continue with the three complete Worked Input Files for progressively more detailed starting points.

5 Worked Input Files

The examples below are complete input files: no declarations are omitted from their code blocks. They reduce maintained regression configurations to one idea at a time. Copy a complete block into a file ending in `.in` and run it as shown in Run a First Simulation.

5.1 Drift without self-fields

This is the smallest useful particle-tracking example. It derives the reference momentum `P0` from an electron kinetic energy `Edes`, samples a Gaussian bunch, and tracks it through a 0.5 m drift with `FIELDSOLVER.TYPE=NONE`.

```
OPTION, PSDUMPFREQ=1000;
OPTION, STATDUMPFREQ=1;
OPTION, AUTOPHASE=0;
OPTION, VERSION=10900;

TITLE, STRING="Minimal drift without self-fields";

REAL n_particles=1000;
REAL bunch_charge=1e-12;

// EMASS, Edes, and P0 are in GeV-based OPAL units.
REAL Edes=1e-3;
REAL gamma=(Edes+EMASS)/EMASS;
REAL beta=sqrt(1.0-1.0/(gamma*gamma));
REAL P0=gamma*beta*EMASS;

VALUE, {Edes, P0};

D1: DRIFT, L=0.5, ELEMEDGE=0.0;
DriftLine: LINE=(D1);

FSNone: FIELDSOLVER, TYPE=NONE,
    NX=8, NY=8, NZ=8,
    PARFFTX=TRUE, PARFFTY=TRUE, PARFFTZ=TRUE,
    BCFFTX=OPEN, BCFFTY=OPEN, BCFFTZ=OPEN;
```

```
Dist: DISTRIBUTION, TYPE=GAUSS,
    NPARTDIST=n_particles,
    SIGMAX=2e-4, SIGMAY=2e-4, SIGMAZ=2e-4,
    SIGMAPX=1e-5, SIGMAPY=1e-5, SIGMAPZ=1e-5;

Source: EMISSIONSOURCE, DISTRIBUTION=Dist;
Sources: EMISSIONSOURCELIST=(Source);

Beam1: BEAM, PARTICLE=ELECTRON,
    PC=P0, NALLOC=n_particles,
    BCHARGE=bunch_charge, CHARGE=-1,
    SOURCES=Sources;

TRACK, LINE=DriftLine, BEAM=Beam1,
    DT={1e-11}, MAXSTEPS={1000}, ZSTOP={0.5};
  RUN, METHOD=PARALLEL, FIELDSOLVER=FSNone;
ENDTRACK;

QUIT;
```

`PC=P0` gives the beam and non-file distribution their reference momentum. The sampled momentum offsets are expressed in normalized momentum $p/(mc)$, while `P0` itself is in GeV/c. `ZSTOP=0.5` ends tracking when the reference trajectory reaches the end of the drift; `MAXSTEPS` is the safety limit. Although `NONE` performs no mesh solve, the current constructor still requires mesh dimensions, uniform boundary values, and all three `PARFFT*` flags.

5.2 Autophased RF cavity

This example is based on the maintained `RFCavity` regression. It needs the DriveGun.T7 field map in a local `fieldmaps/` directory.

```
OPTION, PSDUMPFREQ=1000;
OPTION, STATDUMPFREQ=1;
OPTION, AUTOPHASE=4;
OPTION, VERSION=10900;

TITLE, STRING="Autophased standing-wave cavity";

REAL n_particles=1000;
REAL bunch_charge=1e-12;
REAL rf_frequency=1300.0; // MHz

REAL Edes=1.4e-9;
```

```
REAL gamma=(Edes+EMASS)/EMASS;
REAL beta=sqrt(1.0-1.0/(gamma*gamma));
REAL P0=gamma*beta*EMASS;

VALUE, {Edes, P0};

Gun: RFCAVITY,
    L=0.2927, ELEMEDGE=0.0,
    TYPE="STANDING",
    VOLT=60.0,
    FREQ=rf_frequency,
    LAG=0.0,
    FMAPFN="fieldmaps/DriveGun.T7",
    APVETO=FALSE;

D1: DRIFT, L=0.2073, ELEMEDGE=0.2927;
GunLine: LINE=(Gun, D1);

FSNone: FIELDSOLVER, TYPE=NONE,
    NX=8, NY=8, NZ=8,
    PARFFTX=TRUE, PARFFTY=TRUE, PARFFTZ=TRUE,
    BCFFTX=OPEN, BCFFTY=OPEN, BCFFTZ=OPEN;

Dist: DISTRIBUTION, TYPE=GAUSS,
    NPARTDIST=n_particles,
    SIGMAX=1e-4, SIGMAY=1e-4, SIGMAZ=1e-5,
    SIGMAPX=1e-6, SIGMAPY=1e-6, SIGMAPZ=1e-6;

Source: EMISSIONSOURCE, DISTRIBUTION=Dist, R0Z=1e-5;
Sources: EMISSIONSOURCELIST=(Source);

Beam1: BEAM, PARTICLE=ELECTRON,
    PC=P0, NALLOC=n_particles,
    BCHARGE=bunch_charge, CHARGE=-1,
    SOURCES=Sources;

TRACK, LINE=GunLine, BEAM=Beam1,
    DT={1e-12}, MAXSTEPS={2000}, ZSTOP={0.5};
  RUN, METHOD=PARALLEL, FIELDSOLVER=FSNone;
ENDTRACK;

QUIT;
```

`RFCAVITY` multiplies the field-map fields by the configured RF amplitude and phase. With `OPTION.AUTOPHASE=4`, OPALX tracks the reference particle through each `RFCAVITY` or

TRAVELINGWAVE, searches for the phase giving maximum exit energy, and performs four refinement passes around that maximum. The optimized phase is added to the nominal LAG, stored for the run, and reported in Gun_AP.dat. Set AUTOPHASE=0 globally or APVETO=TRUE on this cavity to retain the supplied phase without the search.

The field map also contains an RF frequency. If it differs from FREQ by more than one percent, OPALX warns and uses the map frequency.

5.3 Emission with binned open-boundary space charge

This AWA-gun-style example uses no external data. A short pulse is emitted at a grounded plane into one CONSTANTEFIELDCAVITY. The field solver uses adaptive velocity bins, the open Hockney backend, and the shifted-Green Dirichlet correction.

```
OPTION, PSDUMPFREQ=1000;
OPTION, STATDUMPFREQ=10;
OPTION, AUTOPHASE=0;
OPTION, VERSION=10900;

TITLE, STRING="Minimal emitted bunch with binned space charge";

REAL n_particles=4096;
REAL bunch_charge=1e-10;
REAL Edes=1e-9;
REAL gamma=(Edes+EMASS)/EMASS;
REAL beta=sqrt(1.0-1.0/(gamma*gamma));
REAL P0=gamma*beta*EMASS;

VALUE, {Edes, P0};

Gun: CONSTANTEFIELDCAVITY,
    L=0.5, ELEMEDGE=0.0,
    EX=0.0, EY=0.0, EZ=-5.0;
GunLine: LINE=(Gun);

Bins: BINNING,
    MAXBINS=64,
    DESIREDWIDTH=0.2,
    BINNINGALPHA=1.1,
    BINNINGBETA=1.6,
    PARAMETER=GAMMAZ,
    ADAPTIVEBINNING=TRUE,
    TABLEPRINTFREQ=20;

FSOpen: FIELDSOLVER, TYPE=OPEN, BINS=Bins,
```

```
    NX=16, NY=16, NZ=16,
    PARFFTX=TRUE, PARFFTY=TRUE, PARFFTZ=TRUE,
    BCFFTX=OPEN, BCFFTY=OPEN, BCFFTZ=OPEN,
    GREENSF=INTEGRATED,
    BBOXINCR=5.0;

Pulse: DISTRIBUTION, TYPE=FLATTOP,
    NPARTDIST=n_particles,
    SIGMAX=7.5e-4, SIGMAY=7.5e-4,
    TRISE=1e-12, TFALL=1e-12,
    TPULSEFWHM=3e-12,
    CUTOFFLONG=3.0,
    EMITTED=TRUE;

Cathode: EMISSIONSOURCE,
    DISTRIBUTION=Pulse,
    R0Z=0.0, P0Z=0.0,
    EMISSIONMODEL=ASTRA, EKIN=0.2,
    SHIFTED_GREENS_FUNCTION=TRUE,
    ZEROFACE_MAXSTEPS=300;
Sources: EMISSIONSOURCELIST=(Cathode);

Beam1: BEAM, PARTICLE=ELECTRON,
    PC=P0, NALLOC=n_particles,
    BCHARGE=bunch_charge, CHARGE=-1,
    SOURCES=Sources;

TRACK, LINE=GunLine, BEAM=Beam1,
    DT={1e-12}, MAXSTEPS={2000}, ZSTOP={0.5};
  RUN, METHOD=PARALLEL, FIELDSOLVER=FSOpen;
ENDTRACK;

QUIT;
```

For an electron, the negative `EZ` accelerates motion in $+z$. `ASTRA` emission gives each particle a forward thermal momentum with magnitude set by `EKIN`. During emission, `BINNING` groups particles by longitudinal gamma; OPALX solves each bin in its approximate rest frame and accumulates the lab-frame electric and magnetic fields.

`SHIFTED_GREENS_FUNCTION` adds the image contribution needed for zero potential at the cathode plane `R0Z=0`. It requires both `TYPE=OPEN` and named binning. The correction stops after 300 global steps; this cutoff and the 16^3 mesh are deliberately small tutorial values and must be converged for a production calculation.

Part II

User Guide

6 Introduction

OPALX is a modern particle-accelerator simulation program built for portable execution on CPUs and GPUs. It retains OPAL's MAD-inspired input language while reworking the runtime around current C++, MPI, Kokkos, and IPPL interfaces.

This manual documents OPALX. Historical OPAL behavior appears only on pages where the **Documentation version** selector is present.

6.1 What OPALX models

OPALX tracks charged macro-particles through ordered beamline elements and can couple them to external electromagnetic fields and self-consistent particle-mesh space charge. A normal calculation combines:

- One or more particle distributions and emission sources.
- A beam definition and ordered beamline.
- External-field elements such as magnets and RF cavities.
- An optional field solver for collective fields.
- A `TRACK` block that defines time steps and stopping conditions.

The current executable supports serial, OpenMP, CUDA, and HIP Kokkos backends, subject to the build configuration and the backend support of each algorithm.

6.2 Where to begin

New users should follow the installation instructions, then run one of the worked input files. The Input Language and Execution page explains how definitions and actions fit together.

The maintained project entry points are:

- OPALX source
- OPALX API documentation
- Regression-test results
- This manual's repository

6.3 Parallel execution

OPALX combines MPI domain decomposition with a selected Kokkos execution backend. GPU runs normally use one MPI rank per device. Numerical results can depend on mesh resolution, time step, decomposition, reduction order, and random sampling, so production studies should record all five.

6.4 Output and logging

The command-line option `--info <level>` controls stdout detail:

Level	Intended use
1	Coarse status and important events; recommended for production.
2	Normal progress and run summaries.
3	Detailed algorithm progress.
4	Fine-grained implementation detail.
5	Full debug or trace output.

For example:

```
opalx inputfile.in --info 1
```

Input-level output frequencies and diagnostics are controlled through `OPTION`, diagnostic elements, and field-output commands.

6.5 Documentation status

OPALX is evolving. A page marked **current** has been checked against the current source, while **experimental** identifies a working interface that may still change. An explicit unavailable notice means that the accompanying OPAL documentation is retained for comparison and planning, not accepted by OPALX.

6.6 Citation

A release-specific OPALX citation has not yet been published. Until one is available, record the exact source revision and cite the publications attached to the physical models used in the calculation. See Publications and Citation.

7 Conventions

OPALX uses SI units internally. Input attributes that use another convention state it explicitly in their parameter tables.

7.1 Physical units

Quantity	Input convention
Length	m
Angle	rad
Time	s unless an attribute states otherwise
Quadrupole coefficient	$\mathrm{T\,m^{-1}}$
Multipole coefficient, $2n$ poles	$\mathrm{T\,m^{-n+1}}$
Electric voltage	MV
Electric field strength	$\mathrm{MV\,m^{-1}}$
Magnetic field strength	T in current OPALX field output
Frequency	MHz
Charged-particle `BEAM.ENERGY`	GeV
Photon `BEAM.ENERGY`	eV, following the CAIN convention
Particle mass	$\mathrm{GeV\,c^{-2}}$ in `BEAM.MASS`
Normalized momentum	$\beta\gamma$ where explicitly stated
Beam current	A
Particle charge	elementary-charge units
Normalized and geometric emittance	m rad

The analytic `LASER` definition uses `WAVELENGTH`, `WAISTX`, and `WAISTY` in meters, `PULSEENERGY` in joules, and `PULSELENGTH` in seconds. `DIR` and `STOKES` are dimensionless three-vectors.

7.2 Symbols

Symbol	Definition
X, Y	Ellipse axes in the transverse plane; $X = Y = R$ for a circular beam.

Symbol	Definition
R	Circular beam radius.
R^*	Effective elliptical radius, $(X+Y)/2$.
σ_x, σ_y, σ_z	RMS beam sizes in the respective coordinates.
σ_r	Circular-beam RMS radius, $\langle r^2 \rangle^{1/2}$.
β, γ	Relativistic velocity and Lorentz factors.
p_0	Reference momentum. Input files commonly expose it as `P0`.
I	Beam current.
I_0	Species-dependent Alfvén current.
k_p	Generalized beam perveance.
q, m	Particle charge and mass.

Coordinate frames and transformations are defined in the Physics coordinate-system chapter.

7.3 Multipole conversion

When importing a normalized quadrupole strength k_1 from codes such as Elegant, convert it to the magnetic gradient required by OPALX using the reference rigidity. In the historical convention,

$$\frac{dB_y}{dx} = \frac{E[\mathrm{GeV}]}{0.29979}\, k_1,$$

with the sign fixed by the coordinate and charge conventions of the source code. A convenient conversion for energy in MeV is

```
def k1_to_gradient(k1, energy_mev):
    return 3.33564095e-3 * energy_mev * k1
```

Always verify the imported sign with a single-particle trajectory before using the result in a lattice study.

8 Input Language and Execution

OPALX reads a MAD-inspired, statement-oriented language. This page is the overview: it explains what can appear in an input file and directs readers to the maintained guide for each interface. Attribute-level details belong on the linked pages and in the Command Reference.

The catalog was checked against the parser and the registrations in `opalx/src/OpalConfigure/Configure.c` at OPALX revision `d1e762f15a2a`.

8.1 A simulation at a glance

Like the NumPy user guide, the manual first groups concepts by what readers are trying to accomplish and then provides a complete lookup catalog.

Reuse expressions

Values and variables

Define named real, Boolean, string, and vector values before using them in later attributes. See Command Format for expression syntax and Control Statements for executable language constructs.

Create particles

Beam and sources

Combine a distribution, one or more emission sources, and a beam definition.

Build the machine

Elements and lines

Create named elements, then assemble them into a `LINE` in tracking order.

Choose self-fields

Solver structures

Configure a field solver and, when needed, a reusable binning definition.

Run the model

Tracking block

Use `TRACK`, `RUN`, and `ENDTRACK` to bind the line, beam, sources, time step, and solver to one execution.

Control the run

Actions and options

Set global runtime options, compose input files, inspect values, and terminate parsing with the control statements.

8.2 Core syntax

Statements normally end with a semicolon. Names are case-insensitive in normal OPALX input, while string values and file paths retain their contents. The complete lexical, expression, array, selection, and constraint rules are in Command Format; the formal grammar is in Appendix B.

Form	Example	Meaning
Action	`OPTION, INFO=TRUE;`	Execute a registered command immediately.
Named definition	`Beam1: BEAM, PARTICLE=ELECTRON, ...;`	Clone a registered class, assign attributes, and store it under a name.
Typed value	`REAL length = 0.5;`	Define a scalar, string, Boolean, or vector value for later expressions.
Immediate assignment	`length = 0.5;`	Evaluate the right-hand expression when the assignment is parsed.
Expression assignment	`momentum := beta * gamma;`	Store an expression that can be reevaluated when its dependencies change.

Form	Example	Meaning
Attribute assignment	`Beam1->BCURRENT = 0.1;`	Update an existing object's attribute; `name->ATTR[n]` updates an array component.
Beamline	`L1: LINE=(D1, Q1, D2);`	Assemble named elements in traversal order.
Tracking block	`TRACK, ...; RUN, ...; ENDTRACK;`	Enter the tracking parser, execute a run, and return to the main parser.
Compound block	`{ statement; statement; }`	Group statements for `IF`, `WHILE`, or a macro body.

```
TITLE, STRING="Minimal beamline study";

REAL cell_length = 1.0;
D1: DRIFT, L=cell_length;
CELL: LINE=(D1);

TRACK, LINE=CELL, BEAM=BEAM1, MAXSTEPS={10}, DT={1.0e-11};
  RUN, METHOD="PARALLEL", FIELDSOLVER=FS1;
ENDTRACK;
QUIT;
```

This fragment illustrates ordering and syntax, not a complete runnable model. The Worked Input Files contain complete examples with beam, distribution, source, and field-solver definitions.

8.3 Parser-native structures

These forms are implemented directly by the parser rather than registered as simulation objects.

Structure	Short explanation
`BOOL name = expression;`	Define a Boolean value. `CONSTANT BOOL` and `CONST BOOL` are accepted equivalent forms.
`REAL name = expression;`	Define or update a real scalar variable. Omitting `REAL` is allowed only when the name already exists.

Structure	Short explanation
`CONST name = expression;`, `CONSTANT REAL name = expression;`	Define a real constant that cannot be redefined. `CONST` and `CONSTANT` are synonyms.
`STRING name = expression;`	Define a string value. `CONSTANT STRING` and `CONST STRING` are also accepted.
`VECTOR name = {...};`, `REAL VECTOR name = {...};`	Define a real-valued array. `CONSTANT VECTOR` and `CONSTANT REAL VECTOR` are also accepted.
`name->ATTRIBUTE = expression;`	Assign an existing object's attribute; use `:=` to retain a non-constant expression.
`SHARED name: CLASS, ...;`	Set the defined object's shared flag. This is an advanced ownership feature whose effect depends on the object class.
`name(arguments): MACRO { ... }`	Define a parameter-substitution macro; `name(actuals);` expands and executes it.
`IF (condition) statement ELSE statement`	Execute one of two statements. Use compound blocks for more than one statement per branch.
`WHILE (condition) statement`	Re-evaluate a Boolean condition and repeat one statement or compound block.
`{ ... }`	Group multiple statements into one compound statement.

8.4 Reusable simulation definitions

These named objects describe the bunch and the numerical structures used by a run. Define dependencies before the object that refers to them.

Definition	Short explanation	Maintained guide
`BEAM`	Defines particle species, reference energy or momentum, current, frequency, allocation, source list, and global processes.	Beam
`DISTRIBUTION`	Defines how initial positions, momenta, or emission times are sampled or read from a file.	Distribution
`EMISSIONSOURCE`	Couples one distribution to offsets, timing, emission energy, and image-charge settings.	Emission source

Definition	Short explanation	Maintained guide
EMISSIONSOURCELIST	Stores an ordered, non-empty list of named emission sources for BEAM.SOURCES or TRACK.SOURCES.	Emission-source list
FIELDSOLVER	Selects the Poisson backend, mesh, decomposition, boundary conditions, optional binning, and image-charge mode.	Field solver
BINNING	Configures the longitudinal histogram used by binned space charge.	Binning

8.5 Beamline definitions

LINE is the lattice container. The remaining entries are every element class registered by the current executable. Only source-audited element pages are linked; plain-text entries are registered but do not yet have a maintained user-guide section.

Definition	Short explanation	Maintained guide
LINE	Orders named elements into the lattice selected by TRACK.LINE.	Beam Lines
DRIFT	Field-free propagation over a finite length.	DRIFT
CONSTANTEFIELDCAVITY	Applies a spatially uniform, constant electric field inside the element.	CONSTANTEFIELDCAVITY
QUADRUPOLE	Defines a thick normal or skew quadrupole.	QUADRUPOLE
MULTIPOLE	Defines a thick hard-edge multipole from normal and skew strength arrays.	MULTIPOLE
MULTIPOLET	Defines a combined-function multipole with fringe and placement controls.	MULTIPOLET
SOLENOID	Defines a solenoid from normalized strength or a field map.	SOLENOID

Definition	Short explanation	Maintained guide
RFCAVITY	Defines a standing-wave or single-gap RF cavity, normally from a field map.	RFCAVITY
TRAVELINGWAVE	Defines a traveling-wave RF structure with cell and phase-advance controls.	TRAVELINGWAVE
RBEND	Defines a rectangular bending magnet.	RBEND
SBEND	Defines a sector bending magnet.	SBEND
VERTICALFFAMAGNET	Defines a vertical fixed-field alternating-gradient magnet.	VERTICALFFAMAGNET
VARIABLE_RF_CAVITY	Defines an RF cavity whose phase, amplitude, and frequency are provided by named time models.	VARIABLE_RF_CAVITY
LASER	Defines a passive analytic laser pulse with direction and polarization.	LASER
MONITOR	Records beam diagnostics at an element location.	MONITOR
PROBE	Records particle crossings over a configured probe span and accumulates peak diagnostics.	PROBE
MARKER	Marks a named lattice location without applying a field.	MARKER

8.6 Time-dependence definitions

These registered definitions provide named functions used by time-dependent RF and scaling attributes. Dedicated parameter pages have not yet been written.

Definition	Short explanation
POLYNOMIAL_TIME_DEPENDENCE	Defines a polynomial in time from coefficients.
SINUSOIDAL_TIME_DEPENDENCE	Defines a sum of sinusoidal components with frequency, phase, amplitude, and offset arrays.

SPLINE_TIME_DEPENDENCE	Defines linear or smoothed cubic interpolation through time-value samples.

8.7 Tracking-block structures

The TRACK action switches to a small nested command set. RUN and ENDTRACK are valid only inside that block.

Structure	Short explanation	Maintained guide
TRACK	Selects the lattice, beam or beams, source list, stepping limits, time integrator, and tracking interval.	Tracking
RUN	Constructs and executes the selected parallel tracking method and field solver.	Tracking
ENDTRACK	Leaves tracking mode and returns input processing to the main parser.	Tracking

8.8 Executable actions

Actions run when the parser encounters them; unlike named definitions, they do not merely describe a later simulation object.

Action	Short explanation	Maintained guide
OPTION	Sets global logging, output, random-number, space-charge, load-balancing, and synchronization behavior.	Runtime options
TITLE	Sets the calculation title.	Control statements
CALL	Temporarily reads and executes another input file, then resumes the caller.	Control statements
ECHO	Writes a message to the OPALX echo stream.	Control statements
HELP or CLASS?	Prints parser help for a registered class or object.	Control statements
VALUE	Evaluates and prints one or more real expressions.	Control statements

Action	Short explanation	Maintained guide
`SELECT`	Selects lattice positions by range, class, type, or name pattern for subsequent selection-aware commands.	Control statements
`DUMPEMFIELDS`	Schedules electromagnetic-field sampling on a Cartesian or cylindrical space-time grid.	Diagnostics and output
`SYSTEM`	Runs an operating-system command on rank zero.	Control statements
`PSYSTEM`	Runs an operating-system command on every rank. Use with particular care in MPI jobs.	Control statements
`STOP`	Stops further input processing.	Control statements
`QUIT`	Stops input processing; conventionally the final statement in an input file.	Control statements

8.9 Execution order and name resolution

OPALX processes input in order. Values, objects, and macros must normally be defined before a later statement refers to them. Named definitions are stored in the main object directory; actions are cloned, parsed, and executed immediately. Before an action executes, OPALX updates objects whose expression dependencies changed.

Within a normal simulation the practical order is:

1. Set `OPTION` and define reusable values.
2. Define distributions, emission sources, an emission-source list, and a beam.
3. Define elements and assemble a `LINE`.
4. Define optional `BINNING` and the selected `FIELDSOLVER`.
5. Enter `TRACK`, issue `RUN`, and close the block with `ENDTRACK`.
6. Finish with `QUIT`.

9 Command Format

OPALX and OPAL use the same MAD-like command grammar. The dialect is close to MAD9; for older MAD-style inputs, see the conversion guide.

9.1 Statements and Comments

Input is free format. Statements are tokenized and terminated by semicolons. Comments may be introduced by `//`, and C-style block comments are also accepted.

General command form:

```
keyword, attribute, ..., attribute;
label: keyword, attribute, ..., attribute;
```

The three parts are:

1. `label`: optional for executable commands, required for definitions stored for later use
2. `keyword`: identifies the command or element type
3. `attribute`: one of

```
attribute-name
attribute-name = attribute-value
attribute-name := attribute-value
```

`=` evaluates the expression immediately. `:=` stores the expression and re-evaluates it when dependent values change. A logical attribute may be given without a value, in which case it implies `TRUE`.

If a labeled command is stored, it can later be re-executed by label only or re-executed with modified attributes:

```
QF: QUADRUPOLE, L=1, K1=0.01;
QF, L=2;

LMD: LMDIF, CALLS=10;
LMD;
LMD, CALLS=100, TOLERANCE=1E-5;
```

9.2 Identifiers or Labels

An identifier names a keyword, element, beamline, variable, or array. Unquoted identifiers begin with a letter and may contain letters, digits, periods, and underscores. Quoted identifiers may contain other special characters, though the source manual discourages that because it complicates downstream post-processing.

Unquoted names are converted to upper case before storage.

9.3 Command Attribute Types

Object attributes are referenced as

```
object-name->attribute-name
object-name->attribute-name[index]
```

The attribute categories used throughout OPAL are:

Attribute Type
String
Logical
Real expression
Deferred expression
Place
Range
Constraint
Variable reference
Regular expression
Array
Array of logical
Array of real
Array of string
Array of token lists

9.4 String Attributes

String attributes carry titles, file names, class names, or options. Strings may be quoted with single or double quotes. Concatenation uses `&`, and the function `STRING(X)` converts a real expression to a string.

Operator / Function	Meaning
`X & Y`	concatenate strings
`STRING(X)`	convert real expression `X` to a string

Examples:

```
TITLE, "This is a title for the program run ""test""";
CALL, FILE="save";

REAL X=1;
STRING L2=LEP&STRING(X+1);
```

9.4.1 Predefined String Attributes

Many string attributes are restricted to a predefined set of accepted values. Such attributes do not need to be quoted. If the supplied value is not in the allowed set, OPAL raises an exception.

9.5 Logical Expressions

Logical expressions use the same precedence and syntax style as C. They are built from relations combined with `&&` and `||`.

Grammar sketch:

```
relation      ::= TRUE | FALSE | real-expr rel-operator real-expr
rel-operator  ::= == | != | < | > | >= | <=
and-expr      ::= relation | and-expr && relation
logical-expr  ::= and-expr | logical-expr || and-expr
```

Operator	Meaning
`X < Y`	true if `X` is less than `Y`
`X <= Y`	true if `X` is not greater than `Y`
`X > Y`	true if `X` is greater than `Y`
`X >= Y`	true if `X` is not less than `Y`
`X == Y`	true if `X` equals `Y`
`X != Y`	true if `X` differs from `Y`
`X && Y`	true if both are true
`X || Y`	true if at least one is true

Examples:

```
OPTION, ECHO=TRUE;
OPTION, ECHO;
X>10 && Y<20 || Z==15
```

9.6 Real Expressions

Any real value may be given as an arithmetic expression. Expression definitions may be entered in any order; OPAL evaluates them when the required operands are available.

Grammar sketch:

```
real-ref  ::= real-variable |
              real-array[index] |
              object->real-attribute |
              object->array-attribute[index]

table-ref ::= table@place->column-name

primary   ::= literal-constant |
              symbolic-constant |
              # |
              real-ref |
              table-ref |
              function-name(arguments) |
              (real-expression)

factor    ::= primary | factor ^ primary
term      ::= factor | term * factor | term / factor
real-expr ::= term | +term | -term | real-expr + term | real-expr - term
```

9.7 Operators

Operator	Meaning
`+X`	unary plus
`-X`	unary minus
`X + Y`	addition
`X - Y`	subtraction
`X * Y`	multiplication
`X / Y`	division
`X ^ Y`	power

9.8 Real Functions

The source chapter lists the following built-ins:

- no arguments: `RANF()`, `GAUSS()`, `GETEKIN()`, `USER0()`
- one argument: `TRUNC`, `ROUND`, `FLOOR`, `CEIL`, `SIGN`, `SQRT`, `LOG`, `EXP`, `SIN`, `COS`, `ABS`, `TAN`, `ASIN`, `ACOS`, `ATAN`, `TGAUSS`, `USER1`, `EVAL`
- two arguments: `ATAN2`, `MAX`, `MIN`, `MOD`, `USER2`

Functions of arrays:

- `VMAX`
- `VMIN`
- `VRMS`
- `VABSMAX`

The source manual explicitly warns that random generators inside ordinary expressions may be re-evaluated at unexpected times. `EVAL(...)` can be used to force immediate evaluation and store the result as a constant.

9.9 Operands in Expressions

9.9.1 Literal Constants

Numerical values follow FORTRAN-style literal syntax. Examples:

```
1, 10.35, 5E3, 314.1592E-2
```

9.9.2 Symbolic Constants

The predefined symbolic constants include mathematical constants and particle masses:

Name	Meaning / Value
`PI`, `TWOPI`, `RADDEG`, `DEGRAD`, `E`	usual mathematical constants
`EMASS`, `MUMASS`, `PMASS`, `DMASS`, `HMMASS`, `H2PMASS`, `ALPHAMASS`, `CMASS`, `XEMASS`, `UMASS`	predefined particle masses
`CLIGHT`, `AMU`	speed of light and atomic mass unit
`_OPALVERSION_`	OPAL version constant
`RANK`	MPI rank

9.9.3 Variable Labels

Global real and vector variables are defined with:

```
REAL X=expression;
REAL X:=expression;
VECTOR X=vector-expression;
VECTOR X:=vector-expression;
```

Example:

```
REAL L=1.0;
REAL X:=L;
D1: DRIFT, L:=X;
D2: DRIFT, L:=2.0-X;
```

9.9.4 Element or Command Attributes

Attributes can appear directly in arithmetic expressions:

```
element-name->attribute-name
command-name->attribute-name
element-name->attribute-name[index]
command-name->attribute-name[index]
```

Example:

```
D1: DRIFT, L=1.0;
D2: DRIFT, L=2.0-D1->L;
```

9.9.5 Deferred Expressions and Random Values

Expressions containing random generators may be deferred and sampled when the target command actually executes. Such a value cannot safely be used as an operand inside another deferred expression.

9.10 Element Selection

Element selection is not available in `OPAL-T` and `OPAL-CYCL`, but it is part of the general language.

9.10.1 Element Selection

A `place` denotes a specific element occurrence or the position immediately before or after the full line:

```
place ::= element-name |
          element-name[index] |
          #S |
          #E |
          line-name::place
```

Examples from the source chapter:

- `C[1]`: first occurrence of `C`
- `#S`: beginning of the line
- `M[2]`: second occurrence of `M`
- `#E`: end of the line
- `S[1]::M[1]`: marker `M` nested in the first occurrence of sub-line `S`

9.10.2 Range Selection

A range is one place or two places joined by `/`:

```
range ::= place | place/place
```

Examples:

- `#S/#E`: the full line
- `A[1]/A[2]`: from the first `A` through the second `A`
- `S[1]/S[2]`: from entry of first `S` through exit of second `S`

9.11 Constraints

Constraints are mainly used in matching and optimization:

```
constraint          ::= array-expr constraint-operator array-expr
constraint-operator ::= == | < | >
```

9.12 Variable Names

Variable references can be:

```
real-variable
object->real-attribute
```

9.13 Regular Expressions

Some commands accept UNIX-style regular expressions. The pattern must be quoted. The source chapter documents the usual forms:

- `.`
- `[abc]`
- `[a-zA-Z]`
- `*`
- `\character`

Examples:

```
SELECT, PATTERN="D.."
SELECT, PATTERN="K.*QD.*\\.R1"
```

9.14 Arrays

An array is normally entered as a list in braces:

```
{value, ..., value}
```

If the array has only one value, the braces may be omitted.

9.14.1 Logical Arrays

```
logical-array ::= { logical-list }
logical-list  ::= logical-expr | logical-list , logical-expr
```

Example:

```
{true,true,a==b,false,x>y && y>z,true,false}
```

9.14.2 Real Arrays

Real arrays mirror scalar arithmetic and support:

- array variables and object array attributes
- `TABLE(...)`
- `ROW(...)`
- `COLUMN(...)`
- component-wise real functions

Example:

```
{a, a+b, a+2*b}
```

Important generator forms:

```
TABLE(n2, expression)
TABLE(n1:n2, expression)
TABLE(n1:n2:n3, expression)
ROW(table, place)
ROW(table, place, column-list)
COLUMN(table, column)
COLUMN(table, column, range)
```

In `TABLE(...)`, the special symbol `#` is replaced by the running index.

9.14.3 String Arrays

String arrays are lists of string expressions. Example:

```
{A, "xyz", A & STRING(X)}
```

9.14.4 Token List Arrays

Token-list arrays are lists of token lists. Example:

```
{X:12:8, Y:12:8}
```

10 Control Statements

These statements control OPALX parsing, variables, file inclusion, and simple flow. Global runtime settings are documented separately under `OPTION`.

10.1 Getting Help

10.1.1 `HELP` Command

The `HELP` command prints information about a command or object type.

```
HELP;                  // Help on the HELP command itself
HELP, NAME=label;      // Show attribute types of label
HELP, label;           // Shortcut form
```

Examples:

```
HELP;
HELP, NAME=FIELDSOLVER;
HELP, FIELDSOLVER;
```

10.2 `STOP` / `QUIT`

```
STOP;
QUIT;
```

`STOP` and `QUIT` terminate execution of the current input stream. In a nested `CALL` file they return to the caller; in the main input file they end the run. Any statements after `STOP` or `QUIT` in the same file are ignored.

10.3 Parameter Statements

10.3.1 Variable Definitions

OPALX supports real scalars, real vectors, logical variables, and string constants.

10.3.1.1 Real Scalar Variables

```
REAL variable-name = real-expression;
```

The expression form

```
REAL variable-name := real-expression;
```

retains the expression tree so it can be reevaluated. With `=`, the right-hand side is evaluated immediately.

Circular expression definitions are not allowed.

Examples:

```
REAL GEV = 100;
P0 = GEV;
```

`P0` is a reserved global reference momentum used to normalize magnetic-field coefficients.

10.3.1.2 Real Vector Variables

```
REAL VECTOR variable-name = vector-expression;
```

This creates a real vector. `VECTOR` without `REAL` is also accepted by the current parser.

Examples:

```
REAL VECTOR A = TABLE(10, #);
REAL VECTOR B = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

10.3.1.3 Logical Variables

```
BOOL variable-name = logical-expression;
```

Example:

```
BOOL FLAG = X != 0;
```

10.3.2 Symbolic Constants

Built-in mathematical and physical constants are provided. Additional constants can be defined with:

```
REAL CONST label: CONSTANT = real-expression;
```

The value is evaluated immediately when the line is read and cannot be redefined later.

Example:

```
CONST IN = 0.0254;
```

10.3.3 Vector Values

Vectors can also be defined directly from expressions:

```
REAL VECTOR vector-name = vector-expression;
```

Examples:

```
VECTOR A_AMPL = {2.5e-3, 3.4e-2, 0, 4.5e-8};
VECTOR A_ON = TABLE(10, 1);
```

10.3.4 Assignment to Variables

To force immediate evaluation and assign the resulting value as a constant, use `EVAL(...)`:

```
variable-name = EVAL(real-expression);
```

This behaves like assignment in C or Fortran.

Example:

```
REAL X = 0;
WHILE (X <= 0.10) {
  VALUE, VALUE={X};
  X = EVAL(X + .01);
}
```

10.3.5 `VALUE`: Output of Expressions

```
VALUE, VALUE = expression-vector;
```

VALUE evaluates its arguments with the current variable state and prints the result. It is useful for debugging and tabulating expressions.

Example:

```
REAL A = 4;
VALUE, VALUE = TABLE(5, #*A);
REAL P1 = 5;
REAL P2 = 7;
VALUE, VALUE = {P1, P2, P1*P2-3};
```

10.4 Miscellaneous Commands

10.4.1 ECHO Statement

```
ECHO, MESSAGE=message;
ECHO, message;
```

The given string is written immediately to the echo stream.

10.4.2 SYSTEM

```
SYSTEM, CMD=string;
SYSTEM, string;
```

SYSTEM executes an operating-system command on rank zero, then returns control to OPALX.

Example:

```
SYSTEM, "ls -l";
```

It can also be used to generate helper files externally and then include them with CALL.

10.4.3 PSYSTEM

PSYSTEM is the parallel counterpart to SYSTEM: the command is executed on all ranks instead of only one.

10.5 `TITLE` Statement

```
TITLE, STRING=page-header;
TITLE, page-header;
TITLE, STRING="";
```

`TITLE` stores the calculation title used by subsequent output.

10.6 File Handling

10.6.1 `CALL` Statement

```
CALL, FILE=file-name;
CALL, file-name;
```

`CALL` switches input processing to another file. Reading continues there until end-of-file or a `STOP` / `QUIT`, after which control returns to the caller.

Example:

```
CALL, FILE="structure";
CALL, "structure";
```

10.6.2 `init.opal`

Before the main input file is processed, OPALX executes `$HOME/init.opal` if it exists. This behaves like an implicit

```
CALL, FILE="$HOME/init.opal";
```

at the beginning of every run.

Typical uses are:

- shared `OPTION` settings
- reusable `MACRO` definitions
- global physics constants

Because it is implicit, it can also be a source of confusion when switching machines. For portable input decks, an explicit `CALL` is often easier to manage.

10.7 SELECT: Element Selection

SELECT marks lattice elements for commands that consume the selection state. It can select all elements, clear a selection, or match a line range, class, type, or name pattern.

```
SELECT, LINE=Line1, RANGE=#S/#E, CLASS=QUADRUPOLE;
SELECT, LINE=Line1, PATTERN="Q.*";
SELECT, LINE=Line1, CLEAR=TRUE;
```

Attribute	Meaning
LINE	Name of the line to inspect.
FULL	Select every position in the line.
CLEAR	Clear all selections.
RANGE	Restrict selection to a lattice range.
CLASS	Match an element class.
TYPE	Match an element type name.
PATTERN	Match element names with a regular expression.

10.8 IF: Conditional Execution

The input language supports C-like conditional execution:

```
IF (logical) statement;
IF (logical) statement; ELSE statement;
IF (logical) { statement-group; }
IF (logical) { statement-group; }
  ELSE { statement-group; }
```

Statements inside the block still end with semicolons, but the closing brace does not.

10.9 WHILE: Repeated Execution

Repeated execution is expressed as:

```
WHILE (logical) statement;
WHILE (logical) { statement-group; }
```

The logical condition is re-evaluated on every iteration. Some variable inside the loop must change in a way that allows termination.

10.10 MACRO: Macro Statements

Macros provide subroutine-like token substitution.

Definitions:

```
name(formals): MACRO { token-list }
name(): MACRO { token-list }
```

Calls:

```
name(actuals);
name();
```

The actual arguments replace all occurrences of the corresponding formal names inside the stored token list.

Example:

```
SHOWIT(X): MACRO {
   VALUE, VALUE={X};
}

KS(X,Y): MACRO {
   Y = 3e-3*X + 0.5e-3;
}

SHOWIT(PI);
REAL X = 2;
REAL Y;
KS(X,Y);
```

11 Runtime Options

`OPTION` changes process-wide parser, tracking, output, and diagnostic settings.

OPALX

```
OPTION, VERSION=10900, PSDUMPFREQ=10, STATDUMPFREQ=1,
        BOUNDPDESTROY=10, QM_MODE=SINGLE;
```

Several attributes remain registered for input compatibility even though the current `ParallelTracker` does not consume them. Those attributes are listed separately so parser acceptance is not confused with runtime support.

The Boolean `INFO` option is distinct from the executable's `--info <level>` argument. `INFO` gates a small set of parser messages; the command-line level controls the `level1`, `level2`, and more detailed runtime streams.

11.1 Command-line logging

Pass `--info N` after the input file to show messages through level `N`:

Level	Intended content
1	Essential run status and rare or critical events.
2	Normal progress and run summaries suitable for production.
3	Algorithm progress, including time-step, repartitioning, and binning setup.
4	Fine-grained implementation details for debugging.
5	Full trace-style output, including frequent step and solver details.

Use level `1` or `2` for production runs. Higher integer values are accepted, but OPALX currently defines no additional message categories above level `5`.

11.2 Parser and messages

Option	Default	Effect
ECHO	FALSE	Echo input while parsing.
INFO	TRUE	Enable informational parser messages.
TRACE	FALSE	Print timing before and after traceable commands. Put this in the first OPTION statement.
WARN	TRUE	Enable warnings from expressions and selectors.
SEED	123456789	Random seed used by current samplers and stochastic physics. -1 requests time-based seeding where supported.
TELL	FALSE	Print all effective option values immediately. Put this in the last OPTION statement whose settings should be shown.

11.3 Output and diagnostics

Frequencies are measured in global tracking steps unless stated otherwise.

Option	Default	Effect
PSDUMPFREQ	10	Phase-space dump frequency. 0 is converted internally to an effectively disabled frequency.
STATDUMPFREQ	10	Statistics dump frequency. 0 is converted internally to an effectively disabled frequency.

Option	Default	Effect
STEPINFOFQ	1	Per-step status frequency; values below zero become 0, which disables the lines.
PRINTRANKDISTRFQ	0	Per-rank particle-count table frequency; 0 disables it and negative values become 0.
RHODUMP	FALSE	Add the scalar charge-density field to supported HDF5 output.
EBDUMP	FALSE	Add the electric and magnetic field at each particle to phase-space HDF5 output.
RANKDUMP	FALSE	Add each particle's current MPI rank to phase-space HDF5 output.
ENABLEHDF5	TRUE	Enable HDF5 phase-space and loss output.
ENABLEVTK	TRUE	Enable supported VTK geometry output.
ASCIIDUMP	FALSE	Make supported monitor and plug-in loss sinks write ASCII instead of HDF5.
BEAMHALOBOUNDARY	0	Halo boundary in rms beam sizes for statistics; values at or below zero disable this count.
HALOSHIFT	0	Constant subtracted from the reported halo parameter.
COMPUTEPERCENTILES	FALSE	Compute the configured beam-size and normalized-emittance percentiles; samples below 100 particles are skipped.

11.4 Tracking, particles, and solvers

Option	Default	Effect and constraints
REPARTFREQ	10	Check particle repartitioning at this frequency. Values at or below zero disable periodic repartitioning.
LOADBALANCINGTHRESHOLD	0.05	Imbalance threshold for load balancing; must lie in [0,1].
BOUNDPDESTROY	10	Delete particles outside this many rms beam sizes. Values at or below zero disable the cut.
MINBINEMITTED	10	Delay space-charge calculation until the primary container has more particles than this threshold.
MINSTEPFORREBIN	200	Earliest step at which the current tracker may collapse emission bins.
AUTOPHASE	6	Number of RF phase-search refinements; 0 disables autophasing.
QM_MODE	SINGLE	SINGLE stores one charge and mass per container; ATTRIBUTES allocates per-particle Q and M views.
AGGRESSIVE_STATE_SYNC	FALSE	MPI-allreduce every shared bunch-state mutation. Intended for diagnosing rank-local state divergence; adds collective overhead.
VERSION	10000	OPAL input-version code. RUN reports incompatible changes when the value predates registered input changes.

MINBINEMITTED and MINSTEPFORREBIN affect runtime bin handling. The definition of the

adaptive histogram itself is documented under Structures / Binning.

11.5 Accepted compatibility options

The following names are accepted and stored, but have no current consumer in the OPALX `ParallelTracker` source. Do not rely on them to change a simulation.

Option	Default	Historical or intended role
`PSDUMPEACHTURN`	`FALSE`	OPAL-CYCL phase-space dump after each turn.
`PSDUMPFRAME`	`GLOBAL`	OPAL-CYCL dump frame: `GLOBAL`, `BUNCH_MEAN`, or `REFERENCE`.
`SPTDUMPFREQ`	`1`	Single-particle trajectory dump frequency. `0` is converted to an effectively disabled frequency.
`SCSOLVEFREQ`	`1`	Requested space-charge solve frequency; parsed values below one become one.
`MTSSUBSTEPS`	`1`	Small-step count intended for multiple-time-stepping integration.
`REMOTEPARTDEL`	`0`	Historical remote-particle rms deletion threshold.
`REBINFREQ`	`100`	Historical frequency for resetting energy-bin identifiers.
`CSRDUMP`	`FALSE`	Historical CSR field and line-density output.
`NUMBLOCKS`	`0`	Recycled-CG Krylov-space vector limit.
`RECYCLEBLOCKS`	`0`	Recycled-CG recycle-space vector count.
`NLHS`	`1`	Old solutions retained for a solver starting-vector extrapolation.

Option	Default	Historical or intended role
CZERO	FALSE	Historical request for a distribution with zero centroid.
RNGTYPE	RANDOM	Accepts RANDOM, HALTON, SOBOL, or NIEDERREITER; current distribution samplers do not select their engine through it.
CLOTUNEONLY	FALSE	OPAL-CYCL closed-orbit and tune-only mode.
IDEALIZED	FALSE	Historical hard-edge path-length compatibility mode.
LOGBENDTRAJECTORY	FALSE	Historical request to write each bend trajectory.
MEMORYDUMP	FALSE	Historical SDDS memory output request; the current OPTION execution path does not apply it.
DELPARTFREQ	1	Historical particle-deletion frequency.

OPAL

The legacy OPAL form is:

```
OPTION, VERSION=integer, ECHO=logical, INFO=logical, TRACE=logical,
        WARN=logical, SEED=real, PSDUMPFREQ=integer,
        STATDUMPFREQ=integer,
        REPARTFREQ=integer, REBINFREQ=integer, TELL=logical;
```

The most frequently used controls are:

Option	Legacy OPAL meaning
VERSION	Input-file compatibility tag, conventionally encoded as major, minor, and patch digits.
ECHO, INFO, TRACE, WARN	Control input echo, information, trace, and warning streams.
TELL	Print the effective option settings.

Option	Legacy OPAL meaning
SEED	Random-number seed; -1 requests time-dependent seeding.
RNGTYPE	RANDOM, HALTON, SOBOL, or NIEDERREITER.
ASCIIDUMP	Select ASCII output for supported loss and diagnostic elements.
AUTOPHASE	Number of auto-phasing refinements; 0 disables auto-phasing.
PSDUMPFREQ, STATDUMPFREQ	Phase-space and statistics output frequencies.
REPARTFREQ, REBINFREQ	Particle repartition and energy-bin reset frequencies.
SCSOLVEFREQ	Space-charge solve reuse frequency for supported integrators.
ENABLEHDF5, ENABLEVTK	Enable or disable the corresponding output families.

11.5.1 Legacy defaults

Option	Default	Option	Default
AMR	FALSE	AMR_REGRID_FREQ	10
AMR_YT_DUMP_FREQ	10	ASCIIDUMP	FALSE
AUTOPHASE	6	BEAMHALOBOUNDARY	0.0
COMPUTEPERCENTILES	FALSE	CSRDUMP	FALSE
CZERO	FALSE	DELPARTFREQ	1
DUMPBEAMMATRIX	FALSE	EBDUMP	FALSE
ECHO	FALSE	ENABLEHDF5	TRUE
ENABLEVTK	TRUE	HALOSHIFT	0.0
IDEALIZE	FALSE	INFO	TRUE
LOGBENDTRAJECTORY	FALSE	MEMORYDUMP	FALSE
MINBINEMITTED	10	MINSTEPFORREBIN	200
NLHS	1	NUMBLOCKS	0
PSDUMPFREQ	10		
RECYCLEBLOCKS	0	REBINFREQ	100
REMOTEPARTDEL	0.0	REPARTFREQ	10
RHODUMP	FALSE	RNGTYPE	RANDOM
SCSOLVEFREQ	1	SEED	123456789
STATDUMPFREQ	10		
TELL	FALSE	TRACE	FALSE
VERSION	unset	WARN	TRUE

Legacy defaults varied by OPAL release and mode. Preserve the version tag and the exact executable revision when reproducing an older calculation.

12 Elements

Elements are named objects placed in three dimensions and assembled into a `LINE`. Shared concepts are described once; select OPALX or OPAL above for version-specific interfaces.

OPALX

The OPALX view covers every element type found in the current source registry, including interfaces without maintained regression inputs. It was audited in `opalx/src/OpalConfigure/Configure.cpp` at source commit `d1e762f15a2a`. Regression coverage is stated separately from parser registration.

Type	Regression use	External data
`DRIFT`	Field-free transport	none
`CONSTANTEFIELDCAVITY`	Uniform electric field	none
`QUADRUPOLE`	Normalized quadrupole	none
`MULTIPOLE`	Normalized dipole/quadrupole arrays	none
`MULTIPOLET`	Physical multipole coefficients with fringe fields	optional time model
`SOLENOID`	Mapped solenoid field	field map
`RFCAVITY`	Standing-wave RF field	field map
`TRAVELINGWAVE`	Traveling-wave RF field	field map
`LASER`	Source registered; no maintained regression input	none
`MONITOR`	Source registered; no maintained regression input	output file
`MARKER`	Source registered; no maintained regression input	none
`PROBE`	Source registered; no maintained regression input	output file
`RBEND`, `SBEND`	Source registered; no maintained regression input	none
`VERTICALFFAMAGNET`	Source registered; no maintained regression input	none
`VARIABLE_RF_CAVITY`	Source registered; no maintained regression input	time models

The Element Physics chapter contains derivations and benchmarks where available.

12.1 Common element syntax

Every element uses

```
name: ELEMENTTYPE, L=length, placement, type_specific_parameters;
```

and accepts the following common attributes.

Parameter	Default	Current behavior
`L`	unset (0 m when read)	Nominal element length. Use a positive value for the element kernels documented here.
`ELEMEDGE`	unset	Places the entrance at path length s in metres. May be combined with `PSI`, but not with absolute pose attributes.
`X`, `Y`, `Z`	`0 m`	Absolute lab-frame origin. Explicitly set at least one of `X`, `Y`, `Z`, `THETA`, or `PHI` to select pose placement.
`THETA`, `PHI`, `PSI`	`0 rad`	Absolute rotations about local y, x, and z. `PSI` is also the optional roll for `ELEMEDGE` placement.
`DX`, `DY`, `DZ`	`0 m`	Local translational misalignment applied after placement.
`DTHETA`, `DPHI`, `DPSI`	`0 rad`	Local rotational misalignment.
`APERTURE`	`ELLIPSE(1,1)` equivalent	Transverse aperture string, for example `CIRCLE(0.02)`, `ELLIPSE(0.03,0.02)`, `RECTANGLE(0.03,0.02)`, or `SQUARE(0.02)`. Dimensions are full widths in metres.
`DELETEONTRANSVERSEEXIT`	`TRUE`	Requests particle deletion when an element reports a transverse aperture exit.
`TYPE`	unset	Type-specific design selector; currently relevant to `RFCAVITY` below.
`WAKEF`, `PARTICLEMATTERINTERACTION`, `OUTFN`	unset	Parsed common compatibility attributes, but not acted on by the eight element implementations on this page.

Exactly one placement strategy is required. Supplying both `ELEMEDGE` and an absolute pose fails; supplying neither also fails. `PSI` alone does not select absolute placement.

12.2 DRIFT

`DRIFT` defines a field-free interval. It advances particles only through the normal tracker integration and provides the reference-path geometry between active elements.

```
D1: DRIFT, L=0.5, ELEMEDGE=0.0;
```

Parameter	Default	Current behavior
`L`	unset	Drift length in metres; use a positive value.
`GEOMETRY`	unset	Parsed boundary-geometry name, but not transferred by the current `DRIFT` implementation.
`NSLICES`	1	Map-tracking compatibility value; inactive in the current `PARALLEL` tracking path.

12.3 CONSTANTEFIELDCAVITY

`CONSTANTEFIELDCAVITY` adds a uniform Cartesian electric field over its local body interval. It is a DC element and is not processed by the RF autophaser.

```
Gun: CONSTANTEFIELDCAVITY,
    L=0.5, ELEMEDGE=0.0,
    EX=0.0, EY=0.0, EZ=-5.0;
```

Parameter	Default	Unit	Current behavior
`EX`	0	MV/m	Uniform local E_x.
`EY`	0	MV/m	Uniform local E_y.
`EZ`	0	MV/m	Uniform local E_z. A negative value accelerates electrons toward local $+z$.

The field is applied for local $0 < z \leq L$ in the particle kernel. Use `RFCAVITY` or `TRAVELINGWAVE` when an oscillating mapped RF field is required.

12.4 QUADRUPOLE

`QUADRUPOLE` is a convenience wrapper around `MULTIPOLE`. It converts normalized strengths to physical gradients using the reference rigidity.

```
Q1: QUADRUPOLE, L=0.5, ELEMEDGE=0.0, K1=0.54;
```

Parameter	Default	Unit	Current behavior
K1	0	m^{-2}	Normal normalized quadrupole strength.
K1S	0	m^{-2}	Skew normalized quadrupole strength.
DK1	0	m^{-2}	Parsed and stored normal error, but not applied by the current field kernel.
DK1S	0	m^{-2}	Parsed and stored skew error, but not applied by the current field kernel.

Use a positive L. Although inherited help text mentions integrated zero-length strengths, the current particle kernel has no nonzero field interval when L=0.

12.5 MULTIPOLE

MULTIPOLE accepts arrays of normalized normal and skew strengths. Array index zero is dipole and index one is quadrupole.

```
M1: MULTIPOLE, L=0.5, ELEMEDGE=0.0, KN={0.0,0.54};
```

Parameter	Default	Current behavior
KN	empty	Normal normalized strengths: index 0 dipole, index 1 quadrupole.
KS	empty	Skew normalized strengths with the same indexing.
DKN	empty	Parsed and stored normal-error array, but not applied by the current field kernel. Missing entries are padded with zero.
DKS	empty	Parsed and stored skew-error array, but not applied by the current field kernel. Missing entries are padded with zero.

> ⚠ Current order limit
>
> The parser accepts longer arrays, but the current particle and reference kernels do not agree: the bulk particle kernel applies only indices 0 and 1, while the host/reference kernel evaluates higher orders. A higher-order entry can therefore change the threaded reference orbit without applying the same field to tracked particles. Use `MULTIPOLET` instead.

Strengths are multiplied by the reference rigidity before tracking. As for `QUADRUPOLE`, use positive length; zero-length integrated operation is not effective in the current parallel kernel.

12.6 `MULTIPOLET`

`MULTIPOLET` defines straight or constant-radius curved magnetic multipoles using physical field coefficients and a smooth fringe model. It is distinct from the normalized `MULTIPOLE` interface.

```
MT1: MULTIPOLET,
    L=0.5, ELEMEDGE=0.0,
    TP={0.0,1.80},
    LFRINGE=0.05, RFRINGE=0.05,
    HAPERT=0.04, VAPERT=0.04;
```

Parameter	Default	Unit	Current behavior
`TP`	empty	$\mathrm{T\,m}^{-k}$	Physical coefficients b_k: index 0 dipole field, index 1 quadrupole gradient, and higher orders through index 5.
`LFRINGE`, `RFRINGE`	unset	m	Left and right tanh-fringe length scales. Set positive values.
`HAPERT`, `VAPERT`	unset	m	Internal horizontal and vertical aperture dimensions used by this field model.
`MAXFORDER`	3	terms	Longitudinal fringe expansion order. Must be in $[1, 9]$; real input is converted to an integer.
`MAXXORDER`	20	terms	Polynomial expansion order used by the curved implementation; real input is converted to an integer.
`ROTATION`	unset	rad	Rotation about the central axis for a straight skew element. Must be zero when `ANGLE` is nonzero.
`EANGLE`	unset	rad	Entrance angle for curved geometry.

Parameter	Default	Unit	Current behavior
BBLENGTH	unset	m	Half-length used for the field calculation's bounding region.
ANGLE	0	rad	Nonzero selects the constant-radius curved implementation; zero selects straight geometry.
VARRADIUS	FALSE	Boolean	Variable-radius selector. Variable-radius curved magnets are currently rejected.
ENTRYOFFSET	0	m	Accepted only with VARRADIUS=TRUE; there is no supported variable-radius curved configuration yet.
SCALING_MODEL	unset	name	Optional TIMEDEPENDENCE object multiplying the magnetic field as a function of time.

The dedicated HAPERT and VAPERT values configure this model's internal aperture; do not substitute the common APERTURE string for them. The six-entry TP capacity is a runtime limit even though the parser accepts a longer array.

12.7 SOLENOID

SOLENOID loads a magnetic field map and scales it with a normalized solenoid strength. The map defines the actual longitudinal field-support interval, which may differ from nominal L.

```
S1: SOLENOID,
    L=0.5, ELEMEDGE=0.0,
    KS=0.12, FMAPFN="solenoid.T7";
```

Parameter	Default	Current behavior
KS	unset (0 when read)	Normalized field-map scale in m^{-1}. It is converted using the reference rigidity.
DKS	unset (0 when read)	Additive scale error for tracked particles; the reference-particle field omits this error.
FMAPFN	unset	Required field-map filename. Tracking cannot apply a solenoid without a loaded map.

Parameter	Default	Current behavior
FAST	TRUE	Passed to the field-map loader as its fast/accuracy selection.

The element does not generate an analytic hard-edge solenoid when FMAPFN is missing. Keep the field map beside the input or use an unambiguous relative path.

12.8 RFCAVITY

RFCAVITY applies a time-dependent standing-wave field from a supported 1D or 2D dynamic field map. The autophased example shows the complete connection to a beam and tracker.

```
Gun: RFCAVITY,
    L=0.2927, ELEMEDGE=0.0,
    TYPE="STANDING", VOLT=60.0,
    FREQ=1300.0, LAG=0.0,
    FMAPFN="fieldmaps/DriveGun.T7";
```

Parameter	Default	Unit	Current behavior
TYPE	unset	name	STANDING is the maintained regression configuration. SINGLEGAP is recognized internally; other common design strings fall back to standing-wave behavior.
VOLT, DVOLT	unset	MV	Nominal field-map amplitude and additive amplitude error.
FREQ	unset	MHz	RF frequency. A field-map value differing by more than 1% replaces it with a warning.
LAG, DLAG	unset	rad	Nominal phase and additive phase error.
FMAPFN	unset	path	Required dynamic field map. Current particle application supports FM2DDynamic and Astra1DDynamic maps.
FAST	TRUE	Boolean	Passed to the field-map loader.
APVETO	FALSE	Boolean	If true, excludes this cavity from automatic phase optimization.
DESIGNENERGY	-1	MeV	Positive target exit kinetic energy; the autophaser also adjusts amplitude to reach it. Negative disables the target.

Parameter	Default	Unit	Current behavior
GEOMETRY	unset	name	Parsed boundary-geometry compatibility attribute; not connected by the current update path.
RMIN, RMAX, ANGLE, PDIS, GAPWIDTH, PHI0	unset	mm or deg	Cyclotron-cavity compatibility inputs; not used by ordinary PARALLEL RF-field application.
PHASE_MODEL, AMPLITUDE_MODEL, FREQUENCY_MODEL	unset	name	Names are stored, but ordinary RFCAVITY tracking does not currently resolve them into time-dependence objects.

The instantaneous map scale is set by the configured amplitude and $\cos(2\pi ft + \mathrm{LAG})$; the magnetic map component uses the corresponding sine factor. OPTION.AUTOPHASE>0 searches the reference-particle phase for maximum exit energy, using the option value as the number of refinement passes.

12.9 TRAVELINGWAVE

TRAVELINGWAVE applies a mapped traveling-wave RF structure. It shares the autophase controls with RFCAVITY and additionally describes the repeated-cell geometry.

```
TW1: TRAVELINGWAVE,
    L=2.8, ELEMEDGE=0.5,
    VOLT=15.0, FREQ=2998.0, LAG=0.0,
    FMAPFN="traveling-wave.T7",
    NUMCELLS=84, MODE=0.3333333333;
```

Parameter	Default	Unit	Current behavior
VOLT, DVOLT	unset	MV/m	Nominal field-map amplitude and additive error.
FREQ	unset	MHz	RF frequency converted internally to angular frequency.
LAG, DLAG	unset	rad	Nominal phase and additive phase error.
FMAPFN	unset	path	Required traveling-wave field map.
FAST	TRUE	Boolean	Passed to the field-map loader.
APVETO	FALSE	Boolean	Excludes the structure from automatic phase optimization when true.
NUMCELLS	unset	cells	Number of cells, converted from real input to an integer. Use a positive integer.
MODE	1/3	turns	RF phase advance between neighboring cells, expressed as a fraction of 2π.

Parameter	Default	Unit	Current behavior
DESIGNENERGY	-1	MeV	Positive autophaser target exit kinetic energy; negative disables the target.

Use a regression-matched map, cell count, and mode as one consistent set. As with RFCAVITY, AUTOPHASE=0 disables phase searching globally and APVETO=TRUE disables it for this element only.

12.10 LASER

LASER defines a passive analytic laser pulse. It is registered by the current source but is not exercised by the maintained regression suite.

```
L1: LASER, L=0.0, ELEMEDGE=0.0,
    WAVELENGTH=800e-9, PULSEENERGY=1.0,
    PULSELENGTH=30e-15, WAISTX=20e-6, WAISTY=20e-6,
    DIR={0,0,-1}, STOKES={1,0,0};
```

Parameter	Unit	Constraint
WAVELENGTH	m	Required and greater than zero.
PULSEENERGY	J	Required and greater than zero.
PULSELENGTH	s	Required and greater than zero.
WAISTX, WAISTY	m	Required and greater than zero.
DIR	none	Required nonzero three-vector; OPALX normalizes it.
STOKES	none	Optional three-vector with $\xi_1^2 + \xi_2^2 + \xi_3^2 \leq 1$.
L	m	Optional non-negative body length.

12.11 MONITOR

MONITOR records bunch crossings at a lattice location.

```
M1: MONITOR, L=0.01, ELEMEDGE=0.5, TYPE=SPATIAL, OUTFN="monitor.h5";
```

Parameter	Default	Current behavior
`L`	at least `0.01` m at update	Physical collection interval.
`TYPE`	non-`TEMPORAL`	`TEMPORAL` selects temporal collection; every other value selects spatial collection.
`OUTFN`	unset	Output filename passed to the monitor implementation.

12.12 MARKER

`MARKER` identifies a lattice location without adding a field. It accepts the common placement attributes. Built-in `#S` and `#E` markers denote the start and end of a line.

```
IP: MARKER, ELEMEDGE=1.2;
```

12.13 PROBE

`PROBE` records particle crossings over a rectangular span.

```
P1: PROBE, L=0.001, ELEMEDGE=0.8,
    XSTART=-10, XEND=10, YSTART=-10, YEND=10,
    STEP=1, OUTFN="probe.h5";
```

Parameter	Unit	Current behavior
`XSTART`, `XEND`	mm	Horizontal endpoints, converted to metres.
`YSTART`, `YEND`	mm	Vertical endpoints, converted to metres.
`STEP`	mm	Sampling step; default `1`.
`WIDTH`	mm	Accepted but not used by the current update path.
`OUTFN`	path	Output filename.

12.14 RBEND and SBEND

RBEND defines rectangular bend geometry whose L is the straight body length. SBEND defines sector geometry whose L is the design-orbit arc length. Both apply normal and skew components through octupole order.

```
B1: SBEND, L=0.5, ELEMEDGE=1.0, ANGLE=0.1,
    K1=0.0, HGAP=0.02, FINT=0.5;
```

Parameter	Unit	Current behavior
ANGLE	rad	Requested bend angle.
K0, K0S	m^{-1}	Normal and skew dipole coefficients. If K0 is omitted, OPALX derives the normal dipole from ANGLE and geometry.
K1, K1S	m^{-2}	Normal and skew quadrupole coefficients.
K2, K2S	m^{-3}	Normal and skew sextupole coefficients.
K3, K3S	m^{-4}	Normal and skew octupole coefficients.
HGAP	m	Half gap; OPALX stores twice this value as the full gap.
FINT	none	Fringe integral; default 0.5.
HAPERT	m	Positive values install a rectangular transverse aperture.
DESIGNENERGY	eV	Optional design energy forwarded to the bend.
E1, E2	rad	Parsed but explicitly rejected by the current native bend port.
GAP, GREATERTHANPI	-	Rejected compatibility attributes. Use HGAP; bends beyond the old compatibility path are unsupported.
WAKEF, PARTICLEMATTERINTERACTION	name	Rejected by the current native bend port.

The classes are registered but currently lack maintained regression inputs. Treat pole-

face, fringe, aperture, and sign conventions as experimental until validated for the intended lattice.

12.15 VERTICALFFAMAGNET

VERTICALFFAMAGNET defines a vertical fixed-field alternating-gradient magnet with a tanh longitudinal fringe.

Parameter	Unit	Meaning
B0	T	Nominal dipole field at zero height.
FIELD_INDEX	m^{-1}	Exponential vertical field index.
WIDTH	m	Full horizontal aperture width.
MAX_HORIZONTAL_POWER	integer	Highest horizontal expansion power.
END_LENGTH	m	Tanh fringe length.
CENTRE_LENGTH	m	Flat-top length.
BB_LENGTH	m	Bounding-box length.
HEIGHT_POS_EXTENT, HEIGHT_NEG_EXTENT	m	Positive and negative vertical aperture extents.

This registered element has no maintained OPALX regression input; validate its placement and field before production use.

12.16 VARIABLE_RF_CAVITY

VARIABLE_RF_CAVITY applies a hard-edge electric field whose phase, amplitude, and frequency are supplied by named time-dependence definitions.

```
VC1: VARIABLE_RF_CAVITY, L=0.2, ELEMEDGE=0.0,
     PHASE_MODEL=Phase, AMPLITUDE_MODEL=Amplitude,
     FREQUENCY_MODEL=Frequency, WIDTH=0.1, HEIGHT=0.1;
```

Parameter	Expected model output
PHASE_MODEL	Phase in radians.
AMPLITUDE_MODEL	Electric field in MV/m.
FREQUENCY_MODEL	Frequency in MHz.
WIDTH, HEIGHT, L	Full cavity dimensions in metres.

12.16.1 Time-dependence definitions

`POLYNOMIAL_TIME_DEPENDENCE` Defines $f(t) = \sum_i c_i t^i$ with time in ns. Use either `P0` through `P3` or `COEFFICIENTS`, not both.

`SINUSOIDAL_TIME_DEPENDENCE` Sums components defined by `FREQUENCIES`, optional `PHASE_OFFSETS`, `AMPLITUDES`, and `DC_OFFSETS`. Arrays must describe compatible component counts.

`SPLINE_TIME_DEPENDENCE` Interpolates `VALUES` at strictly increasing `TIMES` in ns. `ORDER=1` selects linear interpolation and `ORDER=3` selects cubic interpolation with quadratic smoothing. The `ORDER` attribute accepts only those two values.

12.17 Current limitations

- Registration establishes accepted input syntax, not numerical validation. Elements without a maintained regression input are identified above.
- `QUADRUPOLE` and `MULTIPOLE` strength-error inputs are parser-visible but do not affect the current field kernels.
- The current bulk-particle `MULTIPOLE` kernel applies dipole and quadrupole components only, while its host/reference kernel also evaluates higher orders. Do not use higher-order `MULTIPOLE` arrays for production tracking.
- Field-map elements require their external maps; generated maps and large binary support data belong outside the manual repository.
- Establish step-size, placement, aperture, and field-map convergence for each production lattice rather than treating a regression configuration as an accuracy prescription.

OPAL

The following retained OPAL material covers the non-OPAL-CYCL element interfaces from the legacy manual. It is not a statement that these types or attributes are accepted by OPALX.

This chapter is the first mixed migration chapter in the Quarto pilot. It combines:

- legacy OPAL element semantics ported from the Asciidoctor manual
- OPALX-native analytic element descriptions already written directly in Quarto

12.18 Element Input Format

All physical elements are defined by statements of the form

```
label:keyword, attribute,..., attribute
```

where:

label is the name given to the element, for example QF. It is an identifier.
keyword is an element-type keyword, for example QUADRUPOLE.
attribute usually has the form

```
attribute-name=attribute-value
```

Omitted attributes are assigned a default value, normally zero.

Example:

```
QF: QUADRUPOLE, L=1.8, K1=0.015832;
```

12.19 Common Attributes for All Elements

The following attributes are shared by all elements.

Attribute	Meaning
TYPE	Engineering type string used for element selection.
APERTURE	Aperture specification such as SQUARE(a,f), RECTANGLE(a,b,f), CIRCLE(d,f), or ELLIPSE(a,b,f).
L	Physical element length.
ELEMEDGE	Position of the physical start of the element in s.
X, Y, Z	Absolute position components when absolute placement is used.
THETA, PHI, PSI	Orientation angles relative to the first beamline using absolute positioning.
DX, DY, DZ	Position errors that do not affect the design trajectory.
DTHETA, DPHI, DPSI	Rotation errors that do not affect the design trajectory.
WAKEF	Attach a wakefield defined by the WAKE command.
PARTICLEMATTERINTERACTION	Attach a particle-matter interaction handler.
OUTFN	Base file name for element-generated output.
DELETEONTRANSVERSEEXIT	Control whether particles exiting on the transverse sides are deleted in OPAL-T.

Dipoles are a special case because GAP and HGAP define their aperture instead of the generic APERTURE attribute.

Default aperture for all other elements is a circle of `1E6 m`.

12.20 Drift Spaces

```
label: DRIFT, TYPE=string, APERTURE=string, L=real;
```

A `DRIFT` space has no additional attributes.

Examples:

```
DR1: DRIFT, L=1.5;
DR2: DRIFT, L=DR1->L, TYPE=DRF;
```

The length of `DR2` will always be equal to the length of `DR1`. The reference system for a drift space is a Cartesian coordinate system.

12.21 Quadrupole

```
label: QUADRUPOLE, TYPE=string, APERTURE=real-vector,
      L=real, K1=real, K1S=real;
```

A `QUADRUPOLE` is a straight Cartesian magnet with normal and skew quadrupole components.

`K1` normal quadrupole gradient $K_1 = \partial B_y/\partial x$ in Tm^{-1}. Positive `K1` means horizontal focusing for positively charged particles moving in the positive design direction.
`K1S` skew quadrupole component $K_{1s} = -\partial B_x/\partial x$ in Tm^{-1}.
`DK1, DK1S` normalized strength errors for the normal and skew component.

Example:

```
QP1: QUADRUPOLE, L=1.20, ELEMEDGE=-0.5265, K1=0.11;
```

12.22 Sextupole

```
label: SEXTUPOLE, TYPE=string, APERTURE=real-vector,
       L=real, K2=real, K2S=real;
```

A `SEXTUPOLE` is a straight Cartesian element with second-order normal and skew components.

`K2` normal sextupole strength $K_2 = \partial^2 B_y/\partial x^2$ in Tm^{-2}.
`K2S` skew sextupole strength $K_{2s} = -\partial^2 B_x/\partial x^2$ in Tm^{-2}.
`DK2, DK2S` normalized strength errors.

Example:

```
S1: SEXTUPOLE, L=0.4, K2=0.00134;
```

12.23 Octupole

```
label: OCTUPOLE, TYPE=string, APERTURE=real-vector,
      L=real, K3=real, K3S=real;
```

`K3` normal octupole strength $K_3 = \partial^3 B_y/\partial x^3$ in Tm^{-3}.
`K3S` skew octupole strength $K_{3s} = -\partial^3 B_x/\partial x^3$ in Tm^{-3}.
`DK3, DK3S` normalized strength errors.

Example:

```
O3: OCTUPOLE, L=0.3, K3=0.543;
```

12.24 General Multipole

The legacy `MULTIPOLE` element defines a thick straight multipole of arbitrary order.

```
label: MULTIPOLE, TYPE=string, APERTURE=real-vector,
      L=real, KN=real-vector, KS=real-vector;
```

`KN` array of normal coefficients $K_n = \partial^n B_y/\partial x^n$.
`KS` array of skew coefficients $K_{ns} = -\partial^n B_x/\partial x^n$.
`DKN, DKS` normalized strength-error arrays for the normal and skew coefficients.

If `L != 0`, the strengths are interpreted per unit length. If `L == 0`, they are integrated strengths. The multipole order of coefficient `n` corresponds to `2n+2` poles. Superposition of several orders is allowed.

Example:

```
M27: MULTIPOLE, L=1.0, ELEMEDGE=3.8, KN={0.0, 0.11};
```

12.25 General Multipole With Fringe Field Model

`MULTIPOLET` is the extended cyclotron-side multipole element with explicit fringe fields. It can represent either a straight magnet or a curved magnet with a prescribed bend angle.

```
label: MULTIPOLET, L=real, TP=real-vector,
       LFRINGE=real, RFRINGE=real,
       VAPERT=real, HAPERT=real,
       MAXFORDER=real, ROTATION=real,
       EANGLE=real, BBLENGTH=real, ANGLE=real,
       MAXXORDER=real, VARRADIUS=bool,
       ENTRYOFFSET=real, SCALING_MODEL=string;
```

The most important attributes are:

`TP` transverse mid-plane polynomial coefficients $T(x) = B_0 + B_1 x + B_2 x^2 + \ldots$.
`LFRINGE, RFRINGE` left and right fringe-field lengths.
`ANGLE` physical bend angle of the magnet.
`VARRADIUS` when `TRUE`, the bend radius can vary so that the reference path remains in the magnet center. This is a cyclotron-side restricted feature.
`ROTATION` roll of a straight magnet about its central axis to obtain skew fields.
`SCALING_MODEL` optional time-dependent scaling model for the multipole coefficients.

This element uses a fringe-field model based on a scalar-potential expansion with a mid-plane profile multiplied by a longitudinal fringe function.

Example:

```
M30: MULTIPOLET, L=1, RFRINGE=0.3, LFRINGE=0.2,
     ANGLE=PI/6, TP={2.0, 0.1}, VARRADIUS=TRUE, BBLENGTH=2;
```

12.26 Solenoid

```
label: SOLENOID, TYPE=string, APERTURE=real-vector,
      L=real, KS=real;
```

A SOLENOID has one principal strength attribute:

KS the solenoid strength $K_s = \frac{\partial B_s}{\partial s}$, in Tm^{-1}. For positive KS and positive particle charge, the solenoid field points in the direction of increasing s.

The reference system for a solenoid is a Cartesian coordinate system.

In OPAL-T mode, field maps are specified through:

FMAPFN file name of the field map.

Example:

```
SP1: SOLENOID, L=1.20, ELEMEDGE=-0.5265, KS=0.11,
     FMAPFN="1T1.T7";
```

12.27 RF Cavities

An RFCAVITY is the legacy standing-wave cavity element used in both beamline and cyclotron workflows. The common syntax is:

```
label: RFCAVITY, APERTURE=real-vector, L=real,
       VOLT=real, LAG=real;
```

L cavity length in m.
VOLT peak RF voltage in MV. The cavity kick is parameterized as $\delta E = \mathrm{VOLT}\cdot\sin(2\pi(\mathrm{LAG} - \mathrm{HARMON}\cdot f_0 t))$.
LAG phase lag in rad. In OPAL-T this is generally measured relative to the phase that maximizes the reference-particle energy gain. That phase is found by the auto-phasing algorithm unless the cavity is vetoed.
DLAG phase-lag error in rad.

12.27.1 OPAL-T mode

Additional OPAL-T attributes are:

FMAPFN field-map file in T7 format.
TYPE cavity type, either STANDING (default) or SINGLEGAP.
FREQ cavity frequency in MHz. If both the cavity card and the field map define a frequency, OPAL-T warns on mismatch and uses the card value.
APVETO if TRUE, the cavity is excluded from auto-phasing and the phase equals LAG at the arrival time of the reference particle at the field boundary.

Example standing-wave cavity that mimics a DC gun:

```
gun: RFCAVITY, L=0.018, VOLT=-131/(1.052*2.658),
     FMAPFN="1T3.T7", ELEMEDGE=0.00,
     TYPE=STANDING, FREQ=1.0e-6;
```

Example two-frequency standing-wave system:

```
rf1: RFCAVITY, L=0.54, VOLT=19.961, LAG=193.0/360.0,
     FMAPFN="1T3.T7", ELEMEDGE=0.129, TYPE=STANDING,
     FREQ=1498.956;

rf2: RFCAVITY, L=0.54, VOLT=6.250, LAG=136.0/360.0,
     FMAPFN="1T4.T7", ELEMEDGE=0.129, TYPE=STANDING,
     FREQ=4497.536;
```

12.28 Traveling Wave Structure

A `TRAVELINGWAVE` structure is an OPAL-T element reconstructed from a one-dimensional standing-wave field map and a repeated-cell model.

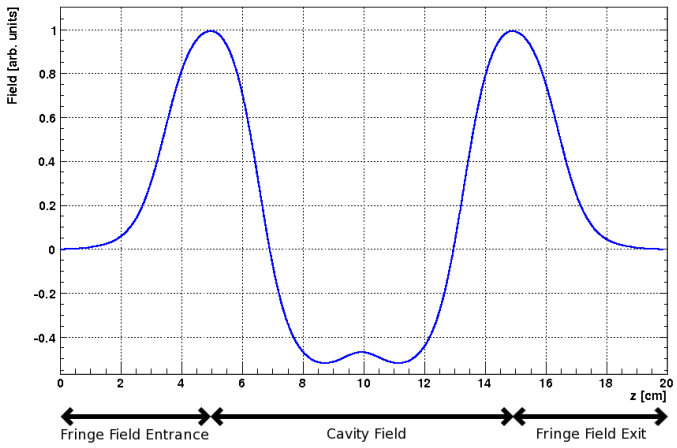


Figure 12.1: The on-axis field of an S-band `TRAVELINGWAVE` structure. The field of a single cavity is shown between its entrance and exit fringe fields.

An example of a 1D traveling-wave field map is shown in Figure 12.1. The map itself is a standing-wave solution for a single accelerating cavity. OPAL-T reconstructs the full traveling-wave tank in three parts:

1. entrance fringe field,
2. repeated accelerating-cell region,
3. exit fringe field.

12.28.1 Fringe fields

The entrance and exit fringe fields are treated as standing-wave regions:

$$\mathbf{E}_{\text{entrance}}(\mathbf{r},t) = \mathbf{E}_{\text{from map}}(\mathbf{r})\,\text{VOLT}\,\cos\!\big(2\pi\,\text{FREQ}\,t + \phi_{\text{entrance}}\big),$$

$$\mathbf{E}_{\text{exit}}(\mathbf{r},t) = \mathbf{E}_{\text{from map}}(\mathbf{r})\,\text{VOLT}\,\cos\!\big(2\pi\,\text{FREQ}\,t + \phi_{\text{exit}}\big).$$

Here `VOLT` and `FREQ` are the element amplitude and frequency, the entrance phase is `\phi_{entrance} = \mathrm{LAG}`, and the exit phase is determined internally from `LAG` and `NUMCELLS`.

12.28.2 Reconstructed body field

The field of the accelerating structure body is reconstructed from the central part of the standing-wave map by superposition:

$$\mathbf{E}(\mathbf{r},t) = \frac{\text{VOLT}}{\sin(2\pi\,\text{MODE})}\Big[\mathbf{E}_{\text{from map}}(x,y,z)\cos\!\big(2\pi\,\text{FREQ}\,t + \text{LAG} + \tfrac{\pi}{2}\,\text{MODE}\big)$$

$$+\mathbf{E}_{\text{from map}}(x,y,z+d)\cos\!\big(2\pi\,\text{FREQ}\,t + \text{LAG} + \tfrac{3\pi}{2}\,\text{MODE}\big)\Big].$$

The cell spacing is

$$d = \lambda\,\text{MODE},$$

where `MODE` is expressed in units of `2`. When the longitudinal coordinate advances past the end of the available map interval, OPAL-T subtracts an integer number of cavity wavelengths until the position falls back inside the repeated-cell range.

The legacy model therefore treats the entry and exit half cells through the standing-wave fringe fields and the interior through a repeated-cell traveling wave assembled from the same on-axis map.

The element attributes are:

```
label: TRAVELINGWAVE, APERTURE=real-vector, L=real,
       VOLT=real, LAG=real, FMAPFN=string,
       ELEMEDGE=real, FREQ=real, NUMCELLS=integer,
       MODE=real;
```

`L` cavity length. In OPAL-T the effective structure length is reconstructed from the field map and `NUMCELLS`, so `L` is not the primary geometric input.

`VOLT` peak RF voltage. The accelerating phase is again expressed by `LAG`.

`LAG` phase lag in `rad`. In OPAL-T this is normally defined relative to the phase that maximizes energy gain of the reference particle, as determined by the auto-phasing algorithm.

DLAG phase-lag error in `rad`.
FMAPFN field map in `T7` format.
FREQ traveling-wave frequency in `MHz`. As for `RFCAVITY`, OPAL-T warns when this differs from the frequency declared in the field map, and the map value takes precedence.
NUMCELLS number of accelerating cells, excluding entry and exit half-cell fringes.
MODE mode in units of `2` ; for example `1/3` stands for a `2 /3` structure.
FAST if `TRUE` and the field map is 1D, OPAL-T builds a 2D interpolation map and avoids an FFT-based evaluation on every particle and every step.
APVETO disable auto-phasing for this structure and use the user-supplied `LAG` directly at the arrival time of the reference particle at the field boundary.

The traveling-wave model requires the particle momentum `P` and charge `CHARGE` to be defined on the relevant optics command before the structure is used.

Example:

```
lrf0: TRAVELINGWAVE, L=0.0253, VOLT=14.750,
      NUMCELLS=40, ELEMEDGE=2.73066,
      FMAPFN="INLB-02-RAC.Ez", MODE=1/3,
      FREQ=1498.956, LAG=248.0/360.0;
```

12.29 Monitor

A `MONITOR` detects all particles passing it and writes the position, momentum, and time of impact into an H5hut file. It always has an effective length of 1 cm, consisting of a 0.5 cm drift, the monitor of zero length, and another 0.5 cm drift. This prevents OPAL-T from missing particles.

The positions of the particles on the monitor are interpolated from the current position and momentum one step before they would pass the monitor.

OUTFN file name into which the monitor writes the collected data.

If `TYPE=TEMPORAL`, the data of all particles are written when the reference particle hits the monitor.

12.30 Collimators

Four collimator families exist in the legacy manual:

- `ECOLLIMATOR`: elliptic aperture
- `RCOLLIMATOR`: rectangular aperture
- `FLEXIBLECOLLIMATOR`: arbitrary aperture description
- `CCOLLIMATOR`: cyclotron-specific radial rectangular collimator

The common OPAL-T beamline syntax is:

```
label: ECOLLIMATOR, TYPE=string, APERTURE=real-vector,
       L=real, XSIZE=real, YSIZE=real;

label: RCOLLIMATOR, TYPE=string, APERTURE=real-vector,
       L=real, XSIZE=real, YSIZE=real;

label: FLEXIBLECOLLIMATOR, APERTURE=real-vector,
       L=real, DESCRIPTION=string, FNAME=string, OUTFN=string;
```

Common attributes:

OUTFN file name for lost-particle output. If omitted, the element label is used. The output is written as H5hut, or ASCII when **ASCIIDUMP** is enabled.

PARTICLEMATTERINTERACTION activates scattering and energy-loss modeling when configured.

12.30.1 OPAL-T mode

Optically, a collimator behaves like a drift space, but during tracking it also introduces an aperture limit. The aperture is checked at the entrance, and if the collimator has nonzero length it is also checked at the exit and at every timestep.

XSIZE horizontal half-aperture in **m**.
YSIZE vertical half-aperture in **m**.

For **ECOLLIMATOR**, the values are ellipse half-axes; for **RCOLLIMATOR**, they are rectangle half-widths.

Example:

```
Col: ECOLLIMATOR, L=1.0E-3, ELEMEDGE=3.0E-3, XSIZE=5.0E-4,
     YSIZE=5.0E-4, OUTFN="Coll.h5";
```

FLEXIBLECOLLIMATOR extends this with a small shape-description language. The supported primitive and composition commands are:

rectangle(width, height) rectangle centered at the origin.
ellipse(width, height) ellipse centered at the origin.
polygon(x0, y0; x1, y1; ...; xN, yN) polygon whose final point is implicitly connected back to the first point. The polygon is triangulated internally, so self-crossing edges are not allowed.
mask('filename.pbm', width, height) black-and-white **PBM** bitmap mask. White pixels stop particles.
translate(command, shiftx, shifty) translate a shape by **(shiftx, shifty)**.

`rotate(command, angle)` rotate a shape about the origin.
`union(command1, command2, ...)` union of several shape definitions.
`difference(command1, command2)` particles pass `command1` but not `command2`.
`symmetric_difference(command1, command2)` particles pass either command but not both at the same time.
`intersection(command1, command2)` particles pass both commands simultaneously.
`repeat(command, N, shiftx, shifty)` repeat a shape `N` times with a translational offset.
`repeat(command, N, angle)` repeat a shape `N` times with successive rotation.

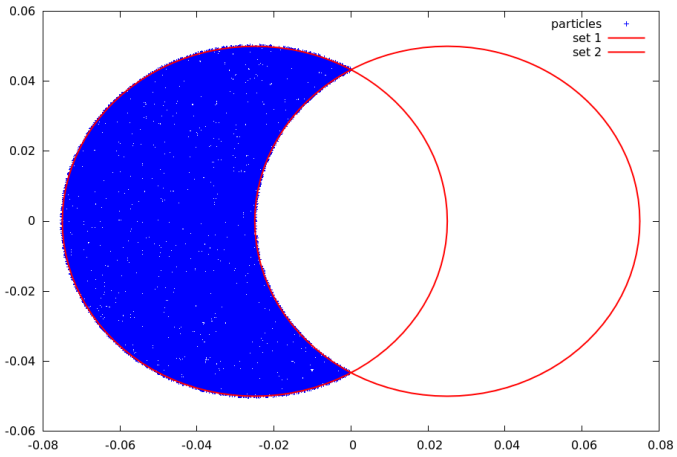


Figure 12.2: Illustration of a difference between two circles.

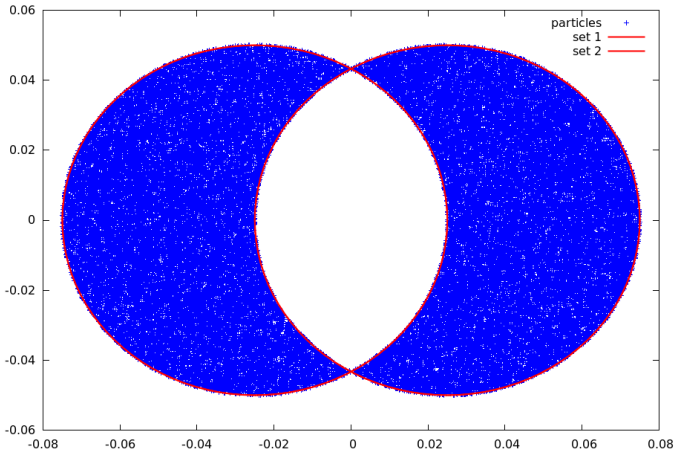


Figure 12.3: Illustration of a symmetric difference between two circles.

The expression language supports the following constants and functions:

Constants and functions	Constants and functions	Constants and functions	Constants and functions
`e`	`pi`	`abs(x)`	`acos(x)`

Constants and functions	Constants and functions	Constants and functions	Constants and functions
`acosh(x)`	`asin(x)`	`asinh(x)`	`atan(x)`
`atanh(x)`	`cbrt(x)`	`ceil(x)`	`cos(x)`
`cosh(x)`	`deg2rad(x)`	`erf(x)`	`erfc(x)`
`exp(x)`	`exp2(x)`	`floor(x)`	`isinf(x)`
`isnan(x)`	`log(x)`	`log2(x)`	`log10(x)`
`rad2deg(x)`	`round(x)`	`sgn(x)`	`sin(x)`
`sinh(x)`	`sqrt(x)`	`tan(x)`	`tanh(x)`
`tgamma(x)`	`atan2(y,x)`	`max(x,y)`	`min(x,y)`
`pow(x,n)`			

A simple elliptic collimator can be described by:

```
ellipse(0.04, 0.03)
```

A regular pepper-pot with rectangular holes can be described by:

```
repeat(
       repeat(
              translate(
                        rotate(
                               rectangle(
                                         0.002,
                                         0.002
                                        ),
                               0.78539
                              ),
                        -0.028,
                        -0.028
                       ),
              16,
              0.004,
              0.0
             ),
       16,
       0.0,
       0.004
      )
```

In `FLEXIBLECOLLIMATOR`, the hole description can be supplied directly through `DESCRIPTION` or through an external file specified by `FNAME`. The external file form is the one that allows comments and line breaks.

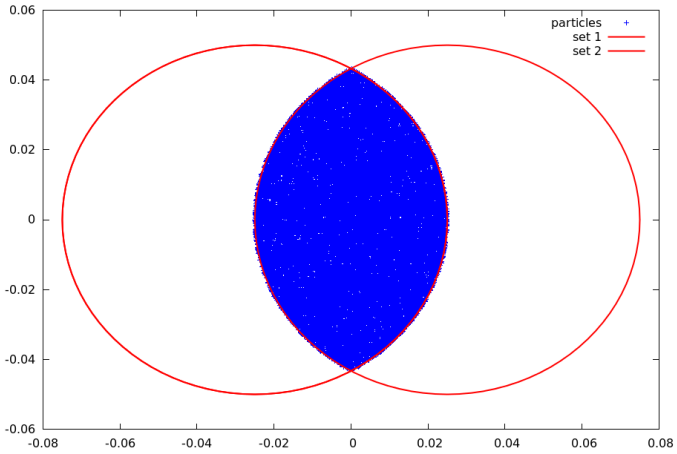


Figure 12.4: Illustration of an intersection between two circles.

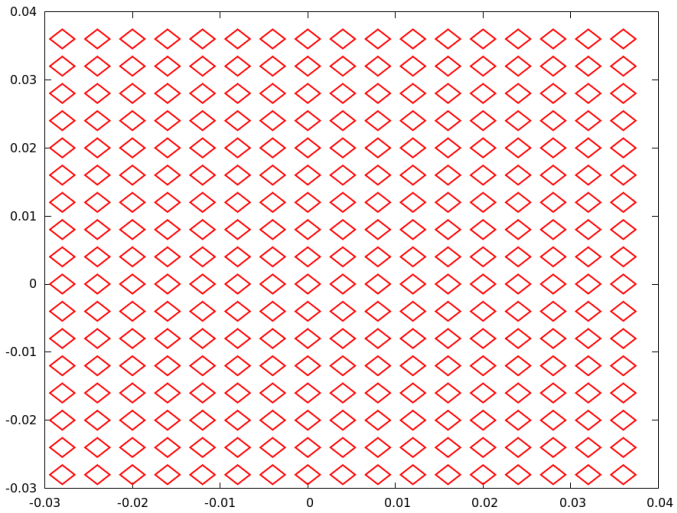


Figure 12.5: Pepper-pot with rectangular holes.

12.31 Correctors

Three types of correctors are available:

- HKICKER: horizontal corrector
- VKICKER: vertical corrector
- KICKER: combined horizontal and vertical corrector

They are defined as:

```
label: HKICKER, TYPE=string, APERTURE=real-vector,
      L=real, KICK=real;
label: VKICKER, TYPE=string, APERTURE=real-vector,
      L=real, KICK=real;
label: KICKER, TYPE=string, APERTURE=real-vector,
      L=real, HKICK=real, VKICK=real;
```

Attributes:

KICK kick angle in **rad** for HKICKER or VKICKER.
HKICK, VKICK horizontal and vertical kick angles for KICKER.
DESIGNENERGY if set, fixes the magnetic field magnitude using the given design energy in MeV and the requested kick angle.

A positive kick increases p_x or p_y respectively.

Examples:

```
HK1: HKICKER, KICK=0.001;
VK3: VKICKER, KICK=0.0005;
KHV: KICKER,  HKICK=0.001, VKICK=0.0005;
```

Correctors do not change the reference trajectory. Otherwise they are implemented like RBEND with E1 = 0 and without fringe fields.

12.32 Bending Magnets

Legacy OPAL supports four bend families:

- RBEND for rectangular bends in OPAL-T
- RBEND3D for rectangular bends with field-map support in OPAL-T
- SBEND for sector bends in OPAL-T

RBEND and SBEND share the same basic dipole semantics but differ in their reference geometry. RBEND has parallel pole faces and an exit angle fixed by ANGLE - E1; SBEND uses a sector reference arc and supports independent entrance and exit edge angles.

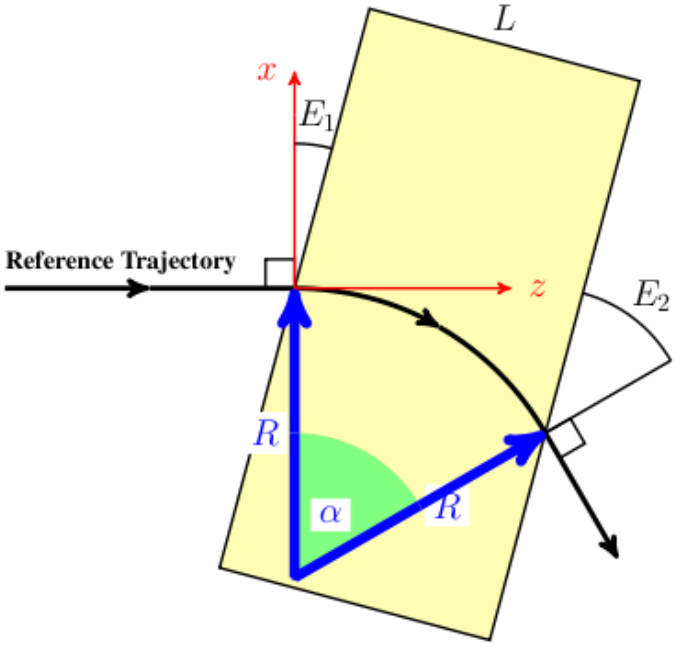


Figure 12.6: Rectangular bend geometry with positive bend angle and positive entrance edge angle.

12.32.1 RBEND

Key attributes are:

- L: physical magnet length
- GAP: full vertical gap
- HAPERT: non-bend-plane aperture
- ANGLE: bend angle in rad
- K0, K0S: normal and skew dipole field amplitudes in T
- E1: entrance edge angle
- DESIGNENERGY: reference-particle energy in MeV
- FMAPFN: fringe-field map, typically 1DProfile1

If ANGLE is set, it overrides K0 and OPAL adjusts the field strength to bend the design particle through the requested angle.

12.32.2 RBEND3D

RBEND3D keeps the rectangular bend geometry but uses explicit field-map support for the field model. The input attributes follow the same geometric meaning as RBEND, and the entrance/exit pole-face interpretation stays the same.

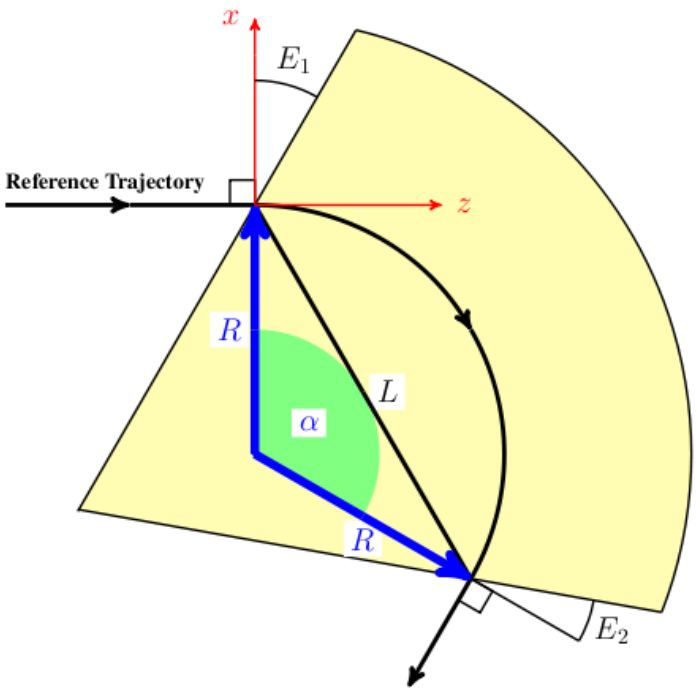


Figure 12.7: Sector bend geometry with independent entrance and exit edge angles.

12.32.3 SBEND

SBEND is the sector-bend version. The main difference is the interpretation of L: it is the chord length of the bend reference arc,

$$L = 2R\sin(\alpha/2)$$

and the element has both E1 and E2 as independent edge angles. Very large bend angles are possible, but once the entrance and exit fringe regions begin to overlap, the approximation becomes unreliable.

12.32.4 Bend setup rules in OPAL-T

The original manual emphasizes the following practical points:

1. Internally OPAL-T treats all bends as positive bends in the local reference frame and then uses ROTATION to orient them in the laboratory transverse plane.
2. If ANGLE is nonzero, K0 and K0S are ignored.
3. When ANGLE is used to define the bend strength, the actual beam bend still depends on whether the bunch energy matches DESIGNENERGY.
4. The hard-edge geometry defines the ideal reference trajectory, while the FMAPFN field map defines the soft-edge magnet seen by tracked particles.

5. If the default field map `1DPROFILE1-DEFAULT` is used, OPAL shifts the fringe-field origins so that the effective length of the soft-edge magnet matches the ideal hard-edge reference bend.

12.32.5 `RBEND` and `SBEND` examples

A simple `RBEND` driven by `ANGLE` and the default fringe-field map is:

```
Bend: RBEND, ANGLE = 30.0 * Pi / 180.0,
      FMAPFN = "1DPROFILE1-DEFAULT",
      ELEMEDGE = 0.25,
      DESIGNENERGY = 10.0E6,
      L = 0.5, GAP = 0.02;
```

For an electron design beam this produces a 30 degree positive bend, a 1 m reference radius, and a field amplitude of about `-0.0350195 T`. The important part of the diagnostic output is the final fringe-field integration report, which tells the user the actual bend angle of the reference particle through the soft-edge magnet.

The same bend can be defined through `K0` instead of `ANGLE`:

```
Bend: RBEND, K0 = -0.0350195,
      FMAPFN = "1DPROFILE1-DEFAULT",
      ELEMEDGE = 0.25,
      DESIGNENERGY = 10.0E6,
      L = 0.5, GAP = 0.02;
```

To numerical precision this reproduces the same geometry and field.

To bend in the opposite direction, the legacy manual gives four equivalent patterns:

```
Bend: RBEND, ANGLE = -30.0 * Pi / 180.0,
             FMAPFN = "1DPROFILE1-DEFAULT",
             ELEMEDGE = 0.25,
             DESIGNENERGY = 10.0E6,
             L = 0.5, GAP = 0.02;

Bend: RBEND, ANGLE = 30.0 * Pi / 180.0,
             FMAPFN = "1DPROFILE1-DEFAULT",
             ELEMEDGE = 0.25,
             DESIGNENERGY = 10.0E6,
             L = 0.5, GAP = 0.02,
             ROTATION = Pi;

Bend: RBEND, K0 = 0.0350195,
```

```
             FMAPFN = "1DPROFILE1-DEFAULT",
             ELEMEDGE = 0.25,
             DESIGNENERGY = 10.0E6,
             L = 0.5, GAP = 0.02;

Bend: RBEND, K0 = -0.0350195,
             FMAPFN = "1DPROFILE1-DEFAULT",
             ELEMEDGE = 0.25,
             DESIGNENERGY = 10.0E6,
             L = 0.5, GAP = 0.02,
             ROTATION = Pi;
```

The recommended style is to define the bend in the positive reference orientation and then use `ROTATION` to place it elsewhere in the transverse plane.

A representative four-dipole chicane setup is:

```
Bend1: RBEND, ANGLE = 20.0 * Pi / 180.0,
              E1 = 0.0,
              FMAPFN = "1DPROFILE1-DEFAULT",
              ELEMEDGE = 0.25,
              DESIGNENERGY = 10.0E6,
              L = 0.25,
              GAP = 0.02,
              ROTATION = Pi;

Bend2: RBEND, ANGLE = 20.0 * Pi / 180.0,
              E1 = 20.0 * Pi / 180.0,
              FMAPFN = "1DPROFILE1-DEFAULT",
              ELEMEDGE = 1.0,
              DESIGNENERGY = 10.0E6,
              L = 0.25,
              GAP = 0.02,
              ROTATION = 0.0;

Bend3: RBEND, ANGLE = 20.0 * Pi / 180.0,
              E1 = 0.0,
              FMAPFN = "1DPROFILE1-DEFAULT",
              ELEMEDGE = 1.5,
              DESIGNENERGY = 10.0E6,
              L = 0.25,
              GAP = 0.02,
              ROTATION = 0.0;

Bend4: RBEND, ANGLE = 20.0 * Pi / 180.0,
```

```
              E1 = 20.0 * Pi / 180.0,
              FMAPFN = "1DPROFILE1-DEFAULT",
              ELEMEDGE = 2.25,
              DESIGNENERGY = 10.0E6,
              L = 0.25,
              GAP = 0.02,
              ROTATION = Pi;
```

`SBEND` can be used in the same role once `L`, `E1`, and `E2` are interpreted using the sector-bend geometry.

12.32.6 Custom `1DProfile1` bend maps

The source manual distinguishes two `1DProfile1` usage modes:

1. the map defines both the fringe fields and the magnet length,
2. the map defines only the fringe fields and the element `L` defines the body length.

In the first mode, a custom map can define an `SBEND` without supplying `L` in order to preserve the exit fringe placement:

```
1DProfile1 1 1 2.0
 -10.0  0.0  10.0 1
  15.0  25.0 35.0 1
  0.00000E+00
  2.00000E+00
  0.00000E+00
  2.00000E+00
```

```
Bend: SBEND, ANGLE = 60.0 * Pi / 180.0,
             E1 = -10.0 * Pi / 180.0,
             E2 = 20.0 * Pi / 180.0,
             FMAPFN = "TEST-MAP.T7",
             ELEMEDGE = 0.25,
             DESIGNENERGY = 10.0E6,
             GAP = 0.02;
```

In the second mode, the field map contains only the entrance and exit fringe profiles and the magnet body is set by `L`:

```
1DProfile1 1 1 2.0
 -10.0  0.0  10.0 1
 -10.0  0.0  10.0 1
  0.00000E+00
```

```
  2.00000E+00
  0.00000E+00
  2.00000E+00
```

```
Bend: SBEND, ANGLE = 60.0 * Pi / 180.0,
             E1 = -10.0 * Pi / 180.0,
             E2 = 20.0 * Pi / 180.0,
             FMAPFN = "TEST-MAP.T7",
             ELEMEDGE = 0.25,
             DESIGNENERGY = 10.0E6,
             L = 0.25,
             GAP = 0.02;
```

With `ANGLE`, OPAL adjusts the field amplitude so that the reference particle is bent by the requested amount after the fringe fields are included. With `K0`, that compensation is not automatic:

```
Bend: SBEND, K0 = -0.1400778,
             E1 = -10.0 * Pi / 180.0,
             E2 = 20.0 * Pi / 180.0,
             FMAPFN = "TEST-MAP.T7",
             ELEMEDGE = 0.25,
             DESIGNENERGY = 10.0E6,
             L = 0.25,
             GAP = 0.02;
```

In the original example this produces a small bend mismatch because the soft-edge effective length no longer matches the ideal hard-edge reference magnet. The manual then corrects this by moving the entrance and exit Enge origins outward by `0.03026 mm`:

```
1DProfile1 1 1 2.0
 -10.0  -0.03026  10.0 1
 -10.0   0.03026  10.0 1
  0.00000E+00
  2.00000E+00
  0.00000E+00
  2.00000E+00
```

This increases the effective magnet length until the final tracked bend of the reference particle matches the desired 60 degree design bend again.

12.32.7 Bend fields from `1DProfile1`

For `RBEND` and `SBEND`, OPAL-T divides the field into three regions:

1. entrance fringe field
2. central field region
3. exit fringe field

The fringe regions are modeled from the `1DProfile1` Enge functions associated with `FMAPFN`.

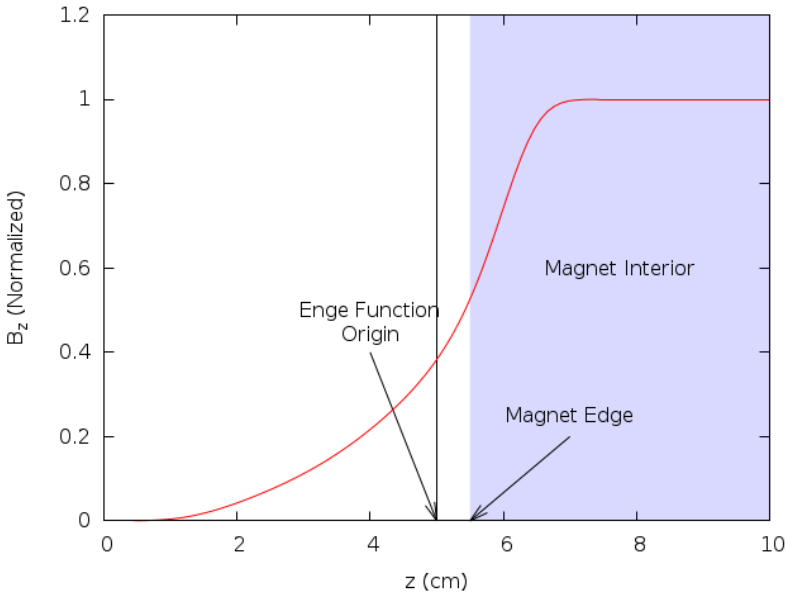


Figure 12.8: Entrance fringe field profile corresponding to the default Enge parameters.

The Enge function is

$$F(z) = \frac{1}{1 + e^{\sum_{n=0}^{N_{\text{order}}} c_n (z/D)^n}}$$

where `D` is the full magnet gap, `N_order` is the Enge order, and `z` is the perpendicular distance from the chosen fringe-field origin to the magnet face.

For a positive bend OPAL-T works with:

- `B_0`: dipole field amplitude,
- `R`: bend radius,
- `n = -(R/B_y)(\partial B_y/\partial x)`: field index set from `K1`,
- `F(\Delta_z)`: the entrance, central, or exit Enge profile depending on the particle position.

The transverse-local coordinates are:

- `y`: vertical offset from the magnet mid-plane,
- `\Delta_x`: perpendicular distance from the reference trajectory in the bend plane,
- `\Delta_z`: perpendicular distance from the entrance or exit fringe-field origin.

Starting from

$$\nabla \cdot \vec{B} = 0, \qquad \nabla \times \vec{B} = 0,$$

and defining

$$F' = \frac{dF}{dz}, \qquad F'' = \frac{d^2F}{dz^2}, \qquad F''' = \frac{d^3F}{dz^3},$$

one may write the exact off-axis field in terms of

$$\kappa(\Delta_z) = \sqrt{\frac{n^2}{R^2} + \frac{F''(\Delta_z)}{F(\Delta_z)}}.$$

Then

$$B_x(\Delta_x, y, \Delta_z) = -B_0 \frac{n}{R} e^{-n\Delta_x/R} \frac{F(\Delta_z)}{\kappa(\Delta_z)} \sin(\kappa(\Delta_z) y),$$

$$B_y(\Delta_x, y, \Delta_z) = B_0 e^{-n\Delta_x/R} \cos(\kappa(\Delta_z) y) F(\Delta_z),$$

$$B_z(\Delta_x, y, \Delta_z) = B_0 e^{-n\Delta_x/R} \left[\frac{F'(\Delta_z)}{\kappa(\Delta_z)} \sin(\kappa(\Delta_z) y) - \frac{1}{2\kappa(\Delta_z)} \left(F'''(\Delta_z) - \frac{F'(\Delta_z) F''(\Delta_z)}{F(\Delta_z)} \right) \left(\frac{\sin(\kappa(\Delta_z) y)}{\kappa(\Delta_z)^2} - y \right.\right.$$

For tracking, OPAL-T uses the small-`y` expansion up to `O(y^2)`.

In the fringe regions:

$$B_x \approx -B_0 \frac{n}{R} e^{-n\Delta_x/R} y,$$

$$B_y \approx B_0 e^{-n\Delta_x/R} \left[F(\Delta_z) - \left(\frac{n^2}{R^2} F(\Delta_z) + F''(\Delta_z) \right) \frac{y^2}{2} \right],$$

$$B_z \approx B_0 e^{-n\Delta_x/R} y F'(\Delta_z).$$

In the central region:

$$B_x \approx -B_0 \frac{n}{R} e^{-n\Delta_x/R} y,$$

$$B_y \approx B_0 e^{-n\Delta_x/R} \left(1 - \frac{n^2}{R^2} \frac{y^2}{2} \right),$$

$$B_z \approx 0.$$

These are the expressions OPAL-T uses internally after it determines whether a particle is in the entrance fringe, body, or exit fringe region.

12.32.8 Default bend field map

When `FMAPFN="1DPROFILE1-DEFAULT"`, OPAL uses built-in Enge coefficients for both the entrance and exit fringe regions. The default map corresponds to a 2 cm gap, which may be overridden by the element `GAP` attribute.

The coefficients are:

$$c_0 = 0.478959, \quad c_1 = 1.911289, \quad c_2 = -1.185953,$$
$$c_3 = 1.630554, \quad c_4 = -1.082657, \quad c_5 = 0.318111.$$

The equivalent explicit `1DProfile1` map is:

```
1DProfile1 5 5 2.0
 -10.0 0.0 10.0 1
 -10.0 0.0 10.0 1
  0.478959
  1.911289
 -1.185953
  1.630554
 -1.082657
  0.318111
  0.478959
  1.911289
 -1.185953
  1.630554
 -1.082657
  0.318111
```

12.33 Variable RF Cavities

`VARIABLE_RF_CAVITY` extends the hard-edge cavity model by making frequency, amplitude, and phase explicit functions of time:

```
FREQUENCY_MODEL=string, AMPLITUDE_MODEL=string, PHASE_MODEL=string
```

The field is modeled as

$$E(t) = a(t)\,\sin\!\left(2\pi \int_0^t f(\tau)\,d\tau + \phi(t)\right)$$

with no field outside the cavity boundary and no transverse dependence in the hard-edge model.

Important geometry attributes are `WIDTH`, `HEIGHT`, and `L`.

12.33.1 Time-dependence models

`POLYNOMIAL_TIME_DEPENDENCE` defines

$$g(t) = p_0 + p_1 t + p_2 t^2 + p_3 t^3,$$

either via `P0..P3` or a longer `COEFFICIENTS` array.

`SPLINE_TIME_DEPENDENCE` uses tabulated `TIMES` and `VALUES` with `ORDER=1` or `3`.

`SINUSOIDAL_TIME_DEPENDENCE` sums one or more sine waves from arrays `FREQUENCIES`, `AMPLITUDES`, `PHASE_OFFSETS`, and `DC_OFFSETS`.

12.33.2 Soft-edged variable cavities

For soft-edged cavities, `VARIABLE_RF_CAVITY_FRINGE_FIELD` adds:

- `CENTRE_LENGTH`
- `END_LENGTH`
- `CAVITY_CENTRE`
- `MAX_ORDER`

The intention is to place a full cavity including the central flat-top field region and explicit fringe-field ends. `VARIABLE_RF_CAVITY_FRINGE_FIELD` is the legacy soft-edge companion of `VARIABLE_RF_CAVITY`.

Example:

```
RF_FREQUENCY: POLYNOMIAL_TIME_DEPENDENCE, P0=rf_p0, P1=rf_p1, P2=rf_p2, P3=rf_p3;
RF_AMPLITUDE: POLYNOMIAL_TIME_DEPENDENCE, P0=1.0;
RF_PHASE: POLYNOMIAL_TIME_DEPENDENCE, P0=phi;

HARD_RF_CAVITY: VARIABLE_RF_CAVITY,
                PHASE_MODEL="RF_PHASE", AMPLITUDE_MODEL="RF_AMPLITUDE",
                FREQUENCY_MODEL="RF_FREQUENCY", L=0.100, HEIGHT=0.200, WIDTH=2.000;

SOFT_RF_CAVITY: VARIABLE_RF_CAVITY_FRINGE_FIELD,
                PHASE_MODEL="RF_PHASE", AMPLITUDE_MODEL="RF_AMPLITUDE",
                FREQUENCY_MODEL="RF_FREQUENCY", L=0.200, HEIGHT=0.200, WIDTH=2.000,
                CENTRE_LENGTH=0.1, END_LENGTH=0.01, CAVITY_CENTRE=0.1, MAX_ORDER=4;
```

12.34 Septum

`SEPTUM` is a cyclotron-side extraction element. Particles hitting the septum are removed from the bunch and written to the element output file.

Main geometric attributes:

- XSTART, XEND
- YSTART, YEND
- WIDTH

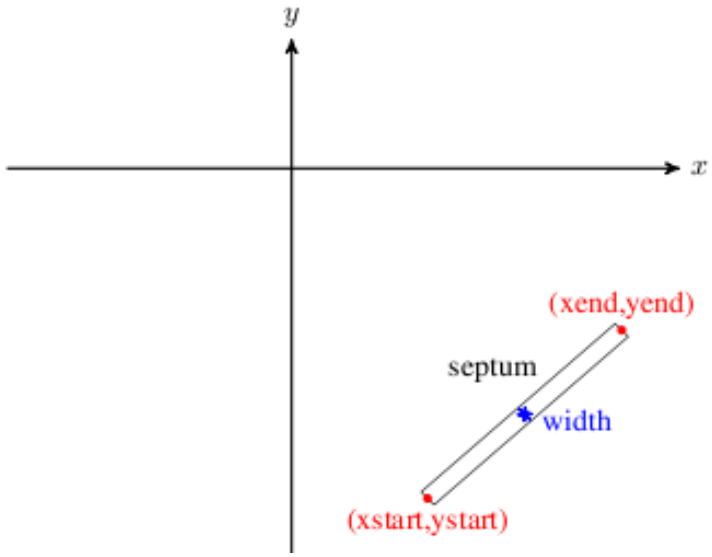


Figure 12.9: Cyclotron SEPTUM geometry.

Example:

```
eec2: SEPTUM, XSTART=4100.0, XEND=4300.0,
      YSTART=-1200.0, YEND=-150.0, WIDTH=0.05;
```

Lost particles are written to OUTFN.h5 or ASCII when ASCIIDUMP is enabled. ## Probe {#opal-probe}

PROBE records particles crossing a line-like detector while leaving them in the simulation.

Main attributes:

- XSTART, XEND
- YSTART, YEND
- STEP: histogram step size for the associated probe histogram and peak files

Example:

```
prob1: PROBE, XSTART=4166.16, XEND=4250.0,
       YSTART=-1226.85, YEND=-1241.3;
```

The hit particles are recorded in OUTFN.h5 (or ASCII with ASCIIDUMP). The same crossings are also accumulated into the .hist and .peaks files that mimic measured probe histograms and their derived peak locations. ## Output Plane {#opal-output-plane}

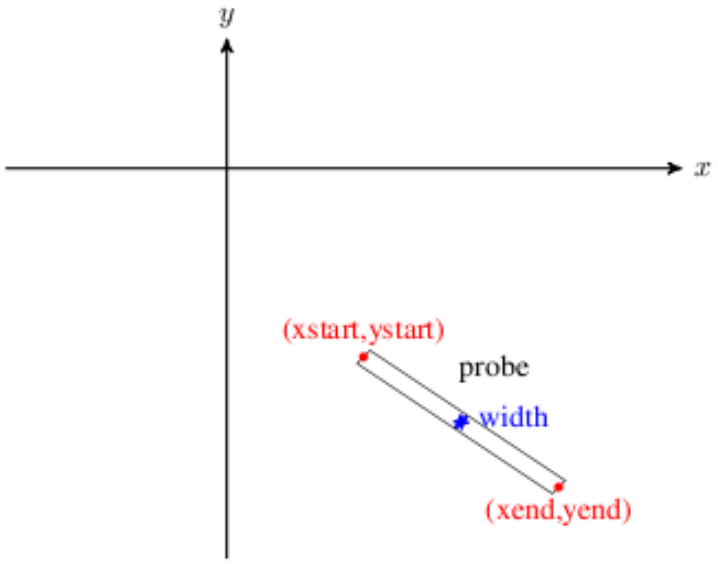


Figure 12.10: Cyclotron PROBE geometry.

OUTPUTPLANE records particles crossing a plane in space. Unlike PROBE, the plane can be defined either by a probe-style segment or by an explicit center and normal.

Two placement styles are available:

- PROBE: uses XSTART, XEND, YSTART, YEND
- CENTRE_NORMAL: uses CENTRE, NORMAL, together with optional WIDTH, HEIGHT, and RADIUS

Additional controls include:

- REFERENCE_ALIGNMENT_PARTICLE
- TOLERANCE
- ALGORITHM = INTERPOLATION | RK4
- VERBOSE

In CENTRE_NORMAL mode the plane may realign itself on the first crossing of REFERENCE_ALIGNMENT_PARTICLE so that the plane center and normal follow that reference trajectory.

Example:

```
outplane1: OUTPUTPLANE, PLACEMENT_STYLE=CENTRE_NORMAL,
           CENTRE={10, 13.5, 0}, NORMAL={1,0,0},
           WIDTH=1.5, HEIGHT=0.8, RADIUS=1,
           ALGORITHM=RK4, VERBOSE=1,
           REFERENCE_ALIGNMENT_PARTICLE=0, OUTFN="FileName.txt";
```

12.35 Stripper

STRIPPER is the cyclotron extraction element that changes particle charge and mass when a particle hits the stripping foil geometry.

Important attributes are:

- XSTART, XEND, YSTART, YEND
- OPCHARGE: output charge state
- OPMASS: output particle mass
- OPYIELD: number of outgoing particles per incoming particle
- STOP: if TRUE, stop the particle after stripping

The geometry matches the PROBE-style line segment. The struck particles are always written to the output file, regardless of STOP. The legacy manual also states explicitly that this element changes charge and mass but does not yet model the full microscopic stripping process.

Example:

```
prob1: STRIPPER, XSTART=4166.16, XEND=4250.0,
                 YSTART=-1226.85, YEND=-1241.3,
                 OPCHARGE=1, OPMASS=PMASS,
                 OPYIELD=2, STOP=FALSE;
```

12.36 Degrader

DEGRADER is an OPAL-T particle-matter interaction element with an elliptical transverse profile and length L.

Important attributes are:

- XSIZE, YSIZE: ellipse axes
- PARTICLEMATTERINTERACTION: attach scattering and energy-loss modeling

If no particle-matter interaction handler is attached, particles hitting the degrader are simply recorded and deleted.

Example:

```
DEGPHYS: PARTICLEMATTERINTERACTION, TYPE=SCATTERING, MATERIAL=GRAPHITE;
DEG1: DEGRADER, L=0.15, ELEMEDGE=0.02, PARTICLEMATTERINTERACTION=DEGPHYS;
```

12.37 Vacuum

VACUUM represents interactions with residual gas and, when configured, the associated stripping physics. When an interaction occurs, the particle state at that instant is written out.

MINR, and MAXR limits. In OPAL-T it is described by the usual L and ELEMEDGE placement parameters.

Important attributes are:

- PRESSURE: average residual-gas pressure in mbar
- TEMPERATURE: gas temperature in K
- GAS = H2 | AIR
- STOP: whether the interacting particle is deleted immediately
- PARTICLEMATTERINTERACTION

Cyclotron mode additionally supports a pressure map through:

- PMAPFN: mid-plane pressure-map file
- PSCALE: scale factor applied to that map

If PMAPFN is supplied, PRESSURE acts as the default value outside the map extent. For stripping studies, PARTICLEMATTERINTERACTION should usually point to a handler with TYPE=BEAMSTRIPPING.

Example:

```
bstp_phys: PARTICLEMATTERINTERACTION, TYPE=BEAMSTRIPPING;
vac: VACUUM, PRESSURE=1E-8, TEMPERATURE=300,
     GAS=H2, STOP=true, PARTICLEMATTERINTERACTION=bstp_phys;
```

The particles stripped in the residual gas are written to the configured output file regardless of the value of STOP. ## Undulator {#opal-undulator}

UNDULATOR couples the beamline to OPAL's full-wave FEL solver path. In the legacy implementation the field solve switches from Poisson to a Finite-Difference Time-Domain full-wave model while the bunch traverses the undulator. This path relies on the external MITHRA library.

To use this feature, OPAL must be built with MITHRA support:

1. install the MITHRA library,
2. set MITHRA_PREFIX to its installation path,
3. configure OPAL with -DENABLE_OPAL_FEL=yes.

When the head of the bunch crosses the undulator start, the solver switches to full-wave and returns to Poisson once the bunch has passed the undulator and its fringe fields.

The effective undulator length is determined by the number of periods and the built-in fringe extent,

$$L = N\lambda + 2\,\text{fringe}, \qquad \text{fringe} = 2\lambda,$$

so there is no separate user-supplied `L` parameter.

The idealized planar-undulator field is

$$B_x = B_0 \cosh(kr)\sin(kz)\cos(\alpha),$$

$$B_y = B_0 \cosh(kr)\sin(kz)\sin(\alpha),$$

$$B_z = B_0 \sinh(kr)\cos(kz),$$

with $k = 2\pi/\lambda$ and polarization angle α.

Important attributes are:

- `K`: undulator strength parameter
- `LAMBDA`: undulator period
- `NUMPERIODS`: number of periods
- `ANGLE`: polarization angle
- `MESHLENGTH`: computational-domain size in the moving bunch frame
- `MESHRESOLUTION`: computational-grid spacing
- `DTBUNCH`: particle update timestep
- `TRUNORDER`: absorbing-boundary truncation order
- `TOTALTIME`: total full-wave simulation time
- `FNAME`: optional job/output description file for the full-wave solver

Example:

```
UND: UNDULATOR, ELEMEDGE = 44.0e-2, K = 10.81, LAMBDA = 8.5e-2, NUMPERIODS = 10, ANGLE = P
     MESHLENGTH = { 12e-3, 12e-3, 4e-3 }, MESHRESOLUTION = { 1e-5, 1e-5, 8e-6},
     FNAME = "wiggler_sims_July_2020/mithra_output.job";
```

13 Field Output Commands

OPAL

OPAL provides two commands for writing external fields:

- `DUMPFIELDS`: write out a static magnetic field map on a Cartesian grid
- `DUMPEMFIELDS`: write out an electromagnetic field map on a 4D grid in space-time; Cartesian and cylindrical grids are supported

`DUMPEMFIELDS` is an extension of `DUMPFIELDS`.

OPALX

`OPALX` currently supports a single field-output command:

- `DUMPEMFIELDS`: write out the external electromagnetic field on a 4D grid in space-time; Cartesian and cylindrical grids are supported

OPAL

13.1 `DUMPFIELDS` Command

`DUMPFIELDS` writes the static magnetic field on a 3D Cartesian grid. The file format is:

```
<number of rows>
1 x [m]
2 y [m]
3 z [m]
4 Bx [kGauss]
5 By [kGauss]
6 Bz [kGauss]
0
<x0> <y0> <z0> <Bx0> <By0> <Bz0>
<x1> <y1> <z1> <Bx1> <By1> <Bz1>
...
```

Attributes:

Attribute	Meaning
FILE_NAME	output file name; existing files are overwritten
X_START, Y_START, Z_START	grid origin in m
DX, DY, DZ	grid spacing in m
X_STEPS, Y_STEPS, Z_STEPS	number of steps; must be positive integers

Example:

```
DUMPFIELDS, X_START=0., X_STEPS=101, DX=0.100,
            Y_START=0., Y_STEPS=101, DY=0.100,
            Z_START=0., Z_STEPS=1, DZ=0.100,
            FILE_NAME="FieldMapXY.dat";
```

This writes a mid-plane map in the x-y plane, starting at (0,0) with 101 x 101 samples and z = 0.

13.2 DUMPEMFIELDS Command

DUMPEMFIELDS writes electromagnetic field data on a 4D grid. Two coordinate systems are supported:

- Cartesian: (x, y, z, t)
- cylindrical about the z axis: (r, phi, z, t) in OPAL, (r, phi, y, t) in OPALX

13.2.1 Cartesian Mode

Cartesian mode is selected with COORDINATE_SYSTEM = CARTESIAN.

OPAL

The file format is:

```
<number of rows>
1  x [m]
2  y [m]
3  z [m]
4  t [ns]
5  Bx [kGauss]
6  By [kGauss]
7  Bz [kGauss]
8  Ex [MV/m]
9  Ey [MV/m]
```

```
10 Ez [MV/m]
0
<x0> <y0> <z0> <t0> <Bx0> <By0> <Bz0> <Ex0> <Ey0> <Ez0>
...
```

OPALX

The file format is:

```
<number of rows>
1  x [m]
2  y [m]
3  z [m]
4  t [ns]
5  Bx [T]
6  By [T]
7  Bz [T]
8  Ex [MV/m]
9  Ey [MV/m]
10 Ez [MV/m]
0
<x0> <y0> <z0> <t0> <Bx0> <By0> <Bz0> <Ex0> <Ey0> <Ez0>
...
```

Attributes in Cartesian mode:

Attribute	Meaning
`FILE_NAME`	output file name; existing files are overwritten
`COORDINATE_SYSTEM`	must be `CARTESIAN`
`X_START`, `Y_START`, `Z_START`	grid origin in `m`
`DX`, `DY`, `DZ`	spatial step size in `m`
`X_STEPS`, `Y_STEPS`, `Z_STEPS`	spatial step counts; must be positive integers
`T_START`	start time in `ns`
`DT`	time step size in `ns`
`T_STEPS`	number of time steps; must be a positive integer

13.2.2 Cylindrical Mode

Cylindrical mode is selected with `COORDINATE_SYSTEM = CYLINDRICAL`.

OPAL

The file format is:

```
<number of rows>
1 r [m]
2 phi [deg]
3 z [m]
4 t [ns]
5 Br   [kGauss]
6 Bphi [kGauss]
7 Bz   [kGauss]
8 Er   [MV/m]
9 Ephi [MV/m]
10 Ez  [MV/m]
0
<r0> <phi0> <z0> <t0> <Br0> <Bphi0> <Bz0> <Er0> <Ephi0> <Ez0>
...
```

OPALX

The file format is:

```
<number of rows>
1 r [m]
2 phi [deg]
3 y [m]
4 t [ns]
5 Br   [T]
6 Bphi [T]
7 By   [T]
8 Er   [MV/m]
9 Ephi [MV/m]
10 Ey  [MV/m]
0
<r0> <phi0> <y0> <t0> <Br0> <Bphi0> <By0> <Er0> <Ephi0> <Ey0>
...
```

Attributes in cylindrical mode:

Attribute	Meaning
`FILE_NAME`	output file name; existing files are overwritten
`COORDINATE_SYSTEM`	must be `CYLINDRICAL`
`R_START`	radial start point in `m`
`DR`	radial step size in `m`
`R_STEPS`	number of radial steps; must be a positive integer
`PHI_START`	start point in `phi`

Attribute	Meaning
DPHI	azimuthal step size
PHI_STEPS	number of azimuthal steps; must be a positive integer
T_START	start time in ns
DT	time step size in ns
T_STEPS	number of time steps; must be a positive integer

OPAL

Additional OPAL cylindrical attributes:

Attribute	Meaning
Z_START	start point in z [m]
DZ	step size in z [m]
Z_STEPS	number of z steps; must be a positive integer

OPALX

Additional OPALX cylindrical attributes:

Attribute	Meaning
Y_START	start point in y [m]
DY	step size in y [m]
Y_STEPS	number of y steps; must be a positive integer
CYL_ORIGIN_X, CYL_ORIGIN_Y, CYL_ORIGIN_Z	optional cylindrical-origin coordinates in m; defaults are 0

14 Beam Lines

The accelerator to be studied is represented as a sequence of physical elements called a beam line. A beam line is built from simpler beam lines whose definitions can be nested to any level. A `LINE` command provides the formal definition:

```
label: LINE = (member, ..., member);
```

Each `member` may be:

- an element label,
- a beamline label,
- a sub-line enclosed in parentheses.

14.1 Simple Beam Lines

The simplest beam line is a list of elements:

```
label: LINE = (member, ..., member);
```

Example:

```
L: LINE = (A, B, C, D, A, D);
```

Line attributes:

`ORIGIN` position vector of the origin of the line. Elements placed using `ELEMEDGE` use this as reference.

`ORIENTATION` orientation vector of Tait-Bryan angles at the line origin.

14.2 Sub-lines

Instead of referring directly to an element, a beamline member can refer to another named beam line. This provides shorthand notation for repeated lattice segments. Lines and sub-lines can be entered in any order, but when a line is used, all of its sub-lines must already be known to the parser.

Example:

```
L: LINE = (A, B, S, B, A, S, A, B);
S: LINE = (C, D, E);
```

The source manual expands this in two steps.

1. Replace `S` by its definition:

```
(A, B, (C, D, E), B, A, (C, D, E), A, B)
```

2. Omit parentheses:

```
A, B, C, D, E, B, A, C, D, E, A, B
```

15 Beam Definitions

OPALX

BEAM defines the physical species and reference state for one particle container. A typical generated distribution uses:

```
Beam1: BEAM, PARTICLE=ELECTRON, PC=0.1,
       NALLOC=100000, BCHARGE=1e-9, SOURCES=Sources1;
```

BCHARGE is normally given as a positive charge magnitude. OPALX obtains the macroparticle sign from CHARGE, which defaults to the selected particle's charge. The named emission-source list must exist when RUN resolves the beam.

15.1 Parameters

Parameter	Default	Unit	Current behavior
PARTICLE	none	-	Selects a predefined species. If omitted, both MASS and CHARGE are required.
MASS	species value	GeV/c^2	Overrides the predefined rest mass, or defines a custom species with CHARGE.
CHARGE	species value	e	Charge in proton-charge units. It also determines the macroparticle charge sign.
ENERGY	unset	GeV	Total charged-particle energy; must exceed MASS.

Parameter	Default	Unit	Current behavior
PC	unset	GeV/c	Reference momentum; must be positive.
GAMMA	unset	-	Reference Lorentz factor; must be greater than one.
BCHARGE	0	C	Bunch-charge magnitude used with NALLOC to calculate macroparticle charge.
NALLOC	required	particles	Positive allocation and macroparticle-weight baseline. Real input is converted to an integer size.
SOURCES	required	name	EMISSIONSOURCELIST used by this non-photon beam.
GLOBALPROCESSES	empty	names	Uppercase array of global processes. Only DECAY is currently accepted.
DAUGHTERBEAM	empty	name	Beam/container that receives daughter particles from a configured decay.
POLARIZATION	unset	-	Muon-only vector {Px, Py, Pz} with $\|\mathbf{P}\| \leq 1$. Setting it, including to zero, enables spin storage.
BCURRENT, BFREQ	unset	-	Legacy attributes explicitly rejected by OPALX; use BCHARGE.

The predefined PARTICLE values are PHOTON, ELECTRON, POSITRON, MUON, PION, PROTON,

ANTIPROTON, DEUTERON, HMINUS, H2P, ALPHA, CARBON, XENON, and URANIUM.

15.2 Reference energy

For generated distributions, set one of GAMMA, ENERGY, or PC. If more than one is supplied, the implementation currently uses GAMMA before ENERGY, and ENERGY before PC; relying on that precedence is discouraged.

FROMFILE and EMITTEDFROMFILE carry absolute particle momenta. A beam using either type must omit all three reference-energy attributes. See Beam and Distributions for their file conventions.

15.3 Allocation and sources

NALLOC is required and must be positive. It remains the denominator used to derive macroparticle charge even when the sum of NPARTDIST over all sources sets a different runtime allocation. Keep NALLOC at least as large as that sum unless the weighting is intentional.

Every tracked non-photon beam requires a non-empty SOURCES name. The target list must contain at least one EMISSIONSOURCE; it is resolved during RUN.

15.4 Multiple species, decay, and spin

Use TRACK.BEAMS to select several beam definitions in a fixed order. OPALX creates one particle container per entry. For decay calculations:

- GLOBALPROCESSES={DECAY} is implemented for muons and pions.
- DAUGHTERBEAM must name another beam selected by the same TRACK command.
- Parent and daughter beams must represent one physical particle per macroparticle: BCHARGE = NALLOC * |CHARGE| * q_e.
- Muon decay requires POLARIZATION; a daughter-muon beam also needs it so the destination container allocates spin storage.

The Decay Processes chapter gives the physics model and validation.

15.5 Photon definitions

PARTICLE=PHOTON is accepted as a definition but is rejected by TRACK. Photon beams require positive ENERGY and NALLOC, and reject MASS, CHARGE, PC, GAMMA, and SOURCES. The current source declares the shared BEAM.ENERGY input unit as GeV; no photon transport path consumes the value yet.

OPAL

The BEAM command defines the particle species, reference energy or momentum, and global beam properties used by tracking and diagnostics.

```
label: BEAM, PARTICLE=name, MASS=real, CHARGE=real,
       ENERGY=real, PC=real, GAMMA=real, BCURRENT=real,
       NPART=real, BFREQ=real;
```

The label is optional and defaults to UNNAMED_BEAM.

15.6 Particle Definition

PARTICLE selects one of the predefined species such as ELECTRON, POSITRON, MUON, PROTON, DEUTERON, HMINUS, H2P, ALPHA, CARBON, XENON, or URANIUM.

For custom species, the user can provide:

- MASS in GeV
- CHARGE in elementary-charge units

15.7 Beam Energy

There are three standard energy attributes, with the usual priority:

- GAMMA
- ENERGY
- PC

15.8 Other Attributes

The other important beam attributes are:

BFREQ bunch frequency in MHz.
BCURRENT bunch current in A.
NPART number of macro particles used in the simulation.

`NSLICE` number of slices per bunch.

`MOMENTUMTOLERANCE` tolerance used when checking consistency between the distribution and the reference momentum.

16 Beam and Distributions

OPALX

OPALX separates the sampled phase space from its placement in the simulation:

```
Dist: DISTRIBUTION, TYPE=GAUSS, NPARTDIST=10000,
      SIGMAX=1e-3, SIGMAY=1e-3, SIGMAZ=2e-3,
      SIGMAPX=1e-4, SIGMAPY=1e-4, SIGMAPZ=1e-4;
Source: EMISSIONSOURCE, DISTRIBUTION=Dist;
Sources: EMISSIONSOURCELIST=(Source);
Beam: BEAM, PARTICLE=ELECTRON, PC=0.1, NALLOC=10000,
      BCHARGE=-1e-9, SOURCES=Sources;
```

The `DISTRIBUTION` defines the particle sample, `EMISSIONSOURCE` adds source offsets and timing, `EMISSIONSOURCELIST` groups sources, and `BEAM.SOURCES` selects the group. See Beam Definitions for the complete `BEAM` attribute table and its reference-energy rules.

16.1 EMISSIONSOURCE

```
name: EMISSIONSOURCE, DISTRIBUTION=distribution_name, ...;
```

Offsets are applied after sampling. Position uses metres, momentum uses dimensionless normalized momentum $p/mc = \beta\gamma$, and time uses seconds.

Parameter	Default	Meaning and constraints
`DISTRIBUTION`	required	Previously defined `DISTRIBUTION` name.
`ROX`, `ROY`, `ROZ`	`0 m`	Position offset added to every sampled particle.
`POX`, `POY`, `POZ`	`0`	Normalized-momentum offset added after sampling.
`TO`	`0 s`	Source start time. For one-shot types, `TO > 0` delays injection.
`EMISSIONMODEL`	`NONE`	`NONE` or `ASTRA`; availability depends on distribution type.
`EKIN`	`0 eV`	Thermal kinetic energy for generated emitted particles; OPALX uses its absolute value.

Parameter	Default	Meaning and constraints
ZEROFACE_ROZ	FALSE	Apply explicit image-charge Dirichlet correction at $z = \mathrm{R0Z}$.
SHIFTED_GREENS_FUNCTION	FALSE	Alternative shifted-Green correction; requires an OPEN field solver.
ZEROFACEPLANEDUMP	0	Non-negative integer dump frequency; requires ZEROFACE_ROZ=TRUE.
ZEROFACE_MAXSTEPS	0	Non-negative correction-step limit; 0 means unlimited.

ZEROFACE_ROZ and SHIFTED_GREENS_FUNCTION are mutually exclusive. A run can activate at most one source plane by either method.

For generated emission, NONE adds the momentum corresponding to EKIN in $+z$. ASTRA samples the forward half-sphere with magnitude

$$p_{\mathrm{th}} = \sqrt{\left(1 + \frac{E_{\mathrm{kin}}}{mc^2}\right)^2 - 1}, \qquad p_z \geq 0,$$

before adding P0X, P0Y, and P0Z.

16.2 EMISSIONSOURCELIST

```
Sources: EMISSIONSOURCELIST=(Source1, Source2);
```

The list is ordered and must contain at least one previously defined EMISSIONSOURCE. Attach it with SOURCES=Sources on BEAM. Several sources may use the same distribution, each with independent offsets and start time.

16.3 DISTRIBUTION

```
name: DISTRIBUTION, TYPE=distribution_type, ...;
```

OPALX accepts exactly GAUSS, MULTIVARIATEGAUSS, FLATTOP, OPALFLATTOP, FROMFILE, and EMITTEDFROMFILE.

Available input parameters

Parameter	Default / unit	Used by	Runtime effect
`TYPE`	required	all	Selects one of the six types above.
`NPARTDIST`	`0`, particles	all	Requested macroparticle count. Use a positive integer except where `EMITTEDFROMFILE` explicitly selects all file records.
`FNAME`	empty, path	file types	Input particle file. Relative paths are resolved from the OPALX input file.
`SIGMAX`, `SIGMAY`, `SIGMAZ`	`0 m`	Gaussian types; transverse flat-top types	Spatial widths; flat-top `SIGMAX` and `SIGMAY` are ellipse semi-axes.
`SIGMAPX`, `SIGMAPY`, `SIGMAPZ`	`0`, $\beta\gamma$	Gaussian types	RMS normalized-momentum widths.
`CORR`	empty array	`MULTIVARIATEGAUSS`	Fifteen off-diagonal correlation coefficients.
`SIGMAT`	`0 s`	flat-top types; optional emitted-file window	Default RMS width for both pulse edges.
`TPULSEFWHM`	`0 s`	flat-top types; optional emitted-file window	Full pulse width at half maximum.
`TRISE`, `TFALL`	`0 s`	flat-top types; optional emitted-file window	Edge times; a positive value overrides the corresponding `SIGMAT`.
`CUTOFFLONG`	`3`, σ	flat-top types; optional emitted-file window	Truncates both Gaussian pulse edges.
`FTOSCAMPLITUDE`	`0 %`	`OPALFLATTOP`	Sinusoidal modulation amplitude, clamped to 100%.

Parameter	Default / unit	Used by	Runtime effect
FTOSCPERIODS	0	OPALFLATTOP	Number of modulation periods across the flat interval.
EMITTED	FALSE	flat-top types; optional emitted-file window	Enables generated time-dependent emission.
EMISSIONSTEPS	100	OPALFLATTOP, EMITTEDFROMFILE	Preferred emission resolution; positive values are rounded up.
CUTOFFX, CUTOFFY	3, σ	none currently	**Accepted but currently ignored by active samplers.**
CUTOFFPX, CUTOFFPY, CUTOFFPZ	3, σ	none currently	**Accepted but currently ignored; Gaussian bounds are fixed internally.**
CORRX, CORRY, CORRZ, CORRT	0	none currently	**Accepted but currently ignored; use CORR with MULTIVARIATEGAUSS.**

Non-file distributions require PC, ENERGY, or GAMMA on BEAM. FROMFILE and EMITTEDFROMFILE contain absolute normalized momenta and reject those three beam-energy attributes.

16.3.1 GAUSS

GAUSS samples independent coordinates. Positions are truncated at the internally fixed $\pm 3\sigma$ bounds and translated so their sampled mean is zero; momenta are unbounded normals around the beam reference momentum.

$$R_i \sim \mathcal{N}(0, \sigma_{R_i}^2), \qquad P_i \sim \mathcal{N}(\bar{P}_i, \sigma_{P_i}^2).$$

Use	Parameters
Count	NPARTDIST (positive integer)
Position	SIGMAX, SIGMAY, SIGMAZ
Momentum	SIGMAPX, SIGMAPY, SIGMAPZ
Source	R0*, P0*, T0; EMISSIONMODEL=NONE only

```
G: DISTRIBUTION, TYPE=GAUSS, NPARTDIST=10000,
   SIGMAX=1e-3, SIGMAY=1e-3, SIGMAZ=2e-3,
   SIGMAPX=1e-4, SIGMAPY=1e-4, SIGMAPZ=2e-4;
```

With `T0 > 0`, the complete sample is injected once when a tracker step crosses `T0`. Named correlations and user-supplied cutoffs do not affect this sampler.

16.3.2 `MULTIVARIATEGAUSS`

Let

$$q = (x, p_x, y, p_y, z, p_z)^T, \qquad D = \mathrm{diag}(\sigma_x, \sigma_{p_x}, \sigma_y, \sigma_{p_y}, \sigma_z, \sigma_{p_z}).$$

`CORR` defines a correlation matrix C and the sampler forms $\Sigma = DCD = LL^T$, then applies the Cholesky factor L to independent normal samples. C must be symmetric positive definite.

Use	Parameters
Count and widths	`NPARTDIST`, all six `SIGMA*` values
Coupling	`CORR` with exactly 15 values
Source	`R0*`, `P0*`, `T0`; `EMISSIONMODEL=NONE` only

The array is the upper triangle in this order:

```
(x,px), (x,y), (x,py), (x,z), (x,pz),
(px,y), (px,py), (px,z), (px,pz),
(y,py), (y,z), (y,pz), (py,z), (py,pz), (z,pz)
```

```
MG: DISTRIBUTION, TYPE=MULTIVARIATEGAUSS, NPARTDIST=10000,
    SIGMAX=1e-3, SIGMAPX=1e-4,
    SIGMAY=1e-3, SIGMAPY=1e-4,
    SIGMAZ=2e-3, SIGMAPZ=2e-4,
    CORR={0.2,0,0,0,0, 0,0,0,0, 0,0,0, 0,0,0};
```

Sampling uses internally fixed three-sigma bounds. `T0 > 0` produces one delayed injection, as for `GAUSS`.

16.3.3 FLATTOP

FLATTOP is a time-dependent emitter. Transverse positions are uniform on an ellipse with semi-axes a = SIGMAX and b = SIGMAY:

$$x = a\sqrt{u}\cos(2\pi v), \qquad y = b\sqrt{u}\sin(2\pi v), \qquad u, v \sim U(0,1).$$

Use	Parameters
Required	NPARTDIST>0, EMITTED=TRUE, SIGMAX, SIGMAY
Pulse	SIGMAT or TRISE/TFALL, TPULSEFWHM, CUTOFFLONG
Source	T0, R0*, P0*, EKIN; EMISSIONMODEL=NONE or ASTRA

For a positive TRISE or TFALL, OPALX converts the corresponding input edge time to an RMS width using

$$\sigma = \frac{T_{\text{edge}}}{\sqrt{2\ln 10} - \sqrt{2\ln(10/9)}}.$$

With edge widths σ_r, σ_f and cutoff C, the flat interval and total emission duration are

$$T_{\text{flat}} = \max\!\left(0, T_{\text{FWHM}} - \sqrt{2\ln 2}(\sigma_r + \sigma_f)\right), \qquad T_{\text{emit}} = C(\sigma_r + \sigma_f) + T_{\text{flat}}.$$

The rate consists of truncated Gaussian rise and fall edges around a constant flat interval. Each tracker step integrates that profile and gives newborn particles a random fractional timestep.

```
FT: DISTRIBUTION, TYPE=FLATTOP, NPARTDIST=10000, EMITTED=TRUE,
    SIGMAX=1e-3, SIGMAY=1e-3,
    TRISE=0.5e-12, TFALL=0.5e-12,
    TPULSEFWHM=10e-12, CUTOFFLONG=3;
```

16.3.4 OPALFLATTOP

OPALFLATTOP uses the same transverse and pulse mathematics as FLATTOP, but precomputes a globally sorted inventory of particle birth times. This gives an exact requested inventory and OPAL-compatible pulse centering.

Use	Parameters
Required	All FLATTOP parameters; NPARTDIST>0 and EMITTED=TRUE are enforced
Resolution	EMISSIONSTEPS
Modulation	FTOSCAMPLITUDE, FTOSCPERIODS
Source	T0, R0*, P0*, EKIN; EMISSIONMODEL=NONE or ASTRA

On the flat interval, birth-time density is proportional to

$$1 + a \sin\left(2\pi n \frac{t}{T_{\text{flat}}}\right), \qquad a = \min\left(1, \frac{|\text{FTOSCAMPLITUDE}|}{100}\right),$$

where $n = |\text{FTOSCPERIODS}|$.

```
OFT: DISTRIBUTION, TYPE=OPALFLATTOP, NPARTDIST=10000, EMITTED=TRUE,
     SIGMAX=1e-3, SIGMAY=1e-3,
     TRISE=0.5e-12, TFALL=0.5e-12,
     TPULSEFWHM=10e-12, CUTOFFLONG=3,
     EMISSIONSTEPS=100, FTOSCAMPLITUDE=5, FTOSCPERIODS=3;
```

16.3.5 FROMFILE

FROMFILE reads an injected 6D sample. Set a positive NPARTDIST; OPALX uses at most that many records from the beginning of the file.

Use	Parameters
Required	FNAME, positive NPARTDIST
Source	R0*, P0*, T0; EMISSIONMODEL=NONE only
Beam	Omit PC, ENERGY, and GAMMA

The first non-comment line is the declared count, the second is a header, and exactly that many data rows must follow. Header order is arbitrary but must contain x, y, z, px, py, and pz (case-insensitive). Blank lines and lines beginning with # are ignored.

```
3
x px y py z pz
0.0 0.001 0.0 0.000 0.0 1.2
0.1 0.002 0.0 0.001 0.0 1.2
-0.1 0.000 0.0 -0.001 0.0 1.2
```

```
FF: DISTRIBUTION, TYPE=FROMFILE, FNAME="particles.txt", NPARTDIST=3;
```

Positions are metres and file momenta are absolute $\beta\gamma$ values. Source offsets are added after reading. `T0 > 0` delays the complete file sample to one tracker step.

16.3.6 EMITTEDFROMFILE

EMITTEDFROMFILE reads old-OPAL emitted records and releases them according to their file times.

Use	Parameters
Required	FNAME; NPARTDIST=0 selects all records, otherwise it selects a prefix
Resolution	EMISSIONSTEPS
Optional window	EMITTED=TRUE plus flat-top pulse parameters
Source	R0*, P0*, T0; EMISSIONMODEL=NONE only
Beam	Omit PC, ENERGY, and GAMMA

Rows are positional: `x px y py t pz [bin]`. The positive-integer `bin` column is optional. A file may use a leading count and header, or a comment header.

```
3
x px y py t pz bin
0.0 0.001 0.0 0.000 -2e-12 1.2 1
0.1 0.002 0.0 0.001 -1e-12 1.2 2
-0.1 0.000 0.0 -0.001 0.0 1.2 3
```

```
EFF: DISTRIBUTION, TYPE=EMITTEDFROMFILE,
     FNAME="emitted.txt", NPARTDIST=0, EMISSIONSTEPS=100;
```

Without bins or a configured window, OPALX negates the old-OPAL file times, centres their span, and uses

$$t_{\text{birth}} = \text{T0} - t_{\text{file}} - t_{\text{centre}}.$$

An enabled flat-top window may replace the inferred span, but it must cover the latest selected file time. Positions and momenta receive the source offsets; the file momenta remain absolute.

16.4 Reproducibility and current limitations

Record the OPALX revision, random seed, `NPARTDIST`, `BEAM.NALLOC`, bunch charge, source list, and every referenced file. Keep `NALLOC` at least as large as the sum of the source distributions' particle counts.

Current source limitations are deliberate documentation constraints:

- only `MULTIVARIATEGAUSS.CORR` affects correlations;
- Gaussian cutoff attributes are parsed but sampling uses internal bounds;
- `FTOSCAMPLITUDE` and `FTOSCPERIODS` affect only `OPALFLATTOP`;
- `ASTRA` is supported only by the generated flat-top emitters; and
- file distributions take momentum from their files, not from `BEAM` energy.

OPAL

The `DISTRIBUTION` command defines how particles are introduced into a simulation. A distribution has a name, a type, and a set of attributes that control its geometry, momentum spread, emission behavior, and optional correlations.

```
Name: DISTRIBUTION, TYPE = DISTRIBUTION_TYPE,
      ATTRIBUTE1 = ...,
      ATTRIBUTE2 = ...;
```

The supported distribution types depend on whether you are looking at the legacy OPAL manual path or the current OPALX implementation.

Type	Description
`FROMFILE`	Read initial particle coordinates from a user-provided text file.
`GAUSS`	Gaussian distribution in one or more dimensions.
`FLATTOP`	Hard-edge transverse distribution with flat-top time structure.
`BINOMIAL`	Binomial family controlled by one shape parameter per axis.
`GAUSSMATCHED`	Matched Gaussian distribution for cyclotron-style matching.
`MULTIGAUSS`	Train of Gaussian pulses along the longitudinal direction.
`GUNGAUSSFLATTOPTH`	Legacy shorthand for emitted `FLATTOP` with `ASTRA`.
`ASTRAFLATTOPTH`	Legacy emitted flat-top photoinjector distribution.

16.5 Units

Lengths are given in meters and times in seconds. Momentum input units depend on `INPUTMOUNITS`.

Attribute	Value	Meaning
`INPUTMOUNITS`	`NONE`	Use normalized momentum components `beta_x gamma`, `beta_y gamma`, `beta_z gamma`. This is the OPAL-T default.
`INPUTMOUNITS`	`EVOVERC`	Use momenta in `eV/c`. This is the OPAL-cycl default.

16.5.1 Momentum unit conversion

To convert from normalized momentum to transverse angle in mrad, use

$$(\beta\gamma)_{\text{ref}} = \frac{P}{m_0 c} = \frac{Pc}{m_0 c^2},$$

and

$$P_x[\text{mrad}] = 1000 \times \frac{\beta_x \gamma}{(\beta\gamma)_{\text{ref}}}.$$

To convert from `eV/c` to dimensionless normalized momentum,

$$\beta_x \gamma = \frac{P_x[\text{eV}/c]}{m_0 c} = \frac{P_x[\text{eV}/c]\, c}{m_0 c^2}.$$

The same relations apply to the `y` and `z` components.

16.6 General Distribution Attributes

The first major distinction is whether the distribution is injected at the start of the simulation or emitted over time.

Attribute	Value	Meaning
`EMITTED`	`FALSE`	Inject the full distribution at the start of the simulation. This is the default.

Attribute	Value	Meaning
EMITTED	TRUE	Emit the particles over time. This is currently an OPAL-T mode.

For injected distributions, the longitudinal coordinate is `z` in meters. For emitted distributions, the longitudinal coordinate is `t` in seconds.

16.6.1 Universal Attributes

These attributes apply to all distribution types:

Attribute	Default	Meaning
WRITETOFILE	FALSE	Write the generated initial distribution to a text file.
SCALABLE	FALSE	Make generation scalable with the number of MPI ranks.
WEIGHT	1.0	Relative weight when used in a distribution list.
NBIN	0	Number of energy bins.
SBIN	100	Sample bins per energy bin.
XMULT, YMULT	1.0	Scale transverse positions after generation.
PXMULT, PYMULT, PZMULT	1.0	Scale momentum components after generation.
OFFSETX, OFFSETY	0.0	Shift average transverse position.
OFFSETPX, OFFSETPY, OFFSETPZ	0.0	Shift average momentum.
ID1, ID2	zero 6-vector	Tracer particles written to `track_orbit.dat` in OPAL-cycl.

16.6.2 Injected Distribution Attributes

Attribute	Default	Meaning
ZMULT	1.0	Scale longitudinal position after generation.
OFFSETZ	0.0	Shift average longitudinal position.

16.6.3 Emitted Distribution Attributes

Attribute	Default	Meaning
TMULT	1.0	Scale emission time after generation.
OFFSETT	0.0	Delay emission relative to the reference particle.
EMISSIONSTEPS	1	Number of timesteps used during emission.
EMISSIONMODEL	NONE	Emission model applied at the cathode.

16.7 Distribution Types

16.7.1 FROMFILE

FROMFILE reads coordinates from an external text file.

```
Name: DISTRIBUTION, TYPE=FROMFILE,
      FNAME="text file name";
```

The type-specific attribute is:

Attribute	Meaning
FNAME	File name containing the particle coordinates.

For an injected FROMFILE distribution, the file format is:

```
N
x1 px1 y1 py1 z1 pz1
x2 px2 y2 py2 z2 pz2
...
xN pxN yN pyN zN pzN
```

For an emitted FROMFILE distribution, z is replaced by t:

```
N
x1 px1 y1 py1 t1 pz1
x2 px2 y2 py2 t2 pz2
...
xN pxN yN pyN tN pzN
```

The emitted case is internally shifted so that emission starts from negative time and particles appear as the simulation clock advances.

When using `FROMFILE`, the particle count must match the `BEAM NPART` expectation, and the mean momentum in the file must be consistent with the beam energy settings.

16.7.2 GAUSS

`GAUSS` creates a six-dimensional Gaussian bunch. The core attributes are:

Attribute	Meaning
`SIGMAX`, `SIGMAY`	RMS transverse widths.
`SIGMAR`	RMS radial width; overrides `SIGMAX` and `SIGMAY` if nonzero.
`SIGMAZ`, `SIGMAT`	RMS bunch length in `z` or `t`. `SIGMAZ` overrides `SIGMAT`.
`SIGMAPX`, `SIGMAPY`, `SIGMAPZ`	RMS momentum spreads.
`CUTOFFX`, `CUTOFFY`, `CUTOFFR`, `CUTOFFLONG`, `CUTOFFPX`, `CUTOFFPY`, `CUTOFFPZ`	Cutoffs expressed in units of the corresponding sigma.

Example:

```
Name: DISTRIBUTION, TYPE       = GAUSS,
      SIGMAX     = 0.001,
      SIGMAY     = 0.003,
      SIGMAZ     = 0.002,
      SIGMAPX    = 0.0,
      SIGMAPY    = 0.0,
      SIGMAPZ    = 0.0,
      CUTOFFX    = 2.0,
      CUTOFFY    = 2.0,
      CUTOFFLONG = 4.0,
      OFFSETX    = 0.001,
      OFFSETY    = -0.002,
      OFFSETZ    = 0.01,
      OFFSETPZ   = 1200.0;
```

16.7.2.1 `GAUSS` for photoinjectors

For emitted beams, `GAUSS` can also produce a half-Gaussian rise, flat-top, half-Gaussian fall time profile. The key extra attributes are:

Attribute	Meaning
`TPULSEFWHM`	Full-width-at-half-maximum pulse length.
`TRISE`	Rise time. Overrides `SIGMAT`.
`TFALL`	Fall time. Overrides `SIGMAT`.
`FTOSCAMPLITUDE`	Oscillation amplitude on the flat top, in percent.
`FTOSCPERIODS`	Number of oscillation periods across the flat top.

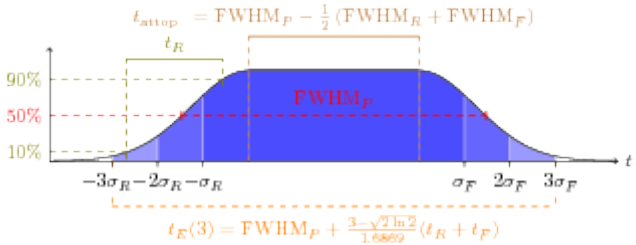


Figure 16.1: Emitted `GAUSS` and `FLATTOP` time profile with half-Gaussian edges and optional flat-top oscillations.

The rise and fall parameters correspond to

$$\text{TRISE} = 1.6869\,\sigma_R, \qquad \text{TFALL} = 1.6869\,\sigma_F,$$

and the pulse FWHM is

$$\text{TPULSEFWHM} = t_{\text{flattop}} + \sqrt{2\ln 2}(\sigma_R + \sigma_F).$$

The total emission time depends on `CUTOFFLONG`:

$$t_E = \text{TPULSEFWHM} + \frac{\text{CUTOFFLONG} - \sqrt{2\ln 2}}{1.6869}(\text{TRISE} + \text{TFALL}).$$

16.7.2.2 Correlations for `GAUSS`

The Gaussian generator also supports experimental correlations. They can be given either as a compact array `R` or through named coefficients such as:

- `CORRX`, `CORRY`, `CORRZ`
- `R51`, `R52`

- R61, R62

In the four-dimensional (x, p_x, z, p_z) subspace, the correlation matrix is

$$\sigma = \begin{bmatrix} 1 & c_x & R_{51} & R_{61} \\ c_x & 1 & R_{52} & R_{62} \\ R_{51} & R_{52} & 1 & c_t \\ R_{61} & R_{62} & c_t & 1 \end{bmatrix}.$$

The implementation constructs correlated samples from the Cholesky factorization of this matrix. This feature is experimental and only documented for Gaussian distributions.

16.7.3 FLATTOP

FLATTOP defines hard-edge distributions and is commonly used to model laser profiles in photoinjectors.

16.7.3.1 Injected FLATTOP

For injected beams, the distribution is a uniformly filled ellipse transversely and uniform in z.

Attribute	Meaning
SIGMAX, SIGMAY	Hard-edge widths.
SIGMAR	Radial hard-edge width; overrides SIGMAX and SIGMAY.
SIGMAZ	Hard-edge bunch length.

16.7.3.2 Emitted FLATTOP

For emitted beams, FLATTOP uses the same longitudinal pulse-shape parameters as the photoinjector-style GAUSS case.

Additional attributes include:

Attribute	Meaning
SIGMAX, SIGMAY, SIGMAR	Hard-edge transverse beam size.
SIGMAT, TPULSEFWHM, TRISE, TFALL	Time-profile parameters.
FTOSCAMPLITUDE, FTOSCPERIODS	Oscillations on the flat top.
LASERPROFFN, IMAGENAME, INTENSITYCUT	Laser-profile image input.
FLIPX, FLIPY, ROTATE90, ROTATE180, ROTATE270	Laser-image transforms.

Example:

```
Dist: DISTRIBUTION, TYPE = FLATTOP,
      SIGMAX = 0.001,
      SIGMAY = 0.002,
      TRISE = 0.5e-12,
      TFALL = 0.5e-12,
      TPULSEFWHM = 10.0e-12,
      CUTOFFLONG = 4.0,
      NBIN = 5,
      EMISSIONSTEPS = 100,
      EMISSIONMODEL = ASTRA,
      EKIN = 0.5,
      EMITTED = TRUE;
```

The legacy manual also describes a laser-image driven transverse sampling path through `LASERPROFFN`, but explicitly marks it as under development.

16.7.3.3 `GUNGAUSSFLATTOPTH` and `ASTRAFLATTOPTH`

These are legacy shorthands for emitted flat-top photoinjector distributions. Both correspond to `FLATTOP`-style emission. `GUNGAUSSFLATTOPTH` automatically enables `EMITTED=TRUE` and `EMISSIONMODEL=ASTRA`, while `ASTRAFLATTOPTH` follows the same idea with a slightly different legacy longitudinal profile generator.

16.7.4 `BINOMIAL`

`BINOMIAL` generates a family of distributions governed by one parameter `m` per axis. Changing `m` moves continuously from hollow-shell and flat-profile shapes toward Gaussian-like limits [1].

The key shape parameters are:

Attribute	Meaning
`MX`	Binomial parameter in `x`.
`MY`	Binomial parameter in `y`.
`MT`, `MZ`	Binomial parameter in the longitudinal direction. `MZ` is the same as `MT`.

The phase-space widths are still set through the usual `SIGMAX`, `SIGMAPX`, `CORRX`, and corresponding `y` and `z/t` variants.

For one plane,

$$\epsilon_x = \sigma_x \sigma_{x'} \cos(\arcsin(\sigma_{12}))\,,$$

TABLE 2: Properties of binomial Phase Space Distributions

m	phase space density $\rho(u,v)$ $u^2+v^2\equiv a^2\le 1$	profile $f(u)=\int_{-\infty}^{+\infty}\rho(u,v)dv$ $u=\frac{x}{x_L}$, $\lvert u\rvert\le 1$,	profile- form	$\frac{x_L}{2\sigma}$ $(\sigma^2\equiv\overline{x^2})$	total emittance	$\frac{\Gamma}{2\sigma}$ Γ = FWHM	$\frac{\Gamma}{x_L}$	$\frac{f(2\sigma)}{f_{max}}$	p = fraction of beam outside 2σ - ellipse	q = fraction of beam outside ampl. $\pm 2\sigma$
0(KV)	$\frac{1}{\pi}\delta(1-a^2)$	$\frac{1}{\pi}(1-u^2)^{-.5}$		0.707	$2\sigma\sigma'$	$(\sqrt{2})$	(2)	–	–	–
0.5	$\frac{1}{\pi}(1-a^2)^{-.5}$	0.5		0.866	$3\sigma\sigma'$	$\sqrt{3}$	2	–	–	–
1	$\frac{1}{\pi}$ (=waterbag)	$\frac{2}{\pi}(1-u^2)^{.5}$		1	$4\sigma\sigma'$	$\sqrt{3}$	$\sqrt{3}$	0	0	0
1.5	$\frac{3}{2\pi}(1-a^2)^{.5}$	$\frac{3}{4}(1-u^2)$		1.118	$5\sigma\sigma'$	1.582	$\sqrt{2}$	20%	8.9%	1.6%
2	$\frac{2}{\pi}(1-a^2)$	$\frac{8}{3\pi}(1-u^2)^{1.5}$		1.225	$6\sigma\sigma'$	1.491	1.217	19.2%	11.1%	2.5%
2.5	$\frac{5}{2\pi}(1-a^2)^{1.5}$	$\frac{15}{16}(1-u^2)^2$		1.323	$7\sigma\sigma'$	1.431	1.082	18.4%	12.0%	3.0%
.....						.....				
3.5	$\frac{7}{2\pi}(1-a^2)^{2.5}$	$\frac{35}{32}(1-u^2)^3$		1.5	$9\sigma\sigma'$	1.361	0.908	17.1%	12.8%	3.5%
....						....				
7	$\frac{7}{\pi}(1-a^2)^6$	$1.52(1-u^2)^{6.5}$		2	$16\sigma\sigma'$	1.272	0.636	15.4%	13.3%	4.1%
....						....				
m	$\frac{m}{\pi}(1-a^2)^{m-1}$	$\frac{1}{I_{2m}}(1-u^2)^{m-.5}$		$\sqrt{\frac{m+1}{2}}$	$2(m+1)\sigma\sigma'$	$*\sqrt{2(1-c)(m+1)}$	$2\sqrt{1-c}$	$(\frac{m-1}{m+1})^{m-.5}$	$(\frac{m-1}{m+1})^m$	recursive formula
$m\to\infty$	$\frac{1}{2\pi\sigma\sigma'}\exp(-\frac{x^2}{2\sigma^2}-\frac{x^2}{2\sigma'^2})$ (Gaussian)	$\frac{1}{\sqrt{2\pi}}\exp(-\frac{x^2}{2\sigma^2})$		∞	∞	$\sqrt{2\ln 2}=1.177$	0	13.5%	13.5%	4.6%
						$*c=\frac{1}{2^{(\frac{1}{m-.5})}}$				

Figure 16.2: Representative transverse properties of binomial phase-space distributions.

with the corresponding Twiss relations

$$\beta_x = \frac{\sigma_x^2}{\epsilon_x}, \qquad \gamma_x = \frac{\sigma_{x'}^2}{\epsilon_x}, \qquad \alpha_x = -\sigma_{12}\sqrt{\beta_x\gamma_x}.$$

Example:

```
Dist: DISTRIBUTION, TYPE    = BINOMIAL,
      SIGMAX  = 2.15e-03,
      SIGMAPX = 1E-6,
      CORRX   = 0.0,
      MX      = 0.01,
      SIGMAY  = 0.50*23.e-03,
      SIGMAPY = 28.0,
      CORRY   = 0.5,
      MY      = 990.0,
      SIGMAT  = 1.0e-1,
      SIGMAPT = 11.96,
      CORRT   = -0.5,
      MT      = 2.0;
```

16.7.5 GAUSSMATCHED

GAUSSMATCHED constructs a matched Gaussian distribution, intended for cyclotron-style matched injection. The main control parameters are:

Attribute	Meaning
DENERGY	Energy step size for the closed-orbit finder.
EX, EY, ET	Projected normalized emittances.
NSTEPS, NSECTORS	Closed-orbit integration controls.
SECTOR	Match using one sector or the full ring.
ORDERMAPS	Order used in the field expansion.
RGUESS	Initial radius guess.
RESIDUUM	Convergence target.
MAXSTEPSCO, MAXSTEPSSI	Iteration limits for the closed-orbit and matching loops.

The legacy manual explicitly notes one limitation: trim-coil field maps are not included in this matched-distribution construction.

16.7.6 MULTIGAUSS

MULTIGAUSS models a train of Gaussian pulses. Transversely it uses a uniform elliptical profile, while longitudinally it generates NPEAKS equally spaced Gaussian peaks [2].

Key attributes are:

Attribute	Meaning
SIGMAX, SIGMAY, SIGMAR	Transverse size.
SIGMAZ, SIGMAT	RMS length of each Gaussian pulse.
SEPPEAKS	Peak-to-peak separation.
NPEAKS	Number of Gaussian pulses.
CUTOFFLONG	Longitudinal cutoff relative to the first and last pulse.
SIGMAPX, SIGMAPY, SIGMAPZ	Momentum spread for injected beams.
CUTOFFPX, CUTOFFPY, CUTOFFPZ	Momentum cutoffs for injected beams.

When emitted, the momentum is assigned by the selected emission model. When injected, the momentum components are sampled from normal distributions.

Example:

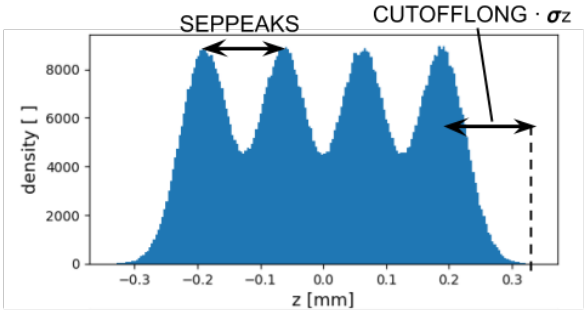


Figure 16.3: Example of an injected `MULTIGAUSS` bunch with several separated longitudinal peaks.

```
Dist: DISTRIBUTION, TYPE = MULTIGAUSS,
      SIGMAPX = 1e-2, SIGMAPY = 1e-2, SIGMAPZ = 1e-2,
      CUTOFFPX = 4.0, CUTOFFPY = 4.0, CUTOFFPZ = 4.0,
      SIGMAR = 340e-6,
      SIGMAZ = 90e-6 / 2.355,
      CUTOFFLONG = 4.0,
      SEPPEAKS = 126e-6,
      NPEAKS = 4,
      EMITTED = FALSE;
```

16.8 Emission Models

Emission models apply only to emitted distributions and determine how thermal energy and cathode physics are translated into initial particle momentum.

16.8.1 `NONE`

`NONE` is the default OPAL-T emission model. It adds a user-specified kinetic energy `EKIN` to the longitudinal momentum only.

Attribute	Default	Meaning
`EKIN`	`1.0 eV`	Thermal energy added during emission.

This model is useful for transversely cold emitted beams. If `EKIN=0`, emitted particles may fail to drift off the cathode cleanly.

16.8.2 ASTRA

ASTRA uses the same EKIN parameter but distributes the momentum three-dimensionally:

$$p_{\text{total}} = \sqrt{\left(\frac{\text{EKIN}}{mc^2} + 1\right)^2 - 1},$$

$$p_x = p_{\text{total}} \sin(\theta)\cos(\phi), \quad p_y = p_{\text{total}} \sin(\theta)\sin(\phi), \quad p_z = p_{\text{total}} |\cos(\theta)|.$$

Here theta is random on [0, pi], and

$$\phi = 2 \arccos(\sqrt{x}),$$

with x a uniform random number on [0, 1].

16.8.3 NONEQUIL

NONEQUIL is a more physical photoemission model for metal cathodes and materials such as CsTe [3], [4], [5].

Its additional parameters are:

Attribute	Default	Meaning
ELASER	4.86 eV	Drive-laser photon energy.
W	4.31 eV	Cathode work function.
FE	7.0 eV	Fermi energy.
CATHTEMP	300 K	Cathode temperature.

Example:

```
Dist: DISTRIBUTION, TYPE = GAUSS,
      SIGMAX = 0.001,
      SIGMAY = 0.002,
      TRISE = 1.0e-12,
      TFALL = 1.0e-12,
      TPULSEFWHM = 15.0e-12,
      CUTOFFLONG = 3.0,
      NBIN = 10,
      EMISSIONSTEPS = 100,
      EMISSIONMODEL = NONEQUIL,
      ELASER = 6.48,
      W = 4.1,
      FE = 7.0,
      CATHTEMP = 325,
      EMITTED = TRUE;
```

16.9 Distribution List

The `RUN` command can accept either a single distribution or a list:

```
RUN, METHOD = "PARALLEL-T",
     BEAM = beam_name,
     FIELDSOLVER = field_solver_name,
     DISTRIBUTION = DIST1;
```

or

```
RUN, METHOD = "PARALLEL-T",
     BEAM = beam_name,
     FIELDSOLVER = field_solver_name,
     DISTRIBUTION = {DIST1, DIST2, DIST3};
```

In a distribution list:

- the first entry is the master distribution
- all other distributions inherit its `EMITTED` or injected mode
- the total number of particles is still controlled by the `BEAM` command
- per-distribution particle counts are apportioned through `WEIGHT`

`FROMFILE` is the special case: its particle count comes from the file rather than from `BEAM` and `WEIGHT`.

17 Structures

Structures are named definitions that configure reusable runtime machinery and are attached to another input object by name. They are declared once, validated by their implementation, and resolved when a run constructs the corresponding runtime component.

17.1 BINNING

BINNING defines a longitudinal histogram used by the binned space-charge solver. Define it separately and attach it to a FIELDSOLVER with BINS:

```
Bins1: BINNING,
    MAXBINS=120,
    DESIREDWIDTH=0.2,
    BINNINGALPHA=1.1,
    BINNINGBETA=1.6,
    PARAMETER=VELOCITYZ,
    ADAPTIVEBINNING=TRUE,
    TABLEPRINTFREQ=10;

FS1: FIELDSOLVER, TYPE=OPEN, BINS=Bins1,
    NX=32, NY=32, NZ=32,
    PARFFTX=TRUE, PARFFTY=TRUE, PARFFTZ=TRUE,
    BCFFTX=OPEN, BCFFTY=OPEN, BCFFTZ=OPEN;
```

Use BINS=NONE, the field-solver default, to disable binning.

Parameters

Parameter	Default	Current behavior
MAXBINS	128	Number of bins in the initial uniform histogram and upper bound after merging. The real input is converted to an integer; use a positive value.

Parameter	Default	Current behavior
DESIREDWIDTH	0.1	Target-width bias used by the adaptive merge cost function.
BINNINGALPHA	1.0	Weight penalizing wide merged bins.
BINNINGBETA	1.5	Weight biasing the cost toward DESIREDWIDTH.
PARAMETER	VELOCITYZ	Bunch quantity used to assign particles to bins.
ADAPTIVEBINNING	TRUE	Merge the initial uniform histogram adaptively. FALSE retains the uniform MAXBINS histogram.
DUMPBINSFILE	NONE	JSON output name. Empty or NONE disables output; .json is appended when missing.
DUMPBINSFREQ	1	JSON dump frequency in global steps. Must be at least one when file output is enabled.
TABLEPRINTFREQ	10	Console bin-statistics frequency in global steps. 0 disables it; negative values are rejected.

The parser accepts VELOCITYZ, POSITIONZ, PZ, and GAMMAZ, but the current runtime implements only VELOCITYZ and GAMMAZ. Selecting either of the other two values fails when the bunch creates its bins.

With normalized momentum $P_z = p_z/(mc)$, the implemented selectors are

$$\text{VELOCITYZ} = \frac{|P_z|}{\sqrt{1+P_z^2}}, \qquad \text{GAMMAZ} = \sqrt{1+P_z^2}.$$

GAMMAZ is therefore a longitudinal proxy based only on P_z, not the full Lorentz factor when transverse momentum is nonzero.

Adaptive merging

OPALX first creates a uniform histogram over the selected quantity. With ADAPTIVEBINNING=TRUE, adjacent bins are merged using a cost function based on particle fraction, combined width,

DESIREDWIDTH, BINNINGALPHA, and BINNINGBETA. With FALSE, the uniform histogram remains unchanged.

The current parser does not enforce numeric ranges for DESIREDWIDTH, BINNINGALPHA, or BINNINGBETA. In particular, the source descriptions of the weights mention [0,1], while the compiled default of BINNINGBETA is 1.5 and maintained regression inputs use values above one. The manual therefore does not claim a range that the implementation contradicts.

Runtime notes

- Binning is enabled only when a field solver resolves a named BINNING definition through BINS.
- The current bin object is constructed from the primary particle container.
- OPTION.MINBINEMITTED controls when space charge begins during emission; OPTION.MINSTEPFORREBIN controls the earliest bin-collapse step.
- DUMPBINSFILE records counts and widths for reproducibility; generated JSON is simulation output and does not belong in the manual repository.

18 Field Solvers

Shared particle-mesh model

Both manuals use `FIELDSOLVER` to connect a particle-mesh space-charge model to tracking. Charge is deposited on a mesh, Poisson's equation

$$\nabla^2\phi = -\frac{\rho}{\epsilon_0}, \qquad \mathbf{E} = -\nabla\phi,$$

is solved with the selected boundary model, and the field is interpolated back to the particles. The version-specific sections below describe the available backends, parameters, and compatibility restrictions.

OPALX

`FIELDSOLVER` selects the mesh Poisson backend and the space-charge mode used by a tracking run. Define the object before `TRACK`, then attach it by name on `RUN`:

```
FS1: FIELDSOLVER, TYPE=OPEN,
    NX=64, NY=64, NZ=128,
    PARFFTX=TRUE, PARFFTY=TRUE, PARFFTZ=TRUE,
    BCFFTX=OPEN, BCFFTY=OPEN, BCFFTZ=OPEN,
    GREENSF=INTEGRATED, BBOXINCR=2.0;

TRACK, LINE=Line1, BEAM=Beam1,
       DT={1e-11}, MAXSTEPS={1000}, ZSTOP={2.0};
  RUN, METHOD=PARALLEL, FIELDSOLVER=FS1;
ENDTRACK;
```

For an active solver, `BEAM.BCHARGE` must be nonzero once more than one macroparticle is present. Use `TYPE=NONE` when self-fields should be disabled.

18.1 Solver types

OPALX deposits particle charge on one distributed mesh, solves Poisson's equation, and interpolates the resulting field back to the particles:

$$\nabla^2\phi = -\frac{\rho}{\epsilon_0}, \qquad \mathbf{E} = -\nabla\phi.$$

The configured backend controls how the mesh boundary is interpreted. Binning and cathode-plane corrections are additional modes described below.

TYPE	Boundary model	Current status	Typical use
NONE	No solve	Available no-op	External fields only
FFT	Periodic in all three axes	Available	A periodically repeated charge distribution
OPEN	Isolated free space	Available	Finite bunches; recommended default
CG	Intended potential solve with field boundary conditions	**Unavailable**	Parser-visible, but construction deliberately fails

18.1.1 NONE

NONE skips binning, charge deposition, the field solve, and field gathering. Particle self-fields remain zero. A syntactically complete FIELDSOLVER definition is still required by the current construction path, including positive mesh sizes and all three PARFFT* flags set to TRUE.

18.1.2 FFT

FFT uses IPPL's periodic FFT Poisson solver and returns the electric-field gradient directly. Set all three BCFFT* values to PERIODIC. Before the solve, OPALX subtracts the mean charge density so the periodic domain has zero net charge.

```
FSPeriodic: FIELDSOLVER, TYPE=FFT,
    NX=64, NY=64, NZ=64,
    PARFFTX=TRUE, PARFFTY=TRUE, PARFFTZ=TRUE,
    BCFFTX=PERIODIC, BCFFTY=PERIODIC, BCFFTZ=PERIODIC;
```

This solver represents an infinite periodic repetition of the simulation box; it is not an approximation to an isolated bunch.

18.1.3 OPEN

OPEN uses IPPL's Hockney free-space FFT solver. The doubled-grid convolution removes the periodic wraparound of an ordinary FFT solve. Set all three BCFFT* values to OPEN.

GREENSF selects the free-space kernel:

- `INTEGRATED` is the default integrated-cell Green function.
- `STANDARD` uses the point-sampled Green function.

Both choices are forwarded only to the `OPEN` backend. Establish mesh convergence for the selected kernel rather than assuming identical finite-grid results.

18.1.4 `CG`

> ⚠ Not usable on current master
>
> `TYPE=CG` is accepted by the input parser, but solver construction throws `Cannot use CGSolver yet, not fully implemented.` The generic all-face Dirichlet boundary setting therefore has no runnable backend at present.

`P3M` is not an accepted `FIELDSOLVER.TYPE` on current master. Historical or development-branch references to it do not describe a released OPALX input.

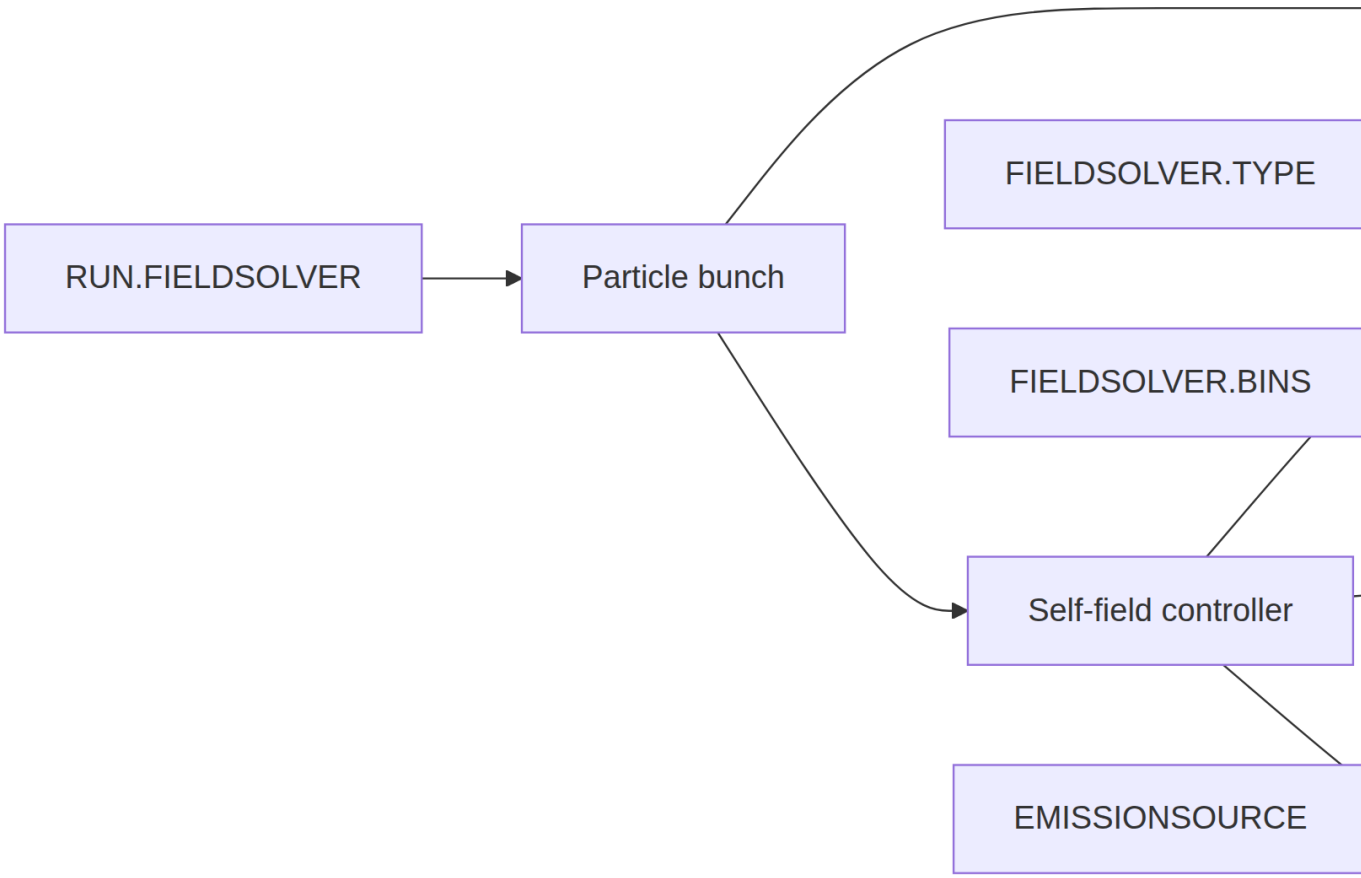
18.2 `FIELDSOLVER` parameters

Parameter	Default	Accepted values / unit	Current effect
`TYPE`	required	`NONE`, `FFT`, `OPEN`, `CG`	Selects the backend described above.
`BINS`	`NONE`	`NONE` or a `BINNING` name	Selects monolithic or per-bin rest-frame space charge.
`NX`, `NY`, `NZ`	none	positive integer cell counts	Number of mesh points in x, y, and z. The parser stores real values; use integers.
`PARFFTX`	`FALSE`	Boolean	Domain decomposition in x. Must currently be `TRUE`.
`PARFFTY`	`FALSE`	Boolean	Domain decomposition in y. Must currently be `TRUE`.
`PARFFTZ`	`TRUE`	Boolean	Domain decomposition in z. Must currently be `TRUE`.
`BCFFTX`, `BCFFTY`, `BCFFTZ`	`OPEN`	`OPEN`, `PERIODIC`, `DIRICHLET`	Mesh boundary selection. All three values must match. See Boundary conditions.
`GREENSF`	`INTEGRATED`	`INTEGRATED`, `STANDARD`	Green-function discretization for `TYPE=OPEN`; otherwise ineffective.
`BBOXINCR`	`2.0`	percent	Adds this fraction of the particle span to each side of the moving mesh.

The compiled `PARFFTX` and `PARFFTY` defaults are not usable defaults: current OPALX rejects any definition unless `PARFFTX`, `PARFFTY`, and `PARFFTZ` are all `TRUE`. Mixed boundary conditions, such as periodic x, y with open z, are also rejected.

18.3 Mesh and particle-in-cell cycle

The runtime objects share one distributed field layout containing charge density, electric field, potential, and scratch fields. Each active space-charge evaluation performs the following cycle:



The mesh follows the particle bounds. `BBOXINCR=p` expands both the lower and upper bound by p percent of the unpadded span, so the total width grows by $2p$ percent. A degenerate span is clamped to $1\,\mu$m. If load balancing repartitions the field layout, OPALX updates fields and particles before refreshing backend-owned FFT state.

Zero or one global particle produces zero self-field without a solve. For two or more particles, an active backend rejects zero per-particle charge; this usually means `BCHARGE` is missing from `BEAM`.

18.4 Boundary conditions

18.4.1 Open and periodic boundaries

Use matching solver and boundary settings:

Physical model	Configuration
Isolated finite bunch	`TYPE=OPEN`; all `BCFFT*=OPEN`
Periodically repeated box	`TYPE=FFT`; all `BCFFT*=PERIODIC`
No self-field	`TYPE=NONE`; all `BCFFT*=OPEN` is conventional

The three axes must use the same boundary value. The boundary attributes set mesh periodicity, but they do not change `FFT` into an open solver or `OPEN` into a periodic solver; select the matching `TYPE` explicitly.

18.4.2 Generic all-face Dirichlet boundaries

The parser accepts `BCFFTX=DIRICHLET`, `BCFFTY=DIRICHLET`, and `BCFFTZ=DIRICHLET` together. These values would impose zero potential on all six mesh faces for the `CG` potential solver. Since `CG` is currently disabled, this is **not a usable mode** on current master.

Cathode-plane Dirichlet corrections are different: keep `TYPE=OPEN` and all three `BCFFT*` values `OPEN`, then select one correction on `EMISSIONSOURCE`.

18.5 Space-charge modes

`TYPE` chooses a backend. `BINS` independently chooses whether OPALX performs one lab-frame electrostatic solve or several rest-frame solves. An emission source can additionally request one cathode-plane correction.

18.5.1 Monolithic electrostatic mode

`BINS=NONE` performs one scatter, one solve, and one electric-field gather for the entire bunch. It applies no rest-frame transformation and does not create a self-magnetic field. This is the least expensive mode and is appropriate only when that electrostatic approximation is adequate.

Explicit image charges can be included in this path. A requested shifted-Green correction is not applied without binning; OPALX emits a warning and proceeds with the uncorrected monolithic solve.

18.5.2 Binned rest-frame mode

Set `BINS` to a named `BINNING` definition. At every field evaluation OPALX rebins particles, then for each populated bin:

1. computes its mean normalized momentum $\mathbf{p}_b$ and mean γ_b;
2. stretches the longitudinal mesh spacing by γ_b;
3. deposits that bin's charge and solves in its approximate rest frame;
4. transforms the result to the laboratory frame; and
5. accumulates electric and magnetic contributions from all bins.

For $\mathbf{v}_b = c\mathbf{p}_b/\gamma_b$ and $\hat{\mathbf{w}}_b = \mathbf{v}_b/|\mathbf{v}_b|$, the transformation is

$$\mathbf{E}_b = \gamma_b \mathbf{E}'_b - (\gamma_b - 1)(\mathbf{E}'_b \cdot \hat{\mathbf{w}}_b)\hat{\mathbf{w}}_b, \qquad \mathbf{B}_b = \frac{\gamma_b}{c^2}\mathbf{v}_b \times \mathbf{E}'_b.$$

Use binning when velocity varies substantially through the bunch, especially during emission or in a chirped beam. Check convergence with respect to both mesh resolution and binning parameters.

18.5.3 Explicit image-charge Dirichlet correction

Set `ZEROFACE_ROZ` to `TRUE` on one emission source to model a grounded plane at that source's `ROZ`. For each real charge q at z, OPALX deposits a mirror charge $-q$ at $2R0Z - z$. Their superposed potential is zero on the plane.

```
Source: EMISSIONSOURCE, DISTRIBUTION=Dist,
    ROZ=0.0,
    ZEROFACE_ROZ=TRUE,
    ZEROFACE_MAXSTEPS=500,
    ZEROFACEPLANEDUMP=0;
```

The mode is implemented for monolithic and binned execution. Use it with the `OPEN` backend; this is the physically consistent and maintained setup, though the current input checks do not reject other backends. In binned mode the real and mirror charges require separate solves per bin.

The moving mesh must span both the bunch and its mirror. Once the bunch is far from the cathode, this can make the mesh unnecessarily large or lose the required resolution. Set `ZEROFACE_MAXSTEPS` to a positive cutoff when the correction becomes negligible; `0` means unlimited. `ZEROFACEPLANEDUMP` is a non-negative diagnostic frequency, is available only for this explicit-image mode, and currently writes only in single-rank runs.

18.5.4 Shifted-Green Dirichlet correction

Set `SHIFTED_GREENS_FUNCTION` to `TRUE` to impose the same grounded plane without placing mirror particles on the mesh. It repeats each binned primary deposition using a shifted free-space Green function, reflects the resulting field in z, and adds the image contribution with the required component signs.

```
Source: EMISSIONSOURCE, DISTRIBUTION=Dist,
    ROZ=0.0,
    SHIFTED_GREENS_FUNCTION=TRUE,
    ZEROFACE_MAXSTEPS=500;
```

This mode **requires both `TYPE=OPEN` and a named `BINS` definition**. Without binning the correction is skipped. Its kernel shift is computed from the rest-frame mesh centre z'_c and plane z_p as

$$\mathbf{s} = (0, 0, 2(z'_c - z_p)).$$

Unlike explicit images, the mesh need not extend from the bunch to the mirror distribution, so the method remains usable at larger bunch-plane separation. It does not support `ZEROFACEPLANEDUMP`.

The explicit-image and shifted-Green modes are mutually exclusive across the entire run. Only one emission source may select either method, and both use the same `ZEROFACE_MAXSTEPS` convention.

18.6 Compatibility and selection

Goal	Backend and boundaries	Space-charge mode	Cathode correction
Disable self-fields	`NONE`; conventionally `OPEN` boundaries	`BINS=NONE`	none
Periodic electrostatics	`FFT`; all `PERIODIC`	monolithic or binned	none
Isolated, nearly mono-velocity bunch	`OPEN`; all `OPEN`	monolithic	optional explicit images
Isolated bunch with velocity spread	`OPEN`; all `OPEN`	named `BINS`	none or explicit images
Emission near a grounded plane	`OPEN`; all `OPEN`	named `BINS`	shifted Green recommended for separation from the plane

Do not select `CG`, generic `DIRICHLET`, or `P3M` for a current production input. Do not combine cathode corrections with `FFT` periodic physics.

18.7 Accuracy, cost, and reproducibility

- A monolithic step performs one Poisson solve. A binned step performs one per populated bin; either cathode correction doubles that count.
- Increase `NX`, `NY`, and `NZ` independently and compare physical observables. A visually smooth field is not a convergence test.
- Vary `BBOXINCR`: too little padding places particles close to the mesh edge; too much padding lowers resolution for fixed mesh dimensions.
- For binned runs, vary the `BINNING` controls and check that observables do not depend materially on the chosen partition.
- Compare correction cutoffs and confirm that fields are negligible before `ZEROFACE_MAXSTEPS` disables the cathode model.
- Start from a maintained regression input with the intended backend. Record all mesh, boundary, Green-function, binning, and correction settings with the result.

OPAL

Space-charge effects are attached to the tracking setup through a `FIELDSOLVER` command and then referenced from `RUN` as described in the tracking chapter. The standard model is fully three-dimensional. The field solve is based on Poisson's equation with open boundaries, either through FFT-based convolution, iterative irregular-domain solvers, or adaptive mesh refinement.

18.8 FFT Based Particle-Mesh (PM) Solver

The classical particle-mesh solver discretizes the charge density on a regular Cartesian mesh

$$\Omega = [-L_x, L_x] \times [-L_y, L_y] \times [-L_t, L_t]$$

with the corresponding grid points. Instead of computing all pairwise particle-particle interactions, the charge is distributed across the mesh and Poisson's equation is solved there.

For isolated bunches the potential satisfies

$$\nabla^2 \phi = -\rho/\epsilon_0 \tag{18.1}$$

and can be written as a convolution of the charge density with the open-space Green function. The FFT-based solver uses the Hockney zero-padding construction so that the open-boundary convolution can be evaluated with periodic FFTs on an enlarged mesh.

18.8.1 FFT-based Convolutions and Zero Padding

The key idea is:

1. deposit the physical charge density on the unpadded mesh
2. enlarge the mesh so the convolution can be represented as a periodic one
3. construct the periodic Green-function array
4. solve the convolution in Fourier space
5. retain the potential and field only in the physical region

This gives the physical open-boundary solution while preserving the efficiency of FFTs.

18.8.2 Algorithm

In practical terms, the FFT space-charge solve proceeds as:

1. deposit charge to the mesh
2. build or reuse the Green-function array
3. perform the FFT-based convolution solve
4. compute the electric field on the mesh
5. interpolate the field back to particles

The method scales much better than direct particle-particle summation and is the standard stable space-charge path.

18.8.3 Interpolation Schemes

The interpolation from mesh fields back to particles follows the standard particle-mesh deposition/interpolation choices used. The legacy manual reserves the detailed interpolation discussion for later revisions, but the solver is intended for ordinary 3D PIC workflows.

18.9 Particle-Particle-Particle-Mesh (P3M) Solver

The `P3M` solver splits the total electrostatic force into a short-range direct part and a long-range mesh part,

$$F = F_{\mathrm{sr}} + F_{\mathrm{lr}} \tag{18.2}$$

so that close encounters are handled by direct summation inside a cutoff radius, while the smooth long-range contribution is handled on the mesh.

In this formulation the Green function is likewise split into short-range and long-range components. `OPAL` supports two practical choices:

- STANDARD: the Ewald-type splitting with interaction parameter ALPHA
- INTEGRATED: an integrated polynomial form with a convenient closed-form cell integration

18.9.1 Use of P^3M solver

The legacy implementation notes are important:

- P3M is only available in OPAL-T
- emission must not be active
- the solver uses OPEN boundary conditions
- ALPHA is only used together with GREENSF=STANDARD

Example:

```
REAL inter_rad = 3.125e-5;

FS_P3M: FIELDSOLVER, FSTYPE="P3M", MX=64, MY=64, MT=64,
        PARFFTX=decx, PARFFTY=decy, PARFFTT=decz,
        RC=inter_rad, GREENSF=INTEGRATED;
```

18.10 Iterative Space Charge Solver

The iterative Poisson solver is intended for irregular geometries where the plain FFT open-boundary solve is not the best model. The discretization is based on finite differences, and the resulting linear system is solved by a preconditioned iterative method with smoothed-aggregation algebraic multigrid preconditioning.

Compilation requirements:

- ENABLE_SAAMG_SOLVER=ON
- Parmetis
- Trilinos

This is the solver family behind FSTYPE=SAAMG.

18.11 Energy Binning

When the bunch energy spread is too large for a single boosted-frame electrostatic approximation, OPAL can split the bunch into several energy bins and perform multiple field solves. This is the basis of the multi-Lorentz-frame approximation.

In cyclotron multi-bunch mode the number of energy bins should be at least the number of neighboring bunches represented by NNEIGHBB. The companion control parameter

MINSTEPFORREBIN defines after how many integration steps the energy bins are merged again.

18.12 The FIELDSOLVER Command

The FIELDSOLVER command selects the solver type, mesh resolution, domain decomposition, boundary conditions, and optional solver-specific parameters.

18.12.1 Core command attributes

Attribute	Meaning
FSTYPE	Solver type: FFT, FFTPERIODIC, SAAMG, P3M, NONE
PARFFTX	Distribute the x direction over MPI ranks
PARFFTY	Distribute the y direction over MPI ranks
PARFFTT	Distribute the z direction over MPI ranks
MX	Number of mesh points in x
MY	Number of mesh points in y
MT	Number of mesh points in z
BCFFTX	Boundary condition in x
BCFFTY	Boundary condition in y
BCFFTZ	Boundary condition in z
GREENSF	Green-function choice for FFT-based solvers
BBOXINCR	Bounding-box enlargement factor in percent
GEOMETRY	Geometry list for irregular-domain solves
ITSOLVER	Iterative linear solver
INTERPL	Boundary-point interpolation scheme
TOL	Iterative solver tolerance
MAXITERS	Maximum number of iterations
PRECMODE	Preconditioner reuse policy
RC	P3M short-range cutoff radius
ALPHA	P3M interaction splitting parameter
ENBINS	Number of energy bins
MINSTEPFORREBIN	Merge energy bins after this many steps

18.12.2 Fieldsolver type

FSTYPE selects the actual solver backend.

- FFT: standard open-boundary FFT-based particle-mesh solver
- FFTPERIODIC: FFT solver with periodic longitudinal treatment
- SAAMG: iterative irregular-domain solver
- P3M: mixed mesh plus short-range direct solver

- `NONE`: disable space-charge field solving

`FFT` solver is the most stable default path.

18.12.3 Domain Decomposition

`PARFFTX`, `PARFFTY`, and `PARFFTT` decide whether the corresponding mesh direction is distributed over MPI processes.

Legacy default:

- `PARFFTX = FALSE`
- `PARFFTY = FALSE`
- `PARFFTT = TRUE`

So the conventional decomposition is serial in `x` and `y`, parallel in `z`.

18.12.4 Number of Grid Points

`MX`, `MY`, and `MT` define the rectangular mesh resolution in `x`, `y`, and `z`. These values determine both the field resolution and the FFT problem size, so they are the main tuning parameters for accuracy and cost.

18.12.5 Boundary Conditions

Boundary conditions are specified independently per axis:

- `BCFFTX`: `OPEN`
- `BCFFTY`: `OPEN`
- `BCFFTZ`: `OPEN` or `PERIODIC`

Using `PERIODIC` in `z` is the standard way to model a DC beam rather than an isolated bunch.

18.12.6 Greens Function

`GREENSF` controls the Green-function model for FFT-based solvers and `P3M`.

Allowed values:

- `INTEGRATED`
- `STANDARD`

The default is `INTEGRATED`. This is the usual recommendation for FFT-based space-charge calculations.

18.12.7 Bounding Box Enlargement

The solver constructs a minimal rectangular box that encloses all particles. `BBOXINCR` enlarges that box by a user-specified percentage before the solve.

Default:

- `BBOXINCR = 2.0`

This is often used to avoid an overly tight computational box around the bunch.

18.12.8 Geometry

`GEOMETRY` attaches a list of geometry objects that define the boundary of the computational domain for the irregular-domain solver. This option is relevant to `SAAMG` workflows.

18.12.9 Iterative Solver

`ITSOLVER` selects the Krylov or direct linear solver used by the iterative field-solver family.

For `SAAMG`, the documented choices are:

- `CG` (default)
- `BICGSTAB`
- `GMRES`

18.12.10 Interpolation for Boundary Points

`INTERPL` controls how grid points near the irregular boundary are interpolated.

Allowed values:

- `CONSTANT`
- `LINEAR` (default)
- `QUADRATIC`

This parameter is specific to `SAAMG`.

18.12.11 Tolerance

`TOL` is the convergence tolerance for the iterative solver.

Default:

- `TOL = 1e-8`

18.12.12 Maximal Iterations

MAXITERS limits the number of iterations taken by the iterative solver.

Default:

- MAXITERS = 100

18.12.13 Preconditioner Behavior

PRECMODE controls how aggressively the SAAMG preconditioner is rebuilt or reused.

Value	Behavior
STD	Rebuild the preconditioner every step
HIERARCHY	Reuse the hierarchy only; this is the default
REUSE	Reuse the full preconditioner

This parameter should only be changed deliberately, because it affects both performance and solver robustness.

18.12.14 Cut-off Radius

RC is the short-range cutoff radius for the P3M solver in the boosted frame. Particles separated by less than this radius are included in the direct particle-particle correction.

Default:

- RC = 0

This corresponds to disabling the direct short-range correction.

18.12.15 Interaction splitting parameter

ALPHA controls the balance between the short-range and mesh parts of the P3M Green-function split. Large ALPHA gives greater weight to the mesh part; small ALPHA gives greater weight to the particle-particle part.

It is commonly chosen as

$$\alpha = C/r_c \tag{18.3}$$

with C of order one and r_c the cutoff radius.

Default:

- ALPHA = 1e8

This parameter is only used for GREENSF=STANDARD.

18.12.16 Number of Energy Bins

ENBINS selects how many energy bins are used when the bunch is split for multi-frame space-charge treatment. In cyclotron multi-bunch mode this should not be smaller than NNEIGHBB.

MINSTEPFORREBIN defines after how many steps the temporarily split energy-bin representation is merged back into one.

18.13 Adaptive Mesh Refinement (AMR) Solver

The AMR path extends the FIELDSOLVER command with hierarchy and refinement controls. It is enabled by compiling OPAL with ENABLE_AMR=ON and enabling OPTION, AMR=TRUE.

The legacy interface describes three AMR solver types:

- FMG
- ML
- AMR_MG

The hierarchy is defined from the base grid by the number of AMR levels, refinement ratios, blocking factors, and maximum grid sizes.

18.13.1 AMR extensions to FIELDSOLVER

Attribute	Meaning
AMR_MAXLEVEL	Maximum AMR refinement level
AMR_REFX, AMR_REFY, AMR_REFZ	Refinement ratio per coordinate
AMR_MAXGRIDX, AMR_MAXGRIDY, AMR_MAXGRIDZ	Maximum base-level box size
AMR_BFX, AMR_BFY, AMR_BFZ	Blocking factors
AMR_TAGGING	Mesh-refinement rule
AMR_DENSITY	Density threshold for CHARGE_DENSITY tagging
AMR_MAX_NUM_PART	Threshold for MAX_NUM_PARTICLES tagging
AMR_MIN_NUM_PART	Threshold for MIN_NUM_PARTICLES tagging
AMR_SCALING	Scaling threshold for POTENTIAL, EFIELD, MOMENTA
AMR_DOMAIN_RATIO	Computational-domain aspect ratio

18.13.2 Mesh refinement strategies

The manual documents these tagging modes:

AMR_TAGGING	Refinement criterion
POTENTIAL	refine where the potential exceeds a scaled level maximum
EFIELD	refine where any field component exceeds a scaled level maximum
MOMENTA	refine cells containing particles above a scaled momentum threshold
CHARGE_DENSITY	refine cells whose density exceeds AMR_DENSITY
MIN_NUM_PARTICLES	refine cells based on AMR_MIN_NUM_PART
MAX_NUM_PARTICLES	refine cells based on AMR_MAX_NUM_PART

AMR_TAGGING is a string attribute and therefore must be quoted in the input deck.

18.13.3 Hardware-Architecture Independent AMR Poisson Solver

The AMR_MG path is the Trilinos-based AMR Poisson solver. It requires ENABLE_AMR_MG_SOLVER=ON and a working Trilinos installation. The legacy manual lists these core packages:

- Tpetra
- Ifpack2
- Amesos2
- Belos
- MueLu

Some base-level linear solvers require additional third-party packages such as SUPERLU, UMFPACK, PARDISO_MKL, MUMPS, or LAPACK.

The AMR multigrid extensions are:

Attribute	Meaning
AMR_MG_SMOOTHER	GS, JACOBI, or SGS; default GS
AMR_MG_NSWEEPS	Number of smoothing sweeps; default 8
AMR_MG_PREC	Bottom-level preconditioner; default NONE
AMR_MG_INTERP	Prolongation operator; default PC
AMR_MG_NORM	Convergence norm; default LINF_NORM
AMR_MG_VERBOSE	Write solver diagnostics; default FALSE
AMR_MG_REBALANCE	Rebalance solver/preconditioner communicators; default FALSE
AMR_MG_REUSE	MueLu reuse mode; default RAP

Attribute	Meaning
AMR_MG_TOL	Solver tolerance; default 1e-10
ITSOLVER	Base-level linear solver; default CG

For ITSOLVER on the AMR base level, the legacy interface documents:

- BICGSTAB
- MINRES
- PCPG
- CG
- GMRES
- STOCHASTIC_CG
- RECYCLING_CG
- RECYCLING_GMRES
- KLU2
- SUPERLU
- UMFPACK
- PARDISO_MKL
- MUMPS
- LAPACK
- SA

18.13.4 Use of AMR

The legacy manual states that AMR is available only in OPAL-cycl multi-bunch mode. Once more than one bunch is present, the AMR hierarchy is built. In this setup the AMR backend manages the MPI parallelism itself, so the other Poisson-solver backends are not combined with the AMR run mode.

18.13.5 AMR Example

```
OPTION, AMR              = TRUE;
OPTION, AMR_REGRID_FREQ  = 10;
OPTION, AMR_YT_DUMP_FREQ = 100000;

REAL Edes      = .072;
REAL turns     = 8;
REAL nstep     = 360;
REAL frequency = 50.650;

ring: CYCLOTRON, ...;
rf0: RFCAVITY, ...;
```

```
rf1: RFCAVITY, ...;
rf2: RFCAVITY, ...;
rf3: RFCAVITY, ...;
rf4: RFCAVITY, ...;
l1: LINE = (ring, rf0, rf1, rf2, rf3, rf4);

Dist1: DISTRIBUTION, ...;

Fs1: FIELDSOLVER, FSTYPE=AMR_MG,
                  MX=128, MY=128, MT=128,
                  PARFFTX=TRUE, PARFFTY=TRUE, PARFFTT=TRUE,
                  BCFFTX=OPEN, BCFFTY=OPEN, BCFFTZ=OPEN,
                  BBOXINCR=20, AMR_MAXLEVEL=2,
                  AMR_MAXGRIDX=32, AMR_MAXGRIDY=32, AMR_MAXGRIDZ=32,
                  AMR_BFX=16, AMR_BFY=16, AMR_BFZ=16,
                  AMR_REFX=2, AMR_REFY=2, AMR_REFZ=2,
                  AMR_DOMAIN_RATIO={1.0, 0.75, 0.75},
                  AMR_TAGGING="CHARGE_DENSITY", AMR_DENSITY=1.0e-9,
                  AMR_MG_VERBOSE=TRUE, AMR_MG_REBALANCE=TRUE, AMR_MG_REUSE=FULL,
                  ITSOLVER=SA, AMR_MG_NORM=LINF_NORM, AMR_MG_NSWEEPS=12;

beam1: BEAM, PARTICLE=PROTON, PC=P0, NPART=1e5, BCURRENT=2.0E-3, CHARGE=1.0, BFREQ=frequen

SELECT, LINE=l1;

TRACK, LINE=l1, BEAM=beam1, MAXSTEPS=nstep*turns, STEPSPERTURN=nstep;
RUN, METHOD="CYCLOTRON-T", BEAM=beam1, FIELDSOLVER=Fs1, DISTRIBUTION=Dist1,
    MBMODE=FORCE, TURNS=11, MB_BINNING=GAMMA_BINNING, MB_ETA=0.25;

ENDTRACK;
STOP;
```

19 Tracking

A tracking block selects a lattice and beam, defines its step schedule, starts the tracker, and returns to normal input processing:

```
TRACK, LINE=Line1, BEAM=Beam1,
       DT={5e-11}, MAXSTEPS={100}, ZSTOP={1.0};
  RUN, METHOD=PARALLEL, FIELDSOLVER=FS1;
ENDTRACK;
```

19.1 TRACK

`TRACK` enters tracking mode. Only tracking commands such as `RUN` and `ENDTRACK` are accepted until the block closes.

OPALX

Parameter	Default	Current behavior
`LINE`	required	Name of the `LINE` to track.
`BEAM`	`UNNAMED_BEAM`	Single beam name. Set it explicitly for normal runs.
`BEAMS`	empty	Ordered beam-name array. A non-empty array takes precedence over `BEAM`.
`SOURCES`	empty	Accepted for compatibility but ignored; sources come from each selected `BEAM`.
`DT`	`{1e-12}` s	Time-step schedule. Use positive values.
`MAXSTEPS`	`{10}`	Maximum-step schedule. Negative values are rejected; values are converted to integers.

Parameter	Default	Current behavior
ZSTART	0 m	Starting reference position.
ZSTOP	{1e6} m	Stop-position schedule.
T0	0 s	Accepted and stored, but not currently forwarded to ParallelTracker; source timing uses EMISSIONSOURCE.T0.
DTSCINIT	1e-12 s	Accepted adaptive-integrator setting; not used by the current parallel path.
DTAU	-1	Accepted adaptive-integrator accuracy setting; not used by the current parallel path.
TIMEINTEGRATOR	RK4	Accepts RK-4, RK4, LF-2, LF2, or MTS, but does not currently select the ParallelTracker integration kernel.
STEPSPERTURN	720	OPAL-CYCL compatibility attribute; inactive in OPALX tracking.
MAP_ORDER	1	Stored compatibility value for map tracking; inactive in the current parallel path.

DT, MAXSTEPS, and ZSTOP may contain different numbers of entries. OPALX extends each shorter array by repeating its final value until all three have the same length. This defines consecutive tracking segments.

19.1.1 Beam selection

```
TRACK, LINE=Line1, BEAMS={ElectronBeam, PositronBeam},
       DT=1e-10, MAXSTEPS=2000, ZSTOP=60.0;
```

BEAMS takes precedence whenever it is non-empty. Each entry must resolve to a BEAM, and OPALX creates one particle container per entry. The order is significant: the first beam

supplies the reference state used to construct the tracking block. Photon beams are rejected when RUN starts.

See Beam Definitions for per-container properties and source selection.

OPAL

Before tracking starts, a LINE and a BEAM must be selected. The compact legacy form is:

```
TRACK, LINE=name, BEAM=name, DT={...}, MAXSTEPS={...}, ZSTOP={...};
```

Parameter	Legacy OPAL behavior
LINE	Beamline or sequence to track.
BEAM	Beam definition providing particle mass, charge, and reference momentum.
TO	Initial simulation time in seconds; default 0.
DT	Time-step array; default {1e-12} s.
MAXSTEPS	Maximum-step array; default {10}.
ZSTART	Starting reference-particle position in metres.
ZSTOP	Longitudinal thresholds at which the next schedule segment becomes active.

If the schedule arrays have different lengths, the final supplied value is reused for the remaining segments.

19.2 RUN

RUN starts or continues particle tracking with the current bunch state.

OPALX

Parameter	Default	Current behavior
METHOD	required	Only PARALLEL is accepted.
FIELDSOLVER	required	Name of the FIELDSOLVER used for the run.
BOUNDARYGEOMETRY	NONE	Optional named boundary geometry.
TURNS	1	Accepted compatibility value; inactive in the current parallel path.

Parameter	Default	Current behavior
TRACKBACK	FALSE	Accepted compatibility value; reverse tracking is not currently applied.

RUN does not register BEAM, BEAMS, SOURCES, or DISTRIBUTION. Selection belongs on TRACK and each BEAM; distribution objects are reached through the beam's emission-source list.

OPAL

The retained non-CYCL OPAL form is:

```
RUN, METHOD="PARALLEL-T", BEAM=beam1, FIELDSOLVER=Fs1,
     BOUNDARYGEOMETRY=Geometry1, TRACKBACK=FALSE,
     DISTRIBUTION=Dist1;
```

Parameter	Legacy OPAL behavior
METHOD	PARALLEL-T selects OPAL-T tracking.
FIELDSOLVER	Field-solver definition used for the run.
DISTRIBUTION	One distribution or a distribution list.
BEAM	Beam definition used for the run.
BOUNDARYGEOMETRY	Boundary geometry used for particle termination.
TRACKBACK	Selects OPAL-T reverse-time tracking.

A later RUN continues from the current bunch state unless the input explicitly reinitializes the bunch.

19.3 ENDTRACK

```
ENDTRACK;
```

OPALX

ENDTRACK closes the tracking parser after RUN and releases the temporary tracking communication object.

OPAL

ENDTRACK leaves track mode and returns the parser to ordinary command mode.

OPALX

19.4 Current OPALX step sequence

At each step `ParallelTracker` performs a half position push, assembles space-charge and external fields, emits particles, applies the Boris momentum update and second half push, advances time, and updates the reference state. The Physics Overview describes this split.

The time step must resolve the shortest relevant field, geometry, emission, and collective-effect scale. Establish convergence by repeating a calculation with smaller steps and comparing physical observables.

OPALX

19.5 Current OPALX parallel execution

OPALX combines MPI with the selected Kokkos backend. GPU runs normally use one MPI rank per GPU and pass `--kokkos-map-device-id-by=mpi_rank` so ranks select distinct devices.

20 Wakefields

OPALX

`WAKE` and `FILTER` are not registered by the current OPALX executable. The OPAL view preserves the complete legacy interface as a compatibility reference and a visible reminder of functionality that may be ported later.

OPAL

`OPAL-T` provides methods to model coherent synchrotron radiation and short-range geometric wakefields.

20.1 Geometric Wakefields

The legacy wakefield interface supports two main routes:

1. compute the wakefield of a round metallic beam pipe from an impedance model
2. import a discretized wake function from file

For a round metallic beam pipe with radius `a`, `OPAL` distinguishes `DC` and `AC` conductivity models. The classical Drude-style conductivities are

$$\sigma_{\mathrm{DC}} = \frac{ne^2\tau}{m} \tag{20.1}$$

and

$$\sigma_{\mathrm{AC}} = \frac{\sigma_{\mathrm{DC}}}{1 - i\omega\tau}. \tag{20.2}$$

With these ingredients, the longitudinal impedance can be transformed back into the longitudinal wake. For the `DC` case the impedance model is

$$Z_{L,\mathrm{DC}}(k) = \frac{1}{ca}\frac{2}{\lambda/k - ika/2} \tag{20.3}$$

with

$$\lambda = \sqrt{\frac{2\pi\sigma|k|}{c}}\,(i + \mathrm{sign}(k))\,. \tag{20.4}$$

The longitudinal wake is then obtained from the inverse cosine transform of the real part of the impedance,

$$W_L(s) = 10^{-12}\frac{2c}{\pi}\Re\left(\int_0^\infty \Re(Z_L(k))\cos(ks)\,dk\right), \tag{20.5}$$

and the transverse wake follows through the Panofsky-Wenzel relation.

OPAL evaluates these integrals numerically with Simpson integration on an equidistant wave-number mesh. Since this is usually done once during initialization, the computational overhead is typically acceptable.

20.2 CSR Wakefields

The CSR model in OPAL-T is built around one-dimensional steady-state or quasi-steady-state approximations. The derivation starts from the Liénard-Wiechert fields of particles moving on a circular orbit and leads to impedance and wakefield models that act on trailing particles only.

The legacy manual distinguishes:

- 1D-CSR
- 1D-CSR-IGF

The 1D-CSR model follows the staged bend-entry, steady-state, and exit picture of Saldin, Schneidmiller, and Yurkov. The 1D-CSR-IGF variant uses an integrated Green function for faster wake evaluation.

Both models neglect transverse structure and several transient corrections, but they capture the dominant longitudinal CSR effect for short bunches in many beamline studies.

20.3 The WAKE Command

WAKE defines the wakefield data attached to an element.

General syntax:

```
label: WAKE, TYPE=string, NBIN=integer, CONST_LENGTH=bool,
             CONDUCT=string, Z0=real, FORM=string, RADIUS=real,
             SIGMA=real, TAU=real, FNAME=string, FILTERS={...};
```

For CSR wakes the reduced syntax is:

```
label: WAKE, TYPE=string, NBIN=integer, FILTERS={...};
```

20.3.1 Core attributes

Attribute	Meaning
TYPE	Wake type: 1D-CSR, 1D-CSR-IGF, LONG-SHORT-RANGE, TRANSV-SHORT-RANGE
NBIN	Number of bins used to build the bunch line density
CONST_LENGTH	Keep bunch length constant during wake evaluation
CONDUCT	Beam-pipe conductivity model: AC or DC
ZO	Beam-pipe impedance in ohms
FORM	Beam-pipe form; the documented choice is ROUND
RADIUS	Beam-pipe radius in meters
SIGMA	Material conductivity parameter
TAU	Relaxation time entering the AC conductivity model
FNAME	SDDS file containing a tabulated wake function
FILTERS	Ordered list of filter names applied to the line density

20.3.2 Practical behavior

- TYPE decides whether the command describes CSR or a short-range geometric wake.
- CONDUCT, ZO, FORM, RADIUS, SIGMA, and TAU have no effect on CSR wake types.
- FNAME imports a wake from SDDS data. Once a file is provided, OPAL ignores the analytic round-pipe parameters and uses the imported wake.
- FILTERS are applied in the given order, and the last filter is also used to compute the derivatives needed by the wake model.

20.4 The FILTER Command

FILTER defines a smoothing or differentiation filter applied to the longitudinal histogram of the bunch.

Syntax:

```
label: FILTER, TYPE=string, NFREQ=integer, THRESHOLD=real,
               NPOINTS=integer, NLEFT=integer, NRIGHT=integer,
               POLYORDER=integer;
```

20.4.1 Filter types

Supported filters are:

- SAVITZKY-GOLAY
- STENCIL
- FIXEDFFTLOWPASS
- RELATIVEFFTLOWPASS

20.4.2 Filter attributes

Attribute	Meaning
TYPE	Select the filter algorithm
NFREQ	Number of Fourier modes to keep for FIXEDFFTLOWPASS
THRESHOLD	Relative spectral cutoff for RELATIVEFFTLOWPASS
NPOINTS	Moving-window width for SAVITZKY-GOLAY
NLEFT	Number of points to the left for SAVITZKY-GOLAY
NRIGHT	Number of points to the right for SAVITZKY-GOLAY
POLYORDER	Least-squares polynomial order for SAVITZKY-GOLAY

20.4.3 Implemented smoothing logic

SAVITZKY-GOLAY and the FFT-based filters naturally provide derivative information. For the STENCIL filter, OPAL applies two smoothing stencils consecutively:

$$f_i = \frac{7f_{i-4} + 24f_{i-2} + 34f_i + 24f_{i+2} + 7f_{i+4}}{96} \tag{20.6}$$

and

$$f_i = \frac{7f_{i-2} + 24f_{i-1} + 34f_i + 24f_{i+1} + 7f_{i+2}}{96}. \tag{20.7}$$

The derivative is then computed with the second-order stencil

$$f_i' = \frac{f_{i-2} - 8f_{i-1} + 8f_{i+1} - f_{i+2}}{h}. \tag{20.8}$$

For the FFT-based smoothers, the line-density Fourier coefficients are computed, high-frequency content is removed according to the chosen criterion, and the smoothed profile is reconstructed by inverse transform.

- `FIXEDFFTLOWPASS` keeps the `NFREQ` lowest modes.
- `RELATIVEFFTLOWPASS` keeps modes above a threshold relative to the dominant coefficient.

21 Geometry

OPALX

GEOMETRY is not registered by the current OPALX executable. Internal boundary-geometry classes remain in the source, but they do not provide a usable OPALX input definition. The OPAL view preserves the complete legacy interface.

OPAL

At present the GEOMETRY command is treated as an experimental feature. It has two roles:

1. terminate particles that cross a boundary surface
2. provide boundary information to geometry-aware field solvers such as SAAMG

The command supports two usage modes:

1. import a surface mesh from an H5hut geometry file
2. construct a predefined analytic shape: ELLIPTIC, RECTANGULAR, or BOXCORNER

21.1 Geometry Command

General form:

```
name: GEOMETRY, TOPO=ELLIPTIC, LENGTH=1.0, A=0.005, B=0.005;
```

21.1.1 Command attributes

Attribute	Meaning	Default
FGEOM	H5hut file containing a surface mesh	none
LENGTH	Geometry length in meters	1.0
TOPO	Analytic topology: ELLIPTIC, RECTANGULAR, BOXCORNER	ELLIPTIC
S	Start position of the geometry in meters	1.0
A	Semi-major axis or horizontal half aperture	0.025

Attribute	Meaning	Default
B	Semi-minor axis or vertical half aperture	0.025
C	Corner height for BOXCORNER	0.01
L1	First section length for BOXCORNER	0.5
L2	Second section length for BOXCORNER	0.2
ZSHIFT	Longitudinal shift for imported H5hut geometry	0.0
XYZSCALE	Uniform scaling factor for imported geometry	1.0
XSCALE	Scaling factor for imported x coordinates	1.0
YSCALE	Scaling factor for imported y coordinates	1.0
ZSCALE	Scaling factor for imported z coordinates	1.0
INSIDEPOINT	A point known to lie inside the imported geometry	none

21.1.2 Analytic and file-based modes

If FGEOM is provided, the file-based surface mesh is used and the analytic TOPO selection is effectively overridden.

The analytic topologies are:

- ELLIPTIC
- RECTANGULAR
- BOXCORNER

The last one is intended for piecewise box-like apertures with a corner transition controlled by C, L1, and L2.

21.1.3 Interaction with field solving and tracking

If the geometry is used only as a particle terminator, any field solver can be used and it is not necessary to attach the geometry to the FIELDSOLVER command.

If the geometry should also define field-solver boundary conditions, it must be:

1. referenced by FIELDSOLVER
2. selected again on RUN via BOUNDARYGEOMETRY

In all cases where particle termination is desired, the boundary geometry must be named on `RUN` so it is loaded and applied during tracking.

22 Particle-Matter Interactions

OPALX

`PARTICLEMATTERINTERACTION` is not registered by the current OPALX executable. Internal material and scattering classes are not a user-facing replacement. The OPAL view preserves the complete legacy interface and model description.

OPAL

Particle-matter interactions are defined through `PARTICLEMATTERINTERACTION`.

22.1 Command Overview

The core attributes are:

Attribute	Meaning
`TYPE`	Interaction handler: `SCATTERING` or `BEAMSTRIPPING`
`MATERIAL`	Surface or medium material
`ENABLERUTHERFORD`	Enable or disable large-angle Rutherford scattering
`LOWENERGYTHR`	Low-energy cutoff in MeV; particles below this are removed

The resulting interaction model is then attached to beamline elements that represent material boundaries or residual-gas regions.

22.2 The Energy Loss

The mean ionization energy loss is modeled with the Bethe-Bloch equation,

$$-\frac{dE}{dx} = \frac{Kz^2Z}{A\beta^2}\left[\frac{1}{2}\ln\left(\frac{2m_ec^2\beta^2\gamma^2T_{\max}}{I^2}\right) - \beta^2\right], \tag{22.1}$$

where `Z`, `A`, `I`, `beta`, `gamma`, and `Tmax` have their usual stopping power meaning and

$$T_{\max} = \frac{2m_e c^2 \beta^2 \gamma^2}{1 + 2\gamma m_e/M + (m_e/M)^2}. \tag{22.2}$$

The manual states that this form is used for incident `PROTON`, `DEUTERON`, `MUON`, `HMINUS`, and `H2P` beams over the documented energy ranges, and also for `ALPHA` particles in their supported range.

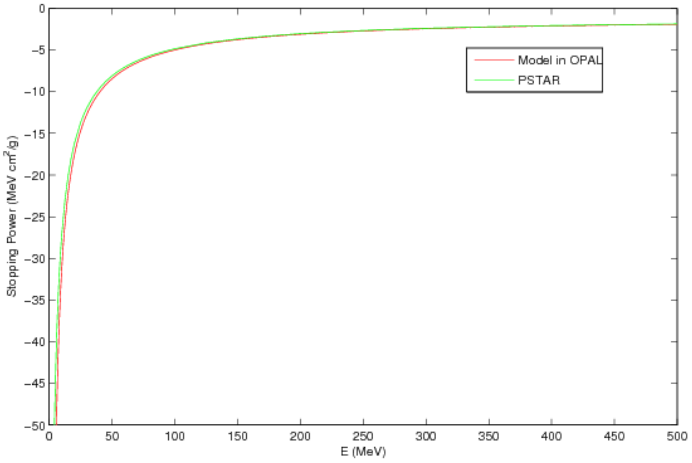


Figure 22.1: Comparison of stopping power with the PSTAR reference.

At low energy the stopping power is switched to Andersen-Ziegler-style semi-empirical fits. The implementation also models energy straggling with a Gaussian width

$$\sigma_0^2 = 4\pi N_A r_e^2 (m_e c^2)^2 \rho \frac{Z}{A} \Delta s. \tag{22.3}$$

Particles whose remaining energy drops below `LOWENERGYTHR` are removed and written to the corresponding loss file.

22.3 The Coulomb Scattering

The Coulomb-scattering model is split into:

- multiple Coulomb scattering
- large-angle Rutherford scattering

The distributions are written in the legacy manual as

$$P_M(\alpha)\, d\alpha = \frac{1}{\sqrt{\pi}} e^{-\alpha^2}\, d\alpha \tag{22.4}$$

and

$$P_S(\alpha)\,d\alpha = \frac{1}{8\ln(204Z^{-1/3})}\frac{1}{\alpha^3}\,d\alpha, \tag{22.5}$$

with transition scale

$$\theta_0 = \frac{13.6\,\mathrm{MeV}}{\beta cp}\,z\sqrt{\Delta s/X_0}\,[1 + 0.038\ln(\Delta s/X_0)]. \tag{22.6}$$

22.3.1 Multiple Coulomb Scattering

For the multiple-scattering branch, independent Gaussian random variables are used to update both transverse coordinates and transverse momenta over the substep. The model gives the cumulative small-angle deflection caused by many soft collisions in the material.

22.3.2 Large Angle Rutherford Scattering

The rare large-angle tail is handled separately as a Rutherford-scattering process. The legacy implementation samples this branch probabilistically and then draws the corresponding scattering angle from the cumulative distribution.

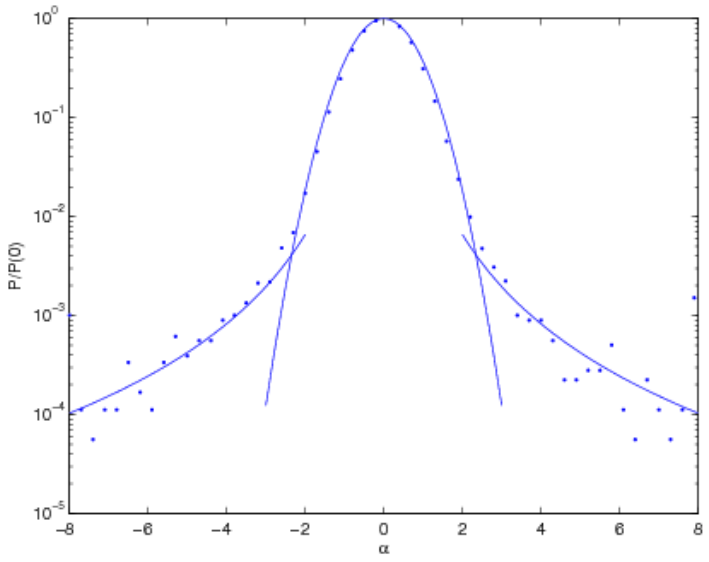


Figure 22.2: Comparison of the Coulomb-scattering angular distribution with the Jackson reference model.

22.4 Beam Stripping Physics

Beam stripping covers:

- interaction with residual gas
- electromagnetic or Lorentz stripping

The common stochastic model assumes a mean free path `lambda` and interaction probability

$$P(x) = 1 - e^{-x/\lambda}. \tag{22.7}$$

22.4.1 Residual Gas Interaction

For a gas mixture, the total inverse mean free path is the sum of the component-wise contributions. The implementation supports charge-exchange and electron-detachment or capture reactions for the incoming species documented in the legacy manual:

- `HMINUS`
- `PROTON`
- `HYDROGEN`
- `H2P`
- `DEUTERON`

The cross sections are fitted from experimental data with different families of analytic expressions depending on projectile and target:

- Nakai function
- Tabata function
- Barnett function
- Bohr function

22.4.2 Electromagnetic Stripping

For `HMINUS`, the model also accounts for magnetic-field-induced dissociation. The transverse magnetic field from the cyclotron map produces a rest-frame electric field through

$$E = \gamma\beta cB. \tag{22.8}$$

The stripping fraction during a time step `dt` is then

$$f_{\mathrm{em}} = 1 - e^{-dt/(\gamma\tau)}. \tag{22.9}$$

The manual explicitly restricts this electromagnetic stripping path to `OPAL-cycl`.

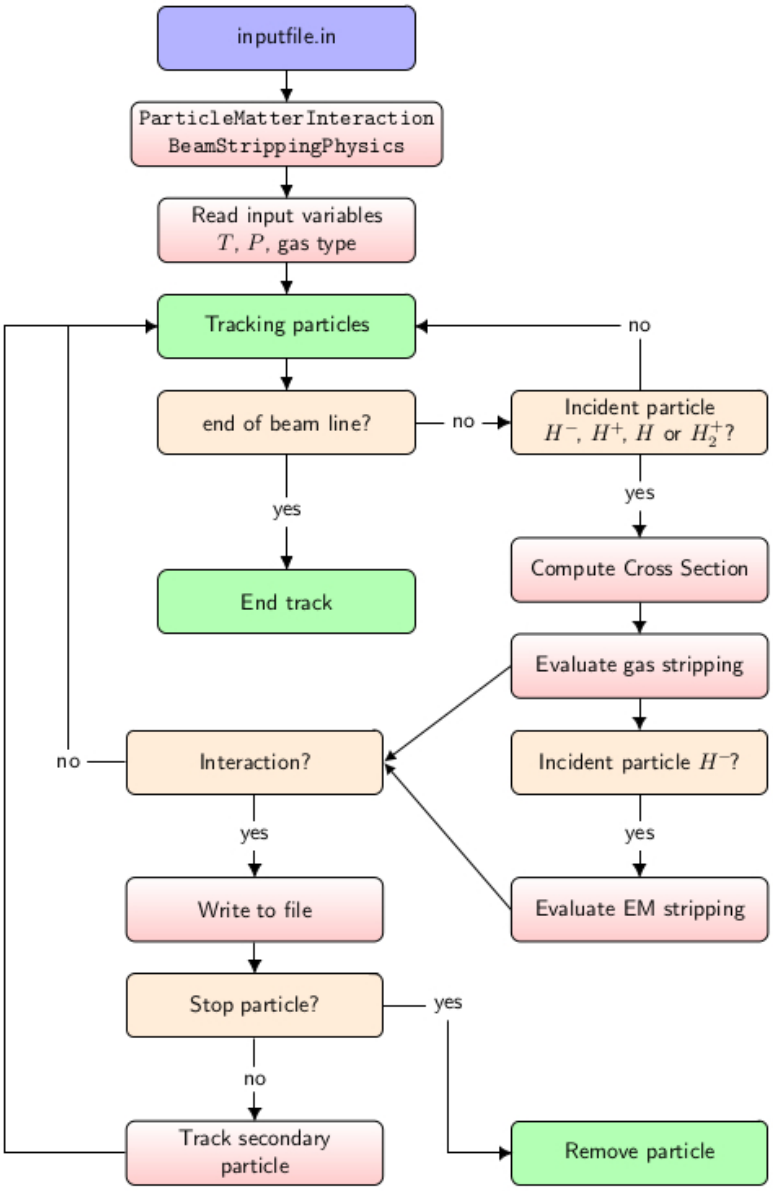


Figure 22.3: BeamStrippingPhysics flow diagram from the original manual.

22.5 The `ScatteringPhysics` Substeps

The implementation uses internal substeps when the main tracking step is too large for accurate material-interaction physics. In the legacy code path, `ScatteringPhysics.cpp` subdivides the step so that the material-interaction substep stays below the documented threshold. Particles already inside the element and particles entering the element are then advanced consistently over the same physical time interval.

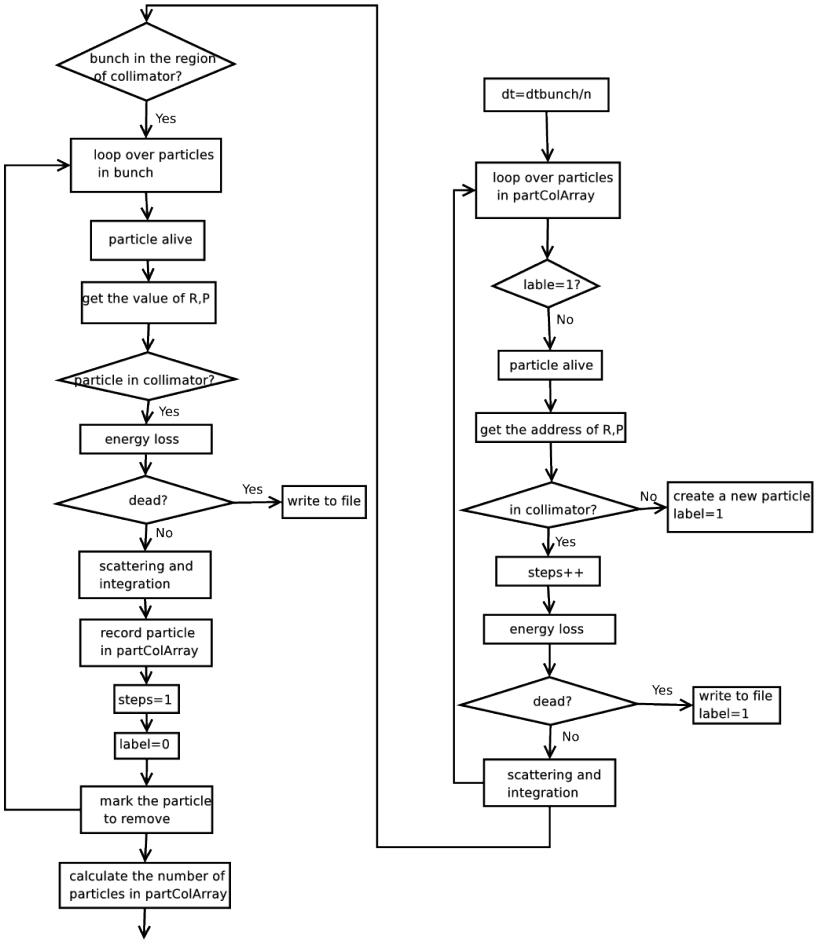


Figure 22.4: ScatteringPhysics flow diagram.

22.6 Available Materials in OPAL

The material database includes the standard beamline materials listed in the legacy manual, among them:

- `Air`
- `Aluminum`

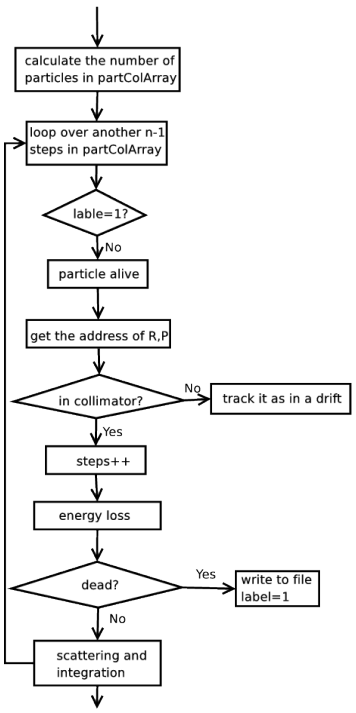


Figure 22.5: ScatteringPhysics flow diagram, continued.

- `AluminaAl2O3`
- `Beryllium`
- `BoronCarbide`
- `Copper`
- `Gold`
- `Graphite`
- `GraphiteR6710`
- `Kapton`
- `Molybdenum`
- `Mylar`
- `Titanium`
- `Water`

For each material the original manual provides:

- atomic number `Z`
- atomic weight `A`
- mass density `rho`
- radiation length `X0`
- mean excitation energy `I`
- Andersen-Ziegler low-energy fit coefficients

22.6.1 Material properties

Material	Z	A	rho [g/cm^3]	X0 [g/cm^2]	I [eV]
Air	7	14	1.205e-3	36.62	85.7
Aluminum	13	26.9815384	2.699	24.01	166.0
AluminaAl2O3	50	101.9600768	3.97	27.94	145.2
Beryllium	4	9.0121831	1.848	65.19	63.7
BoronCarbide	26	55.251	2.52	50.13	84.7
Copper	29	63.546	8.96	12.86	322.0
Gold	79	196.966570	19.32	6.46	790.0
Graphite	6	12.0172	2.210	42.7	78.0
GraphiteR6710	6	12.0172	1.88	42.7	78.0
Kapton	6	12	1.420	40.58	79.6
Molybdenum	42	95.95	10.22	9.80	424.0
Mylar	6.702	12.88	1.400	39.95	78.7
Titanium	22	47.867	4.540	16.16	233.0
Water	10	18.0152	1.0	36.08	75.0

22.6.2 Andersen-Ziegler coefficients

Material	A1	A2	A3	A4	A5	B1	B2	B3	B4	B5
Air	2.954	3.350	1.683e3	1.900e3	2.513e-2	1.9259	0.5550	27.15125	26.0665	6.2768
Aluminum	4.154	4.739	2.766e3	1.645e2	2.023e-2	2.5	0.625	45.7	0.1	4.359
AluminaAl2O3	14.270	1.343e1	1.069e4	7.723e2	2.153e-2	5.4	0.53	103.1	3.931	7.767
Beryllium	2.248	2.590	9.660e2	1.538e2	3.475e-2	2.1895	0.47183	7.2362	134.30	197.96
BoronCarbide	3.519	3.963	6065.0	1243.0	7.782e-3	5.013	0.4707	85.8	16.55	3.211
Copper	3.969	4.194	4.649e3	8.113e1	2.242e-2	3.114	0.5236	76.67	7.62	6.385
Gold	4.844	5.458	7.852e3	9.758e2	2.077e-2	3.223	0.5883	232.7	2.954	1.05
Graphite	0.0	2.601	1.701e3	1.279e3	1.638e-2	3.80133	0.41590	12.9966	117.83	242.28
GraphiteR6710	0.0	2.601	1.701e3	1.279e3	1.638e-2	3.80133	0.41590	12.9966	117.83	242.28
Kapton	0.0	2.601	1.701e3	1.279e3	1.638e-2	3.83523	0.42993	12.6125	227.41	188.97

Material	A1	A2	A3	A4	A5	B1	B2	B3	B4	B5
Molybdenum	6.424	7.248	9.545e3	4.802e2	5.376e-3	9.276	0.418	157.1	8.038	1.29
Mylar	2.954	3.350	1683	1900	2.513e-2	1.9259	0.5550	27.1512	526.0665	6.2768
Titanium	4.858	5.489	5.260e3	6.511e2	8.930e-3	4.71	0.5087	65.28	8.806	5.948
Water	4.015	4.542	3.955e3	4.847e2	7.904e-3	2.9590	0.53255	34.247	60.655	15.153

22.7 Example of an Input File

The original manual points to particlematterinteraction.in as the reference input deck. It combines material elements with collimator-style aperture settings and a particle-matter interaction definition.

22.8 A Simple Test

The documented benchmark uses a cold Gaussian beam passing through a copper slit or elliptic collimator and compares both absorbed and scattered populations as well as the downstream energy and angular spectra.

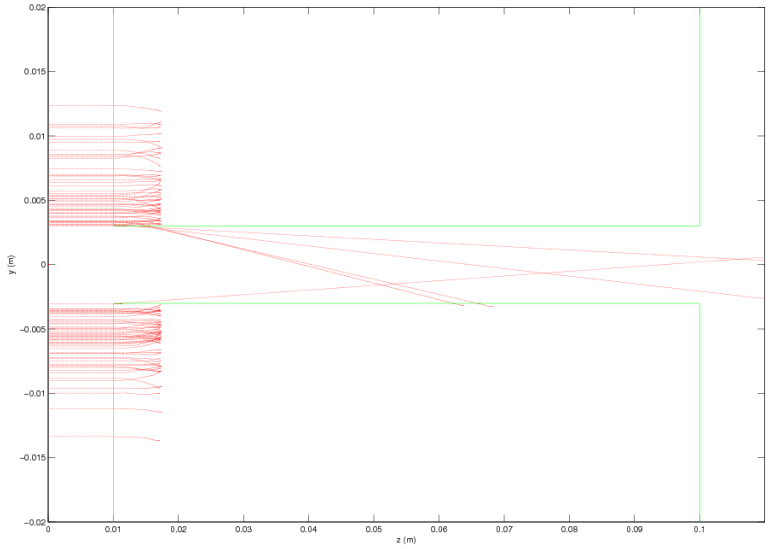


Figure 22.6: Passage of protons through the collimator.

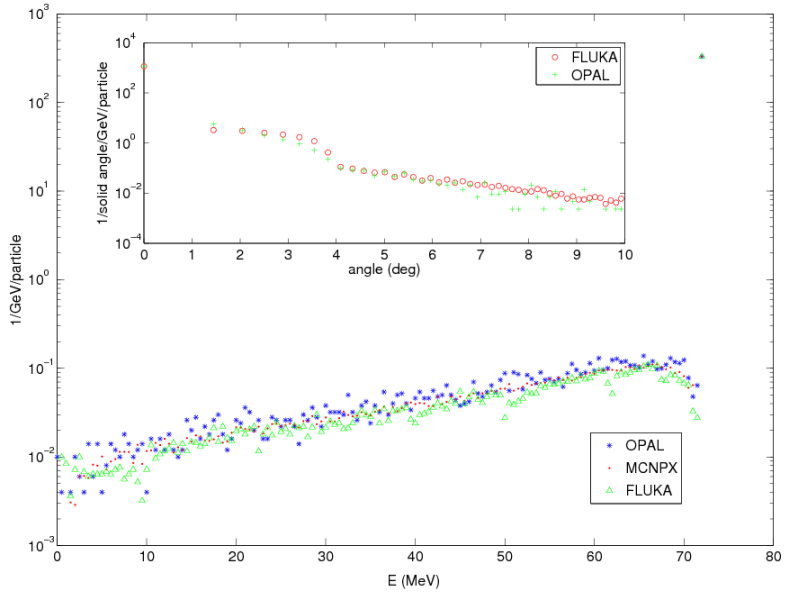


Figure 22.7: Energy spectrum and scattering angle after the elliptic collimator.

23 Multi-Objective Optimization

OPAL

Optimization problems with more than one criterion are treated in `OPAL` as multi-objective optimization problems. The legacy implementation is based on `opt-pilot`, integrated into `OPAL`.

23.1 Definition

A general multi-objective optimization problem is written as

$$\begin{aligned} \min \quad & f_m(\mathbf{x}), && m = \{1, \dots, M\}, \\ \text{subject to} \quad & g_j(\mathbf{x}) \geq 0, && j = \{1, \dots, J\}, \\ & -\infty \leq x_i^L \leq x_i \leq x_i^U \leq \infty, && i = \{0, \dots, n\}. \end{aligned} \tag{23.1}$$

The objectives are minimized over a bounded design space, optionally subject to constraints. `OPAL` treats the mapping from design variables to objective space as a black-box evaluation driven by accelerator simulations.

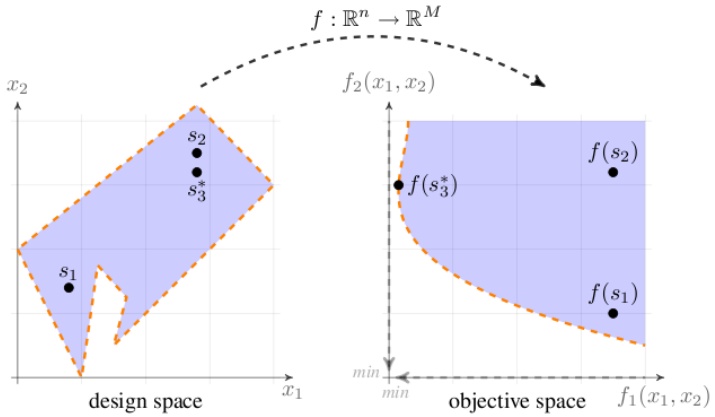


Figure 23.1: Mapping from design space to objective space.

23.2 Pareto Optimality

In most practical cases the objectives are conflicting, so no single design point minimizes all objectives at once. The optimization therefore seeks a set of non-dominated points approximating the Pareto front.

The legacy discussion emphasizes the dominance relation:

- a solution dominates another if it is no worse in all objectives
- and strictly better in at least one objective

This is the ordering relation that drives the multi-objective search.

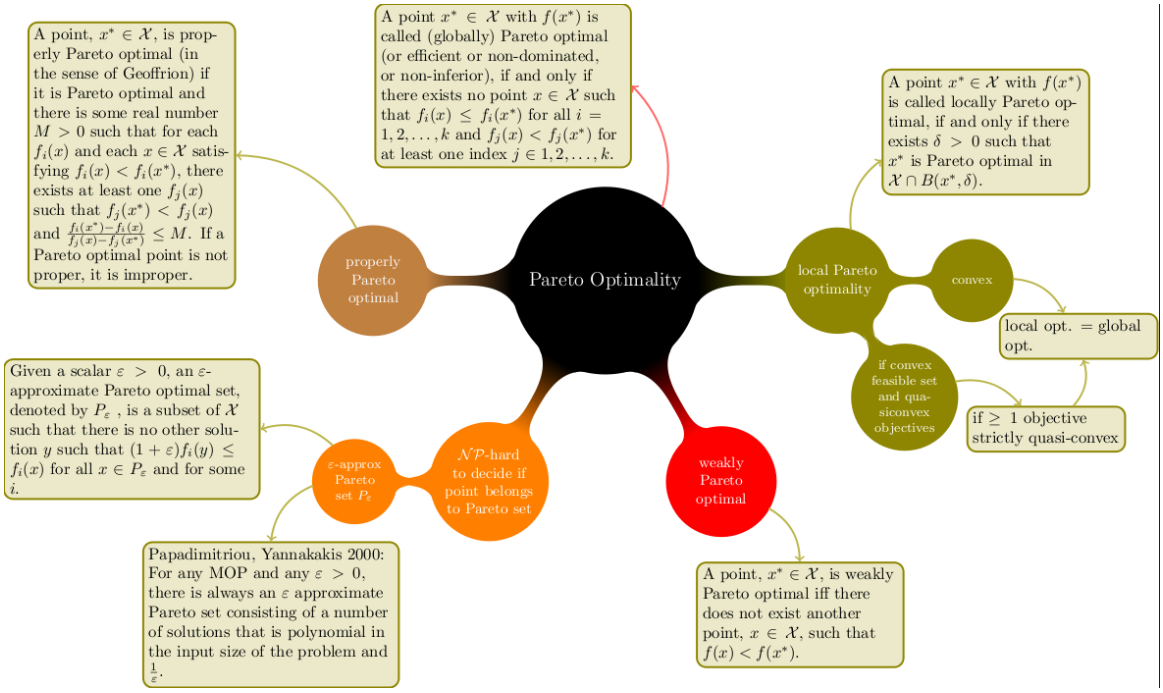


Figure 23.2: Different notions of Pareto optimality used in the legacy discussion.

23.3 MOGA Theory

The chapter frames the optimizer as a multi-objective genetic algorithm workflow. The focus is practical rather than theoretical: population management, mutation, crossover, and evaluation of the objective and constraint expressions on top of ordinary `OPAL` simulations.

23.4 Optimiser OPAL Commands

The optimization workflow is defined by:

- `DVAR` for design variables
- `OBJECTIVE` for the quantities to minimize
- `CONSTRAINT` for admissibility conditions
- `OPTIMIZE` for the optimization run itself

23.4.1 Basic Syntax

Design variables are introduced one by one:

```
d1: DVAR, VARIABLE="x1", LOWERBOUND=-1.0, UPPERBOUND=1.0;
d2: DVAR, VARIABLE="x2", LOWERBOUND=-1.0, UPPERBOUND=1.0;
d3: DVAR, VARIABLE="x3", LOWERBOUND=-1.0, UPPERBOUND=1.0;
```

Objectives are then defined as expressions to minimize:

```
obj1: OBJECTIVE, EXPR="-statVariableAt('energy', 1.0)";
obj2: OBJECTIVE, EXPR="statVariableAt('emit_x', 1.0)";
```

and constraints in the same expression language:

```
con1: CONSTRAINT, EXPR="statVariableAt('rms_x', 1.0) < 1.0";
con2: CONSTRAINT, EXPR="statVariableAt('numParticles', 1.0) > 1000";
```

Template files must contain placeholder variables such as `_x1_` for the corresponding `DVAR` entries.

23.4.2 `OPTIMIZE` Command

`OPTIMIZE` starts the optimization run.

Important attributes include:

Attribute	Meaning
`INPUT`	Path to the template or input file
`OUTPUT`	Base name for generated result files
`OUTDIR`	Output directory for generations and results
`OBJECTIVES`	List of objective definitions
`DVARS`	List of design variables
`CONSTRAINTS`	List of constraints
`INITIALPOPULATION`	Initial population size
`STARTPOPULATION`	Optional restart population in JSON format
`NUM_MASTERS`	Number of master ranks
`NUM_COWORKERS`	Number of worker ranks per individual

Attribute	Meaning
DUMP_DAT	Dump legacy plain-text generation format
DUMP_FREQ	Generation dump frequency
DUMP_OFFSPRING	Dump offspring instead of parents
NUM_IND_GEN	Individuals per generation
MAXGENERATIONS	Maximum number of generations
EPSILON	Hypervolume tolerance
EXPECTED_HYPERVOL	Reference hypervolume
HYPERVOLREFERENCE	Hypervolume reference point
CONV_HVOL_PROG	Hypervolume-based convergence threshold
ONE_PILOT_CONVERGE	Single-pilot convergence mode
SOL_SYNCH	Solution-exchange frequency
INITIAL_OPTIMIZATION	Speed up generation of the first population
BIRTH_CONTROL	Enforce strict population sizes
MUTATION_PROBABILITY	Mutation probability per individual
MUTATION	Mutation type
GENE_MUTATION_PROBABILITY	Mutation probability per gene
RECOMBINATION_PROBABILITY	Crossover probability
CROSSOVER	Crossover model
SIMBIN_CROSSOVER_NU	Simulated binary crossover parameter
SIMTMPDIR	Simulation scratch directory
TEMPLATEDIR	Template directory
FIELDMAPDIR	Field-map directory
DISTDIR	Distribution directory
RESTART_FILE	Optional H5 restart file
RESTART_STEP	Restart step inside the H5 restart file

23.4.3 DVAR Command

DVAR defines a design variable.

Attribute	Meaning
VARIABLE	Variable name used in the template
LOWERBOUND	Lower admissible value
UPPERBOUND	Upper admissible value

23.4.4 OBJECTIVE Command

OBJECTIVE defines one quantity to minimize.

Attribute	Meaning
`EXPR`	Objective expression

23.4.5 `CONSTRAINT` Command

`CONSTRAINT` defines one admissibility condition.

Attribute	Meaning
`EXPR`	Constraint expression

23.4.6 Available Expressions

The expression language supports both ordinary mathematical functions and data accessors for simulation output.

Standard mathematical functions include:

- `sqrt`
- `pow`
- `exp`
- `log`
- `ceil`
- `floor`
- `fabs`
- `fmod`
- `sin`
- `asin`
- `cos`
- `acos`
- `tan`
- `atan`
- `sq`

The legacy manual also documents output-access functions such as:

- `fromFile(file)`
- `sddsVariableAt(var, refpos, sdds_file)`
- `sddsVariableAt(var, refvar, refpos, sdds_file)`
- `statVariableAt(var, refpos)`
- `statVariableAt(var, refvar, refpos)`
- `sumErrSq(meas_file, var_name, sdds_file)`
- `radialPeak(file, turn)`
- `sumErrSqRadialPeak(meas_file, sim_file, begin, end)`
- `probVariableWithID(var, id, probe_file)`

- `septum(probe)`

These allow optimization directly against `.stat`, SDDS, probe, and measurement-derived quantities.

23.4.7 Example Input File

The original manual includes a full example based on the files 05-DL_QN3.in and tmpl/05-DL_QN3.tmpl. The example optimizes three quadrupole strengths through `DVAR` definitions, evaluates beam statistics as objectives, and runs `OPTIMIZE` with a small initial population and generation count.

The corresponding runtime pattern is:

```
OPTIMIZE, INPUT="tmpl/05-DL_QN3.tmpl", OBJECTIVES={...}, DVARS={...},
          INITIALPOPULATION=5, MAXGENERATIONS=3, NUM_IND_GEN=3,
          NUM_MASTERS=1, NUM_COWORKERS=1, TEMPLATEDIR="tmpl",
          FIELDMAPDIR="fieldmaps", OUTPUT="optLinac", OUTDIR="results";
```

23.4.8 Output

The optimizer writes generation data in JSON format and optionally in the legacy plain-text format. The Pareto-front data are written to `ParetoFront_.json`, while runtime logs such as `opt.trace.0`, `opt-progress.0`, and `pilot.trace.0` record job dispatch and validity.

OPALX

This optimization feature is not yet available in OPALX. The OPAL view is retained as a compatibility reference and a reminder of intended future functionality.

24 Sampler

OPALX

`SAMPLE` and `SAMPLING` are not registered by the current OPALX executable. The OPAL view preserves the complete legacy workflow as a compatibility reference and a reminder of intended future functionality.

OPAL

The sampler is a companion to the optimizer. Instead of evolving a multi-objective population, it generates samples of the design variables and runs the corresponding simulations. This is useful for surrogate-model training, validation studies, and uncertainty quantification.

24.1 Sampler OPAL Commands

The syntax is intentionally close to the optimizer syntax, especially for the shared `DVAR` definitions.

24.1.1 Basic Syntax

Design variables are still defined through `DVAR`. The sampler then specifies how each design variable is sampled and how the resulting combinations are evaluated.

24.1.2 `SAMPLE` Command

`SAMPLE` controls the overall sampling run and whether the per-variable sample sets are combined in raster or non-raster fashion.

Attribute	Meaning
`RASTER`	Choose raster or non-raster combination of the individual sample sets
`INPUT`	Path to the input or template file
`OUTPUT`	Base name for generated result files
`OUTDIR`	Directory where simulations and results are written
`DVARS`	List of design variables to sample

Attribute	Meaning
OBJECTIVES	Quantities to evaluate and store for each sample
SAMPLINGS	Sampling-method definitions
NUM_MASTERS	Number of master ranks
NUM_COWORKERS	Number of worker ranks per simulation
TEMPLATEDIR	Template directory
FIELDMAPDIR	Field-map directory
DISTDIR	Distribution directory
KEEP	File extensions that should not be deleted after evaluation
RESTART_FILE	Optional H5 restart file
RESTART_STEP	Restart step inside the H5 restart file
JSON_DUMP_FREQ	Frequency of appending finished samples to the JSON result

24.1.2.1 Raster versus non-raster mode

- RASTER=TRUE: evaluate every combination of the per-variable sample points
- RASTER=FALSE: evaluate aligned sequences, giving a total count equal to the minimum sample count across all variables

So with raster mode the total sample count is

$$N = N_1 \times N_2 \times \cdots \times N_n, \tag{24.1}$$

whereas with non-raster mode it is

$$N = \min(N_1, N_2, \ldots, N_n). \tag{24.2}$$

24.1.3 SAMPLING Command

Each SAMPLING definition describes how one design variable is sampled.

Attribute	Meaning
VARIABLE	Name of the corresponding design variable
TYPE	Sampling method
RANDOM	Random or sequential mode
SEED	Random seed; default 42
STEP	Step size for randomized sequences
FNAME	File containing sample values

Attribute	Meaning
N	Number of samples for this variable

24.1.4 Available Sampling Methods

The legacy manual documents:

Method	Meaning
FROMFILE	Read values from a named column in a file
UNIFORM	Uniform floating-point sampling between lower and upper bounds
UNIFORM_INT	Uniform integer sampling between lower and upper bounds
GAUSSIAN	Gaussian sampling, with bounds interpreted as +-5 sigma
LATIN_HYPERCUBE	Random Latin-hypercube sampling
RANDOM_SEQUENCE_UNIFORM_INT	Randomized integer sequence sampling
RANDOM_SEQUENCE_UNIFORM	Randomized floating-point sequence sampling

The method semantics depend on whether RANDOM is enabled:

- in sequential mode, the command generates a deterministic sequence
- in random mode, the same method produces randomized samples from the same bounds or file source

24.2 Example Input File

The original manual gives a complete example in Sample.in:

```
nstep: DVAR, VARIABLE="nstep", LOWERBOUND=10, UPPERBOUND=40;
MX:    DVAR, VARIABLE="MX",    LOWERBOUND=16, UPPERBOUND=32;

SM1: SAMPLING, VARIABLE="nstep", TYPE="FROMFILE", FNAME="samples.dat";
SM2: SAMPLING, VARIABLE="MX",    TYPE="UNIFORM_INT", SEED=122, N=6;

SAMPLE,
    RASTER        = FALSE,
    DVARS         = {nstep, MX},
    SAMPLINGS     = {SM1, SM2},
    INPUT         = "Ring.tmpl",
```

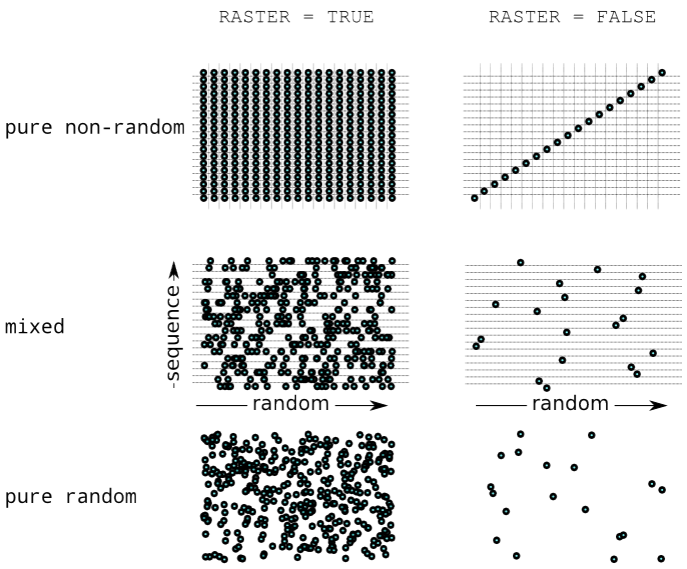


Figure 24.1: Difference between raster and non-raster sampling.

```
    OUTPUT         = "RingSample",
    OUTDIR         = "RingSample",
    TEMPLATEDIR    = "template",
    FIELDMAPDIR    = "Fieldmaps",
    NUM_MASTERS    = 1,
    NUM_COWORKERS = 1;
QUIT;
```

The accompanying samples.dat file provides named columns for the `FROMFILE`-based sampling source. The template file then uses placeholder variables such as `_nstep_` in the same way as the optimizer.

Part III

User Guide Appendix

25 Appendix A — Field Maps

OPALX

25.1 Current OPALX Field Maps

OPALX selects a field-map reader from the file header. The current factory can construct exactly four map types:

Header descriptor	OPALX reader	Use
`2DMagnetoStatic` (`2DMa...`)	`FM2DMagnetoStatic`	two-dimensional static magnetic map
`2DDynamic` (`2DDy...`)	`FM2DDynamic`	two-dimensional time-dependent electromagnetic map
`AstraDynamic` (`AstraD...`)	`Astra1DDynamic`	non-equidistant ASTRA dynamic on-axis map
`AstraMagnetoStatic` (`AstraM...`)	`Astra1DMagnetoStatic`	non-equidistant ASTRA static magnetic on-axis map

The descriptor is read from the beginning of the first non-comment record. Comments begin with `#`.

25.1.1 `AstraDynamic` and `AstraMagnetoStatic`

The ASTRA readers accept non-equidistant axial samples and construct an equidistant internal spline representation. `AstraDynamic` starts with its descriptor and Fourier coefficient count, followed by the frequency in `MHz` and `(z [m], Ez [MV/m])` records. `AstraMagnetoStatic` contains `(z [m], Bz [T])` records after its descriptor. Neither current reader provides the old `FAST` path.

25.1.2 `2DMagnetoStatic`

The header declares `XZ` or `ZX` ordering, followed by the two grid extents and spacing counts. Samples contain `(Bz, Br)` in `XZ` order or `(Br, Bz)` in `ZX` order. The reader uses bilinear interpolation and normalizes the on-axis longitudinal field to a peak magnitude of `1 T` unless normalization is disabled in the header.

25.1.3 2DDynamic

This RF format declares XZ or ZX ordering, a longitudinal grid, frequency in MHz, a radial grid, and four values per sample. In XZ order these are Ez, Er, an unused magnitude column, and Hphi; the first two columns are reversed in ZX order. Electric values are normalized to a peak on-axis longitudinal field of 1 MV/m unless normalization is disabled.

25.1.4 Recognized but unsupported headers

The header parser still recognizes the following legacy prefixes, but the current factory has no construction branch for them. They therefore fail when an element tries to load the map and must not be used in OPALX:

Recognized header	Parsed family
1DDy..., 1DMa..., 1DPr..., 1DEl...	legacy one-dimensional maps and profiles
2DEl...	two-dimensional electrostatic map
3DDy..., 3DMa..., 3DEl...	legacy three-dimensional maps
H5hut/HDF5 field blocks	three-dimensional magnetic or dynamic block map
AstraE...	ASTRA electrostatic map
1DPROFILE1-DEFAULT	built-in legacy profile marker

Recognition is not support: Fieldmap::getFieldmap() rejects all of these types. This distinction reflects OPALX source commit d1e762f15.

OPAL

25.2 Introduction

This appendix documents the field-map types accepted by OPAL-T. The legacy implementation supports several one-, two-, and three-dimensional field-map formats that originated from different external codes and accelerator applications.

The original manual groups them into three families:

1. 2D and 3D sampled field maps with interpolation on a grid
2. 1D on-axis profiles reconstructed off axis from a truncated Fourier series
3. Enge-function fringe-field models for selected bend magnets

In all cases, electric maps are normalized to a peak on-axis longitudinal field of 1 MV/m and magnetic maps to a peak on-axis longitudinal field of 1 T, unless the map type is explicitly documented as an exception.

25.3 Comments in Field Maps

Comments are introduced by `#` and extend to the end of the line. They are accepted:

- at the beginning of the file
- between data lines
- at the end of a complete data record

They must not split a record that is expected to stay on one line. A valid example is:

```
# This is a valid comment
1DMagnetoStatic 40 # This is also valid
-60.0 60.0 9999
# and this is valid too
0.0 2.0 199
```

A broken example is:

```
1DMagnetoStatic # invalid split record
40
-60.0 60.0 # invalid inline split # 9999
0.0 2.0 199
```

25.4 Normalization

ASCII field maps are normalized with the maximum absolute value of the longitudinal on-axis field. The documented exception is `1DProfile1`. This normalization can be disabled by appending `FALSE` to the first line of the field map.

25.5 Field-Map Warnings and Errors

If `OPAL-T` encounters a parsing error, it disables the corresponding element, emits a warning, and continues the simulation. The main warning categories are:

- too few lines compared with the header counts
- too many lines compared with the header counts
- wrong number or type of values on a given line
- file not found
- unrecognized field-map descriptor
- too few Fourier coefficients for a 1D field reconstruction

For one-dimensional maps the reconstruction quality is checked with the legacy criteria

$$\frac{\sum_{i=1}^{N}(F_{z,i}-\tilde{F}_{z,i})^2}{\sum_{i=1}^{N}F_{z,i}^2}\leq 10^{-2}, \tag{25.1}$$

and

$$\frac{\max_i|F_{z,i}-\tilde{F}_{z,i}|}{\max_i|F_{z,i}|}\leq 10^{-2}, \tag{25.2}$$

where `F_z` is the sampled field from the file and `\tilde F_z` is the field reconstructed after low-pass filtering.

25.6 Types and Format

The legacy appendix lists both implemented and planned descriptors.

25.6.1 Implemented families

Type	Meaning
`1DMagnetoStatic`	one-dimensional static magnetic field
`AstraMagnetoStatic`	ASTRA magnetic field format with non-equidistant sampling
`1DDynamic`	one-dimensional time-dependent RF field
`AstraDynamic`	ASTRA dynamic field format
`1DProfile1`	Enge-like bend fringe profile
`2DElectroStatic`	two-dimensional electrostatic field
`2DMagnetoStatic`	two-dimensional static magnetic field
`2DDynamic`	two-dimensional dynamic field
`3DMagnetoStatic`	three-dimensional static magnetic field
`3DMagnetoStatic_Extended`	mid-plane magnetic field extended to 3D
`3DDynamic`	three-dimensional dynamic field

25.6.2 Planned but not implemented in the historical appendix

Type	Note
`1DElectroStatic`	not implemented; old workaround was `1DDynamic` with very low frequency
`3DElectroStatic`	not implemented

25.7 Field-Map Orientation

Two- and three-dimensional maps carry an orientation string in the header.

For 2D maps:

- `XZ`: primary direction in `z`, secondary in `r`
- `ZX`: primary direction in `r`, secondary in `z`

For 3D maps:

- `XYZ`: primary direction in `z`, secondary in `y`, tertiary in `x`

The primary index increases fastest and the tertiary index slowest. Accordingly, the first field component column corresponds to the primary axis-order interpretation. For 2D dynamic maps in `XZ` orientation the four data columns are:

- `Ez`
- `Er`
- `|E|` (unused by the legacy code)
- `Hphi`

In the alternative 2D orientation, the first two columns are interchanged while the third and fourth are unchanged.

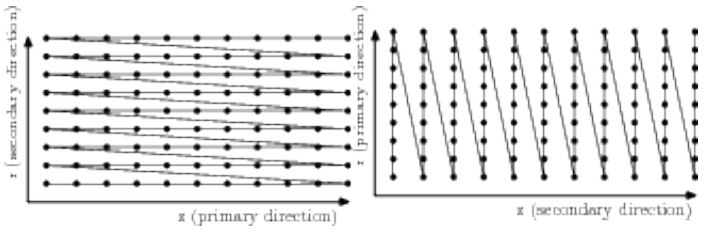


Figure 25.1: Axis-ordering conventions for the legacy field-map orientation rules.

25.8 `FAST` Attribute for 1D Field Maps

For some 1D field maps, the boolean attribute `FAST` can be used to speed up field evaluation. When `FAST = TRUE`, `OPAL-T` generates an internal 2D field map and then uses bilinear interpolation rather than the slower Fourier-series technique.

This is faster, but the legacy appendix warns that it can introduce numerical noise if the generated 2D grid is too coarse.

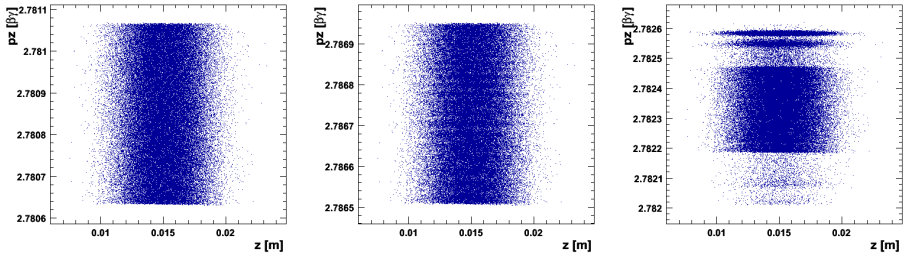


Figure 25.2: Example of field noise caused by insufficient one-dimensional reconstruction.

25.9 1DMagnetoStatic

A `1DMagnetoStatic` map stores one-dimensional static magnetic data on axis. An example header from the appendix is:

```
1DMagnetoStatic 40
-60.0 60.0 9999
0.0 2.0 199
```

This describes a map with `10000` longitudinal points (`9999` spacings) from `-60 cm` to `60 cm` relative to `ELEMEDGE`. The field is normalized so that `max(|B_on axis|) = 1 T`. The code computes `Nz/2` complex Fourier coefficients and retains only `N_Fourier` during tracking.

If `FAST = TRUE`, the third line defines the radial range of an internally generated 2D map. In the example, that is `0 cm` to `2 cm` with `200` radial values.

Line	Contents
1	`1DMagnetoStatic N_Fourier [TRUE|FALSE]`
2	`z_start z_end Nz` in cm
3	`r_start r_end Nr` in cm
remaining lines	`Bz` samples in tesla

25.10 AstraMagnetoStatic

This format accepts non-equidistant longitudinal samples in meters. The first column is `z`, the second is the on-axis longitudinal magnetic field. `OPAL-T` uses cubic-spline interpolation to resample the data to an equidistant mesh, then computes Fourier coefficients from that internal mesh.

Line	Contents
1	`AstraMagnetoStatic N_Fourier [TRUE|FALSE]`

Line	Contents
remaining lines	`z_i [m]`, `Bz_i [T]`

The appendix explicitly notes that `AstraMagnetoStatic` does not provide a `FAST` path.

25.11 `1DDynamic`

A `1DDynamic` map is the RF analogue of `1DMagnetoStatic`. An example header is:

```
1DDynamic 40
-3.0 57.0 4999
1498.953425154
0.0 2.0 199
```

The field is sampled on axis, normalized so that `max(|E_on axis|) = 1 MV/m`, and evaluated at runtime with the time factor `cos(omega t + phi)`.

Line	Contents
1	`1DDynamic N_Fourier [TRUE|FALSE]`
2	`z_start z_end Nz` in cm
3	frequency in MHz
4	`r_start r_end Nr` in cm for the optional `FAST` path
remaining lines	`Ez` samples in MV/m

25.12 `AstraDynamic`

`AstraDynamic` accepts a frequency line followed by non-equidistant `z` and `Ez` samples in meters and MV/m. `OPAL-T` spline-resamples the data to an internal equidistant mesh before the Fourier reconstruction.

Line	Contents
1	`AstraDynamic N_Fourier [TRUE|FALSE]`
2	frequency in MHz
remaining lines	`z_i [m]`, `Ez_i [MV/m]`

The appendix again notes that there is no `FAST` variant.

25.13 `1DProfile1`

`1DProfile1` is the special bend-fringe profile format. It uses Enge functions for the entrance and exit fringe fields [6],

$$F(z) = \frac{1}{1 + e^{\sum_{n=0}^{N_{\text{order}}} c_n (z/D)^n}}, \tag{25.3}$$

where `D` is the full gap of the magnet and `c_n` are fitted coefficients. Unlike the other ASCII formats, `1DProfile1` is not normalized to unit peak field.

The appendix warns that the polynomial degree should be odd and the highest coefficient should be positive, otherwise the fitted profile may turn upward again at large distance.

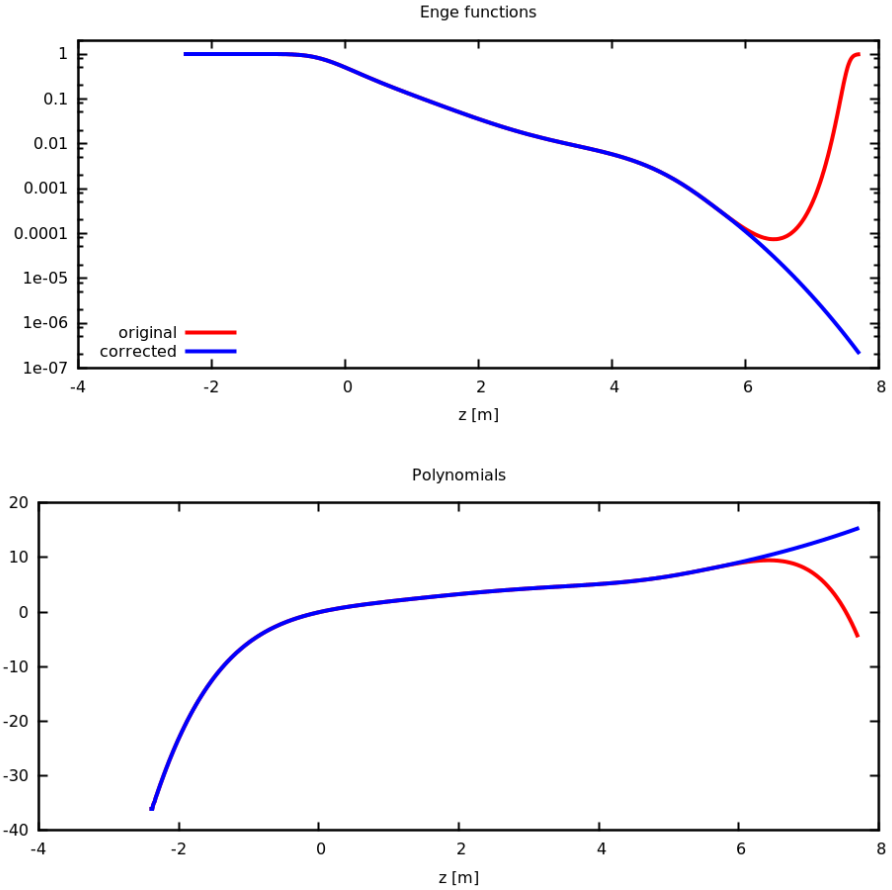


Figure 25.3: Polynomial Enge-style profile used for `1DProfile1`.

The generic file layout is:

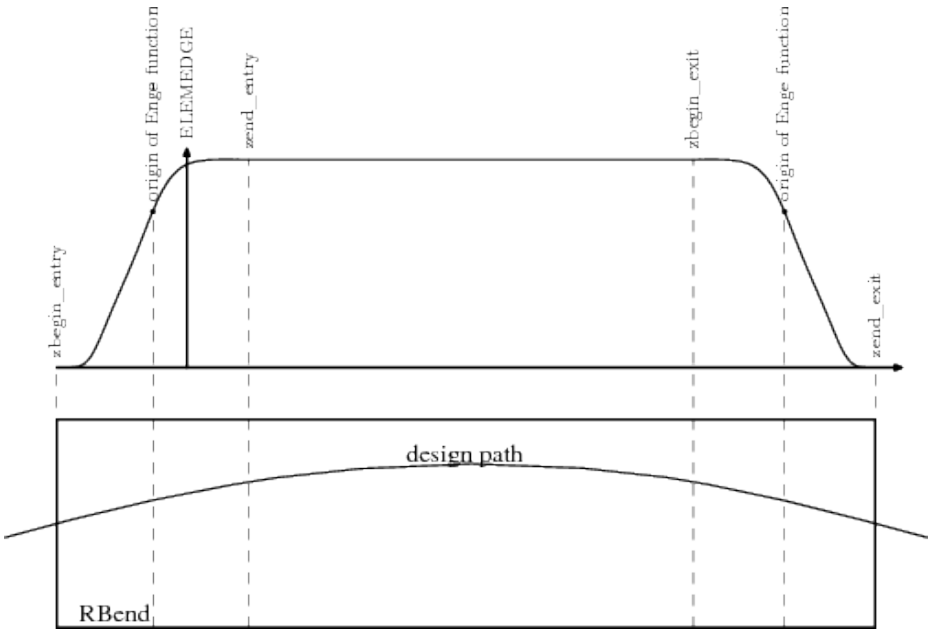


Figure 25.4: Profile shape for `1DProfile1`.

Line	Contents
1	`1DProfile1 N_Enge_Entrance N_Enge_Exit Gap`
2	three entrance parameters plus a placeholder
3	three exit parameters plus a placeholder
following lines	entrance coefficients `c_0 ... c_N` then exit coefficients

25.13.1 `1DProfile1` Type 1

Type 1 stores the magnet length implicitly in the difference between the second exit and entrance parameters. The appendix defines

$$\begin{aligned} \text{Entrance Parameter 1} &= \text{Entrance Parameter 2} - \Delta_1 \\ \text{Entrance Parameter 3} &= \text{Entrance Parameter 2} + \Delta_2 \\ \text{Exit Parameter 2} &= L - \text{Entrance Parameter 2} \\ \text{Exit Parameter 1} &= \text{Exit Parameter 2} - \Delta_3 \\ \text{Exit Parameter 3} &= \text{Exit Parameter 2} + \Delta_4 \end{aligned} \tag{25.4}$$

and recovers internally

$$
\begin{aligned}
L &= \text{Exit Parameter 2} - \text{Entrance Parameter 2} \\
\Delta_1 &= \text{Entrance Parameter 2} - \text{Entrance Parameter 1} \\
\Delta_2 &= \text{Entrance Parameter 3} - \text{Entrance Parameter 2} \\
\Delta_3 &= \text{Exit Parameter 2} - \text{Exit Parameter 1} \\
\Delta_4 &= \text{Exit Parameter 3} - \text{Exit Parameter 2.}
\end{aligned}
\tag{25.5}
$$

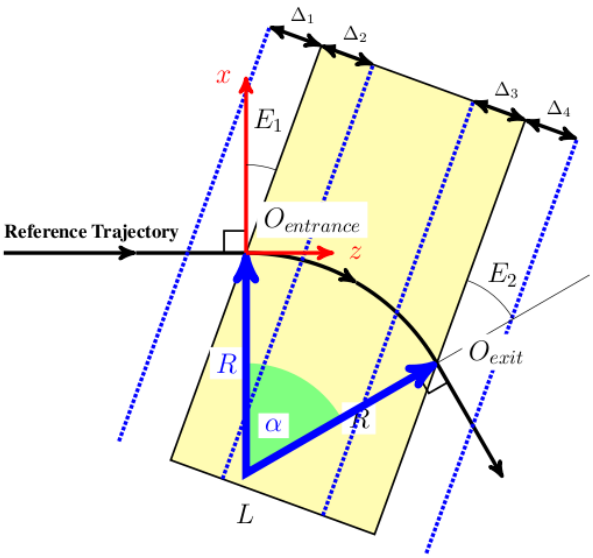


Figure 25.5: `RBEND` profile geometry for `1DProfile1` type 1.

25.13.2 `1DProfile1` Type 2

Type 2 removes the magnet-length information from the field map and uses the beamline element attribute `L` instead. This allows the Enge origins to move relative to the physical edges.

$$
\begin{aligned}
&\text{Entrance Parameter 2} = \text{perpendicular distance of the entrance Enge origin from the entrance edge,} \\
&\text{Exit Parameter 2} = \text{perpendicular distance of the exit Enge origin from the exit edge.}
\end{aligned}
\tag{25.6}
$$

The remaining deltas are still recovered from the parameter differences. The two main advantages noted in the appendix are:

- the Enge origins can be adjusted to model the fringe more accurately
- one map can be reused for magnets with identical fringe fields but different lengths

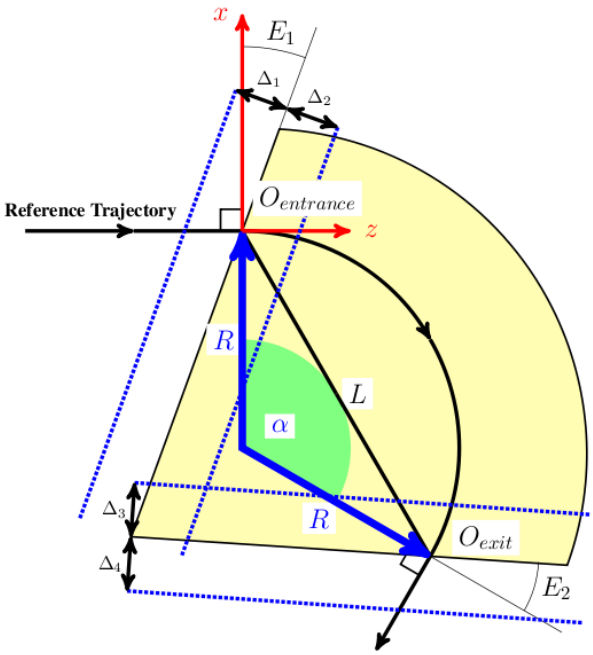


Figure 25.6: `SBEND` profile geometry for `1DProfile1` type 1.

25.14 Two-Dimensional Families

25.14.1 `2DElectroStatic`

Example header:

```
2DElectroStatic XZ
-3.0 51.0 4999
0.0 2.0 199
0.00000e+00 0.00000e+00
4.36222e-06 0.00000e+00
...
```

This format describes a sampled electrostatic field with bilinear interpolation. In the legacy workflow these maps were often produced with Poisson Superfish [7]. In the source example the map spans `5000` longitudinal points and `200` radial points. The field is non-negligible from `-3.0 cm` to `51.0 cm` relative to `ELEMEDGE`, and the radial support is `0.0 cm` to `2.0 cm`. In `XZ` orientation the first column is `Ez` and the second is `Er`; for `ZX` orientation the columns are exchanged.

Line	Contents
1	`2DElectroStatic Orientation [TRUE|FALSE]`

Line	Contents
2	fastest-changing axis extent and spacing count
3	slowest-changing axis extent and spacing count
remaining lines	`(Ez, Er)` for `XZ`, `(Er, Ez)` for `ZX`

25.14.2 `2DMagnetoStatic`

Example header:

```
2DMagnetoStatic ZX
0.0 2.0 199
-3.0 51.0 4999
0.00000e+00 0.00000e+00
0.00000e+00 4.36222e-06
...
```

This is the magnetostatic analogue of `2DElectroStatic`. The field is normalized so that `max(|Bz on axis|) = 1 T`. In the source example the orientation is `ZX`, so the fastest-changing index is radial and the stored pair is `(Br, Bz)` rather than `(Bz, Br)`.

Line	Contents
1	`2DMagnetoStatic Orientation [TRUE|FALSE]`
2	fastest-changing axis extent and spacing count
3	slowest-changing axis extent and spacing count
remaining lines	`(Bz, Br)` for `XZ`, `(Br, Bz)` for `ZX`

25.14.3 `2DDynamic`

Example header:

```
2DDynamic XZ
-3.0 51.0 4121
1498.953425154
0.0 1.0 75
0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00
...
```

This is the two-dimensional RF format. It adds a frequency line and stores four values per grid point:

- `Ez`
- `Er`

- `|E|` (present but unused by `OPAL-T`)
- `Hphi`

The first two columns follow the `XZ` / `ZX` orientation rule, while the third and fourth do not.

Line	Contents
1	`2DDynamic Orientation [TRUE|FALSE]`
2	fastest-changing axis extent and spacing count
3	frequency in MHz
4	slowest-changing axis extent and spacing count
remaining lines	`Ez/Er`, `Er/Ez`, `|E|`, `Hphi`

25.15 Three-Dimensional Families

25.15.1 `3DMagnetoStatic`

Example header:

```
3DMagnetoStatic
-1.5 1.5 227
-1.0 1.0 151
-3.0 51.0 4121
0.00e+00 0.00e+00 0.00e+00
...
```

The plain 3D magnetostatic format stores full sampled vector-field data on a Cartesian grid and uses trilinear interpolation. The legacy ordering is `z` fastest, then `y`, then `x`. The data columns are `Bx`, `By`, and `Bz`.

Line	Contents
1	`3DMagnetoStatic [TRUE|FALSE]`
2	`x_start x_end Nx` in cm
3	`y_start y_end Ny` in cm
4	`z_start z_end Nz` in cm
remaining lines	`Bx By Bz`

25.15.2 `3DMagnetoStatic_Extended`

Example header:

```
3DMagnetoStatic_Extended
-9.9254 9.9254 133
-2.0 1.0 15
-22.425 47.425 465
-8.10970000e-05
...
```

This format stores only a mid-plane magnetic map and reconstructs the 3D field using Maxwell consistency and a perfect-magnetic-conductor mid-plane symmetry. The stored quantity is the perpendicular field component on the mid-plane, and the upper half-plane is mirrored to negative `y`.

Line	Contents
1	`3DMagnetoStatic_Extended [TRUE|FALSE]`
2	`x_start x_end Nx` in cm
3	`y_start y_end Ny` in cm
4	`z_start z_end Nz` in cm
remaining lines	stored mid-plane `By` values

25.15.3 `3DDynamic`

Example header:

```
3DDynamic
1498.9534
-1.5 1.5 227
-1.0 1.0 151
-3.0 51.0 4121
0.00e+00 0.00e+00 0.00e+00 0.00e+00 0.00e+00 0.00e+00
...
```

The dynamic 3D format adds a frequency specification and then stores six field components per sample:

- `Ex`
- `Ey`
- `Ez`
- `Hx`
- `Hy`
- `Hz`

The grid ordering is again `z` fastest, then `y`, then `x`.

Line	Contents
1	`3DDynamic [TRUE|FALSE]`
2	frequency in MHz
3	`x_start x_end Nx` in cm
4	`y_start y_end Ny` in cm
5	`z_start z_end Nz` in cm
remaining lines	`Ex Ey Ez Hx Hy Hz`

26 Appendix B — Input Language Syntax

This appendix reproduces the compact grammar shared by OPALX and OPAL input language. Words in italic in the legacy appendix denote syntactic entities, while monospaced tokens must be entered literally.

26.1 Statements

Entity	Form
comment	`// ...` or `/* ... */`
identifier	`[a-zA-Z][a-zA-Z0-9-]`
integer	`[0-9]+`
string	single-quoted or double-quoted text
command	`_keyword_ _attribute-list_` or `_label_ : _keyword_ _attribute-list_`
attribute	`_attribute-name_`, `_attribute-name_ = _attribute-value_`, or `_attribute-name_ := _attribute-value_`

The distinction between `=` and `:=` is semantic as well as syntactic:

- `=` evaluates the expression immediately.
- `:=` stores the expression for later evaluation.

An attribute value may be any of the parser categories used in the original appendix:

- string expression
- logical expression
- real expression
- array expression
- constraint
- variable reference
- place
- range
- token list
- token-list array
- regular expression

26.2 Real Expressions

Entity	Form
real-expression	`_real-term_`, unary `+/-` `_real-term_`, or recursive `+` / `-` combinations
real-term	`_real-factor_` or recursive `*` / `/` combinations
real-factor	`_real-primary_` or recursive exponentiation `_real-factor_ ^ _real-primary_`
real-primary	real literal, symbolic constant, `#`, real name, array lookup, object attribute lookup, table reference, function call, or parenthesized expression

Built-in scalar functions listed by the legacy appendix are:

- `RANF`, `GAUSS`
- `ABS`, `TRUNC`, `ROUND`, `FLOOR`, `CEIL`, `SIGN`
- `SQRT`, `LOG`, `EXP`
- `SIN`, `COS`, `TAN`, `ASIN`, `ACOS`, `ATAN`, `ATAN2`
- `MAX`, `MIN`, `MOD`, `POW`

Functions of arrays are:

- `VMIN`
- `VMAX`
- `VRMS`
- `VABSMAX`

The grammar also allows:

- `_array_[ _index_ ]`
- `_object-name_ -> _real-attribute_`
- `_object-name_ -> _array-attribute_[ _index_ ]`
- `_table-reference_`

26.3 Real Variables and Constants

Entity	Form
real-prefix	empty, `REAL`, `CONST`, or `REAL CONST`
real-definition	`_real-prefix_ _real-name_ = _real-expression_` or `:=` variant

Entity	Form
symbolic-constant	PI, TWOPI, DEGRAD, RADDEG, E, EMASS, PMASS, HMMASS, UMASS, CMASS, MMASS, DMASS, XEMASS, CLIGHT

The original appendix also lists _real-name_, _object-name_, _table-name_, and _column-name_ as identifiers.

26.4 Logical Expressions and Variables

Logical expressions are built from relations combined with && and ||. The relation grammar allows:

- _logical-name_
- TRUE
- FALSE
- _real-expression_ _relation-operator_ _real-expression_

The relation operators are:

- ==
- !=
- <
- >
- >=
- <=

Logical variables use the prefixes:

- BOOL
- BOOL CONST

and the same = versus := distinction as for reals.

26.5 String Expressions and Constants

String expressions may be:

- a literal string
- an identifier interpreted as a string
- a concatenation _string-expression_ & _string_

String definitions use the prefix STRING:

- `STRING name = expression`
- `STRING name := expression`

26.6 Real Array Expressions and Definitions

The legacy appendix distinguishes scalar and array grammars explicitly. Array expressions mirror scalar arithmetic:

- unary `+` and `-`
- binary `+`, `-`, `*`, `/`
- exponentiation
- parentheses

The array primaries are:

- `{ _array-literal_ }`
- `_array-reference_`
- `_real-function_( _array-expression_ )`
- `( _array-expression_ )`

The array literal grammar is recursive:

- `_array-literal_ : _real-expression_`
- `_array-literal_ : _array-literal_ , _real-expression_`

Array references may be:

- `_array-name_`
- `_object-name_ -> _array-attribute_`

Real array definitions use:

- `_array-prefix_ : REAL VECTOR`
- `_array-definition_ : _array-prefix_ _array-name_ = _array-expression_`
- `_array-definition_ : _array-prefix_ _array-name_ := _array-expression_`

26.7 Constraints

Constraints are first-class attribute values in the grammar. The appendix gives exactly:

- `_constraint_ : _array-expression_ _constraint-operator_ _array-expression_`

with allowed operators:

- `==`
- `<`

- `>`

This is the form used by optimization-related commands when comparing vector or sampled quantities.

26.8 Variable References

The variable-reference grammar in the appendix is intentionally small:

- `_variable-reference_ : _real-name_`
- `_variable-reference_ : _object-name_ -> _attribute-name_`

This is the basic hook that lets later statements reuse parser-visible values and object attributes.

26.9 Token Lists and Regular Expressions

The final grammar sections define parser tokens that are not scalar numbers.

Token lists:

- `_token-list_ : _anything-except-comma_`
- `_token-list-array_ : _token-list_`
- `_token-list-array_ : _token-list-array_ , _token-list_`

Regular expressions:

- `_regular-expression_ : "` *UNIX-regular-expression* `"`

These forms appear mostly in higher-level command attributes and parser-side matching constructs rather than in beam-dynamics expressions.

27 Appendix C — OPAL-MADX Conversion Guide

27.1 Purpose

This appendix summarizes the unit and convention changes needed when translating between `MADX` and `OPAL`.

The legacy discussion is built around the transverse beam sigma matrix,

$$\sigma_{\text{beam}} = \begin{pmatrix} \sigma_x & \sigma_{xp_x} \\ \sigma_{xp_x} & \sigma_{p_x} \end{pmatrix} = \epsilon \begin{pmatrix} \beta & -\alpha \\ -\alpha & \gamma \end{pmatrix}, \tag{27.1}$$

with the correlation convention

$$\sigma_{xp_x} = \delta \sqrt{\sigma_x \sigma_{p_x}}. \tag{27.2}$$

The relativistic quantities used throughout the appendix are

$$\gamma = \frac{E_{\text{kin}} + m_p}{m_p}, \qquad \beta = \sqrt{1 - \frac{1}{\gamma^2}}, \qquad \beta\gamma = \frac{\beta}{\sqrt{1-\beta^2}}, \tag{27.3}$$

and the magnetic rigidity is

$$B\rho = \frac{(\beta\gamma)\, m_p\, 10^9}{c}\ [\mathrm{T\,m}], \tag{27.4}$$

with `m_p = 0.939277 GeV` and `c = 299792458 m/s`.

27.2 `MADX` to `OPAL` Output Conventions

The legacy appendix lists the following quantity-by-quantity conversion rules.

Quantity	MADX	OPAL output	Conversion
transverse momentum	\bar p_x [rad]	\bar p_x [\beta\gamma]	\bar p_x[\beta\gamma] = \bar p_x[rad] \,(\beta\gamma)
correlation	\delta	\delta	\delta = \sigma_{x p_x}/(\bar p_x\bar x)
emittance	\epsilon_x [m\,rad]	\epsilon_x [m\,\beta\gamma]	\epsilon_x = \sqrt{\sigma_x\sigma_{p_x} - \sigma_{x p_x}^2}
Twiss \alpha	dimensionless	dimensionless	\alpha = -\sigma_{x p_x}/\epsilon_x
Twiss \beta_T	m/rad	m/(\beta\gamma)	\beta_T = \sigma_x / \epsilon_x
Twiss \gamma_T	rad/m	(\beta\gamma)/m	\gamma_T = \sigma_{p_x} / \epsilon_x
quadrupole strength	k_1 [m^{-2}]	k_1 [T/m]	k_1[T/m] = k_1[m^{-2}]\,B\rho

The same rules apply plane by plane in both transverse directions.

27.3 Equivalent Explicit Formulas

The original appendix writes the most frequently used conversions explicitly:

$$\bar{p}_x = \sqrt{\sigma_{p_x}}, \qquad \bar{x} = \sqrt{\sigma_x}, \tag{27.5}$$

$$\epsilon_x = \sqrt{(\bar{p}_x\bar{x})^2 - (\delta\,\bar{x}\bar{p}_x)^2} = \sqrt{\sigma_x\sigma_{p_x} - \sigma_{xp_x}^2}, \tag{27.6}$$

$$\alpha = -\frac{\delta\,\bar{x}\bar{p}_x}{\epsilon_x}, \qquad \beta_T = \frac{\bar{x}^2}{\epsilon_x}, \qquad \gamma_T = \frac{\bar{p}_x^2}{\epsilon_x}. \tag{27.7}$$

These are the formulas to use when reconstructing an OPAL beam from MADX Twiss or sigma-matrix output.

27.4 MADX Element Positions to OPAL Input

The appendix highlights one convention difference that shows up repeatedly:

- `MADX at :=` gives the element center.
- `OPAL ELEMEDGE` expects the element entrance.

So the conversion is

$$\text{ELEMEDGE} = \text{element center} - \frac{1}{2}\text{element length}. \tag{27.8}$$

Quantity	MADX	OPAL input
element position conversion	`at :=` center of element center minus half length	`ELEMEDGE` at start of element entrance position

27.5 OPAL Output to OPAL Input Energy-Style Conversion

The appendix also gives the conversion from transverse momentum in `\beta\gamma` form to an energy-style input quantity in eV:

$$p_x[\text{eV}] = m_p\, 10^9 \left(\sqrt{\bar{p}_x^2 + 1} - 1\right). \tag{27.9}$$

This is the relation to use when an `OPAL` output quantity must be mapped back into an input-style energy convention.

27.6 Practical Use

This guide is most useful when:

- reconstructing an `OPAL` beam from `MADX` Twiss or sigma-matrix output
- translating focusing strengths and element positions
- reconciling unit conventions in transverse and longitudinal phase-space descriptions

28 Appendix D — Auto-phasing Algorithm

OPAL

28.1 Standing Wave Cavity

For external RF fields in `OPAL-T`, the instantaneous field is the interpolated spatial field multiplied by `cos(omega t + phi)`. The energy gain of a particle is therefore

$$\Delta E(\varphi, r) = qV_0 \int_{z_{\text{begin}}}^{z_{\text{end}}} \cos(\omega t(z, \varphi) + \varphi)\, E_z(z, r)\, dz. \tag{28.1}$$

To maximize the gain, one differentiates with respect to the lag `\varphi` and sets the derivative to zero. This yields the lag rule

$$\tan(\varphi) = -\Gamma_1/\Gamma_2. \tag{28.2}$$

The original appendix defines

$$\Gamma_1 = \sum_{i=1}^{N-1} \left(1 + \omega \frac{\partial t}{\partial \varphi}\right) \int_{z_{i-1}}^{z_i} \sin\left(\omega \left(t_{i-1} + \Delta t_i \frac{z - z_{i-1}}{\Delta z_i}\right)\right) \left(E_{z,i-1} + \Delta E_{z,i} \frac{z - z_{i-1}}{\Delta z_i}\right) dz, \tag{28.3}$$

and

$$\Gamma_2 = \sum_{i=1}^{N-1} \left(1 + \omega \frac{\partial t}{\partial \varphi}\right) \int_{z_{i-1}}^{z_i} \cos\left(\omega \left(t_{i-1} + \Delta t_i \frac{z - z_{i-1}}{\Delta z_i}\right)\right) \left(E_{z,i-1} + \Delta E_{z,i} \frac{z - z_{i-1}}{\Delta z_i}\right) dz. \tag{28.4}$$

Assuming piecewise-linear field and time-of-flight models between samples, the integrals can be evaluated analytically in terms of four coefficients:

$$\Gamma_{11,i} = -\frac{\cos(\omega t_i) - \cos(\omega t_{i-1})}{\omega \Delta t_i}, \qquad \Gamma_{12,i} = \frac{-\omega \Delta t_i \cos(\omega t_i) + \sin(\omega t_i) - \sin(\omega t_{i-1})}{\omega^2 (\Delta t_i)^2}, \tag{28.5}$$

$$\Gamma_{21,i} = \frac{\sin(\omega t_i) - \sin(\omega t_{i-1})}{\omega \Delta t_i}, \qquad \Gamma_{22,i} = \frac{\omega \Delta t_i \sin(\omega t_i) + \cos(\omega t_i) - \cos(\omega t_{i-1})}{\omega^2 (\Delta t_i)^2}. \tag{28.6}$$

With these, the appendix rewrites the accumulated sums in a form suitable for an iterative implementation.

28.1.1 Implemented Iterative Idea

The legacy algorithm starts from a guessed kinetic-energy evolution,

```
K[i] = K[i-1] + (z[i] - z[0]) * q * V;
b[i] = sqrt(1. - 1. / ((K[i] - K[i-1]) / (2.*m*c^2) + 1)^2);
t[i] = t[0] + (z[i] - z[0]) / (c * b[i])
```

which assumes a roughly linear energy increase proportional to the maximum voltage. From that provisional `t(z)` model, `OPAL` computes the lag from Equation 28.2, then updates the kinetic energy using the analytic segment formulas,

$$K_i = K_{i-1} + q\,\Delta z_i \Big[\cos\varphi\,(E_{z,i-1}(\Gamma_{21,i} - \Gamma_{22,i}) + E_{z,i}\Gamma_{22,i}) - \sin\varphi\,(E_{z,i-1}(\Gamma_{11,i} - \Gamma_{12,i}) + E_{z,i}\Gamma_{12,i})\Big], \tag{28.7}$$

rebuilds the time-of-flight model, and iterates until `\varphi` converges.

28.2 Traveling Wave Structure

Traveling-wave auto-phasing is slightly more complicated because the field is composed of:

- an entry fringe field
- two standing-wave core contributions with phase offsets
- an exit fringe field

The appendix uses the field map `FINLB02-RAC.T7` as the representative traveling-wave example.

The corresponding energy gain is written as the sum of four integrals over the entry fringe, two core components, and exit fringe. In the notation of the appendix, the two core contributions are shifted by one cell length `s` and by phase offsets `\varphi_{c1}` and `\varphi_{c2}`.

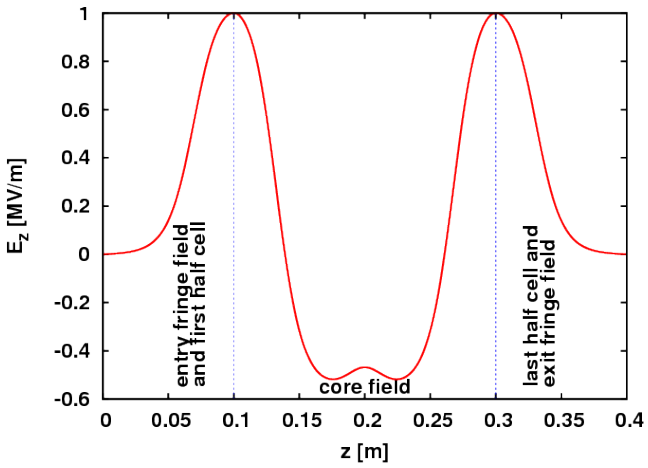


Figure 28.1: Cropped `1DDynamic` field map used in the traveling-wave auto-phasing discussion.

28.2.1 Example

The legacy example is:

```
FINLB02_RAC: TravelingWave, L=2.80, VOLT=14.750*30/31,
             NUMCELLS=40, FMAPFN="FINLB02-RAC.T7",
             ELEMEDGE=2.67066, MODE=1/3,
             FREQ=1498.956, LAG=FINLB02_RAC_lag;
```

From this, the appendix derives

$$V_{\text{core}} = \frac{V_0}{\sin(2\pi/3)} = \frac{2V_0}{\sqrt{3}}, \qquad \varphi_{c1} = \frac{\pi}{6}, \qquad \varphi_{c2} = \frac{\pi}{2}, \tag{28.8}$$

and an exit-fringe phase offset determined by `NUMCELLS` and `MODE`.

28.2.2 Alternative Approach for Traveling-Wave Structures

In the ultra-relativistic limit, where `beta` changes only weakly along the structure, the time-of-flight can be approximated linearly,

$$t(z,\varphi) \approx \frac{z}{\beta c} + t_0. \tag{28.9}$$

For the traveling-wave example, the appendix then rewrites the derivative condition in terms of periodic core and fringe fields over a period `3 s`, and shows how the expression simplifies because `\omega/(\beta c) \approx 10\pi`. The final reduced condition is expressed as a convolution,

$$0 \equiv \int_0^{3s} g(\xi - z)\,(G(z) - 26H(z))\,dz = \mathcal{F}^{-1}\big(\mathcal{F}(g)\,(\mathcal{F}(G) - 26\mathcal{F}(H))\big)\,, \tag{28.10}$$

with the phase relation

$$-\frac{\omega}{\beta c}\,\xi = \varphi. \tag{28.11}$$

The appendix also uses the trigonometric identity

$$\sin(a + \pi + \pi/6) + \sin(a + \pi - \pi/6) = -\sqrt{3}\,\sin(a), \tag{28.12}$$

which is the final algebraic step that collapses the two core terms into the reduced convolution form.

OPALX

28.3 Current OPALX Implementation

`OPALX` already contains an implemented cavity autophasing path. The runtime entry points are:

- `OPTION, AUTOPHASE = n`
- `ParallelTracker::autophaseCavities()`
- `OrbitThreader::autophaseCavities()`
- `Algorithms/CavityAutophaser`

The global option `AUTOPHASE` is interpreted as the number of refinements used in the phase search. If it is zero, no autophasing is performed. If it is positive, `OPALX` scans active `RFCAVITY` and `TRAVELINGWAVE` elements and chooses the phase that maximizes the reference-particle exit energy.

28.3.1 Scope and Control

The current implementation applies to:

- `RFCAVITY`
- `TRAVELINGWAVE`

The per-element veto flag is:

- `APVETO`

If `APVETO` is set, the element is skipped by the automatic search and its stored phase is used as-is.

28.3.2 Search Strategy

The implementation uses a two-stage numerical procedure:

1. Build an initial phase estimate from the cavity model itself.
2. Refine that estimate by explicitly tracking the reference particle through the cavity and maximizing the final kinetic energy.

The implementation object is `CavityAutophaser`. Its public entry point is

```
double getPhaseAtMaxEnergy(const Vector_t<double, 3>& R,
                           const Vector_t<double, 3>& P,
                           double t,
                           double dt);
```

Internally it keeps the cavity-local initial reference state and then:

- calls `guessCavityPhase(...)`
- calls `optimizeCavityPhase(...)`
- writes the chosen phase back to the element

28.3.3 Initial Estimate for `RFCAVITY`

For standing-wave cavities, the first estimate is delegated to `RFCavity::getAutoPhaseEstimate(...)`. That routine samples the on-axis field map, builds a spline, constructs iterative time-of-flight and energy histories, and evaluates an `A/B` phase condition equivalent in structure to the legacy appendix derivation.

So the OPALX standing-wave path is directly connected to the analytic appendix logic, but implemented numerically on the actual on-axis sampled field.

28.3.4 Initial Estimate for `TRAVELINGWAVE`

For traveling-wave structures, the first estimate is delegated to `TravelingWave::getAutoPhaseEstimate(...`. That implementation explicitly splits the field into:

- entry field
- two phase-shifted core contributions
- exit field

and iterates on the same energy-maximization idea while respecting the traveling-wave phase offsets already stored in the object.

28.3.5 Refinement by Direct Tracking

After the initial guess, `CavityAutophaser::optimizeCavityPhase(...)` performs a local energy scan around the current phase. The search uses:

- an initial step `dphi = pi / 360`
- one-sided bracketing to find the uphill direction
- `AUTOPHASE` refinement rounds, each halving `dphi`

For each trial phase, the code evaluates the actual reference-particle energy by calling

```
rfc->trackOnAxisParticle(...)
```

through `CavityAutophaser::track(...)`. The final selected phase is therefore the one that maximizes the tracked on-axis reference-particle kinetic energy, not just the analytic estimate.

28.3.6 Special Cases

The current implementation handles two important special cases.

1. DC-gun-like structures If the RF frequency is effectively at or below `2*pi`, the code treats the structure as a DC gun and selects `0` or `pi` depending on the sign of the accelerating field and particle charge.
2. Design-energy-driven amplitude adjustment If a cavity has a positive design energy, the code iteratively rescales the cavity amplitude so that the optimized exit energy matches the requested design energy before storing the final phase.

28.3.7 Outputs and Persistence

After optimization, `OPALX`:

- stores the chosen phase back on the element
- marks the element with `APVETO` so it is not re-autophased again in the same pass
- records the phase in `OpalData`
- persists cavity phases through the H5 restart/output path

For non-optimizer runs, the code also writes an auxiliary `<element-name>_AP.dat` file with the on-axis tracked result used during the autophasing evaluation.

28.3.8 Practical Summary

For the current `OPALX` code base, the correct user-level description is:

- autophasing is implemented
- it is enabled globally with `OPTION, AUTOPHASE`
- it acts on `RFCAVITY` and `TRAVELINGWAVE`
- it can be disabled per element with `APVETO`
- the selected phase is the phase that maximizes the tracked reference-particle energy at cavity exit

Part IV

Physics

29 Overview

29.1 Introduction

OPALX is a fully three-dimensional program to track in time, relativistic particles taking into account space charge forces, self-consistently in the electrostatic approximation, and short-range longitudinal and transverse wake fields. OPALX is one of the few codes that is implemented using a parallel programming paradigm from the ground up. This makes OPALX indispensable for high statistics simulations of various kinds of existing and new accelerators. It has a comprehensive set of beamline elements, and furthermore allows arbitrary overlap of their fields, which gives OPALX a capability to model both the standing wave structure and traveling wave structure. Beside IMPACT-T it is the only code making use of space charge solvers based on an integrated Green [8], [9], [10] function to efficiently and accurately treat beams with large aspect ratio, and a shifted Green function to efficiently treat image charge effects of a cathode [11], [8], [9],[10]. For simulations of particles sources i.e. electron guns OPALX uses the technique of energy binning in the electrostatic space charge calculation to model beams with large energy spread. In the very near future a parallel Multigrid solver taking into account the exact geometry will be implemented.

29.2 Variables in OPALX

OPALX uses the following canonical variables to describe the motion of particles. The physical units are listed in square brackets.

X Horizontal position x of a particle relative to the axis of the element [m].

PX $\beta_x\gamma$ Horizontal canonical momentum Units.

Y Vertical position y of a particle relative to the axis of the element [m].

PY $\beta_y\gamma$ Vertical canonical momentum Units.

Z Longitudinal position z of a particle in floor co-ordinates [m].

PZ $\beta_z\gamma$ Longitudinal canonical momentum Units.

The independent variable is **t** [s].

29.3 Integration of the Equation of Motion

OPALX integrates the relativistic Lorentz equation

$$\frac{\mathrm{d}\gamma\mathbf{v}}{\mathrm{d}t} = \frac{q}{m}[\mathbf{E}_{ext} + \mathbf{E}_{sc} + \mathbf{v} \times (\mathbf{B}_{ext} + \mathbf{B}_{sc})]$$

where γ is the relativistic factor, q is the charge, and m is the rest mass of the particle. $\mathbf{E}$ and $\mathbf{B}$ are abbreviations for the electric field $\mathbf{E}(\mathbf{x}, t)$ and magnetic field $\mathbf{B}(\mathbf{x}, t)$. To update the positions and momenta OPALX uses the Boris-Buneman algorithm [12].

29.3.1 Implemented split-step update

The present OPALX tracker is the time-based `ParallelTracker` path. At the algorithmic level it implements a second-order leap-frog (`LF2`) style update with a Boris-Buneman momentum kick.

For a particle state $(\mathbf{x}^n, \mathbf{P}^{n-1/2})$, where $\mathbf{P} = \gamma\mathbf{v}/c$, one full step is organized as:

$$\mathbf{x}^{n+1/2} = \mathbf{x}^n + \frac{\Delta t}{2}\frac{\mathbf{P}^{n-1/2}}{\sqrt{1 + |\mathbf{P}^{n-1/2}|^2}},$$

then fields are assembled at that half-step position,

$$\mathbf{E}^{n+1/2} = \mathbf{E}_{sc}^{n+1/2} + \mathbf{E}_{ext}^{n+1/2}, \qquad \mathbf{B}^{n+1/2} = \mathbf{B}_{sc}^{n+1/2} + \mathbf{B}_{ext}^{n+1/2},$$

then the Boris momentum update is applied,

$$\mathbf{P}^{n+1/2} \xleftarrow{\frac{1}{2}\mathbf{E}} \xleftarrow{\mathbf{B}} \xleftarrow{\frac{1}{2}\mathbf{E}} \mathbf{P}^{n-1/2},$$

and finally the second half drift is performed,

$$\mathbf{x}^{n+1} = \mathbf{x}^{n+1/2} + \frac{\Delta t}{2}\frac{\mathbf{P}^{n+1/2}}{\sqrt{1 + |\mathbf{P}^{n+1/2}|^2}}.$$

In the actual code this appears as:

1. `timeIntegration1()`: first half position push
2. `resetFields()`
3. `computeSpaceChargeFields()`
4. `emitFromEmissionSources()`
5. `computeExternalFields()`
6. `timeIntegration2()`: Boris kick plus second half position push
7. `incrementT()` and `updateReference()`

So the implemented method is not just "a Boris pusher in isolation"; it is a split PIC/tracking loop in which the field solve and external-element queries are placed between two half drifts.

29.3.2 Frame handling during field assembly

One important implementation detail is that space-charge and external fields are not evaluated in the same coordinate frame.

For space charge, `computeSpaceChargeFields()` temporarily transforms the bunch from the reference frame into a beam-aligned frame whose z-axis follows the mean momentum. Self-fields are computed there, then the particle positions and the computed $\mathbf{E}$ and $\mathbf{B}$ contributions are rotated back to the reference frame.

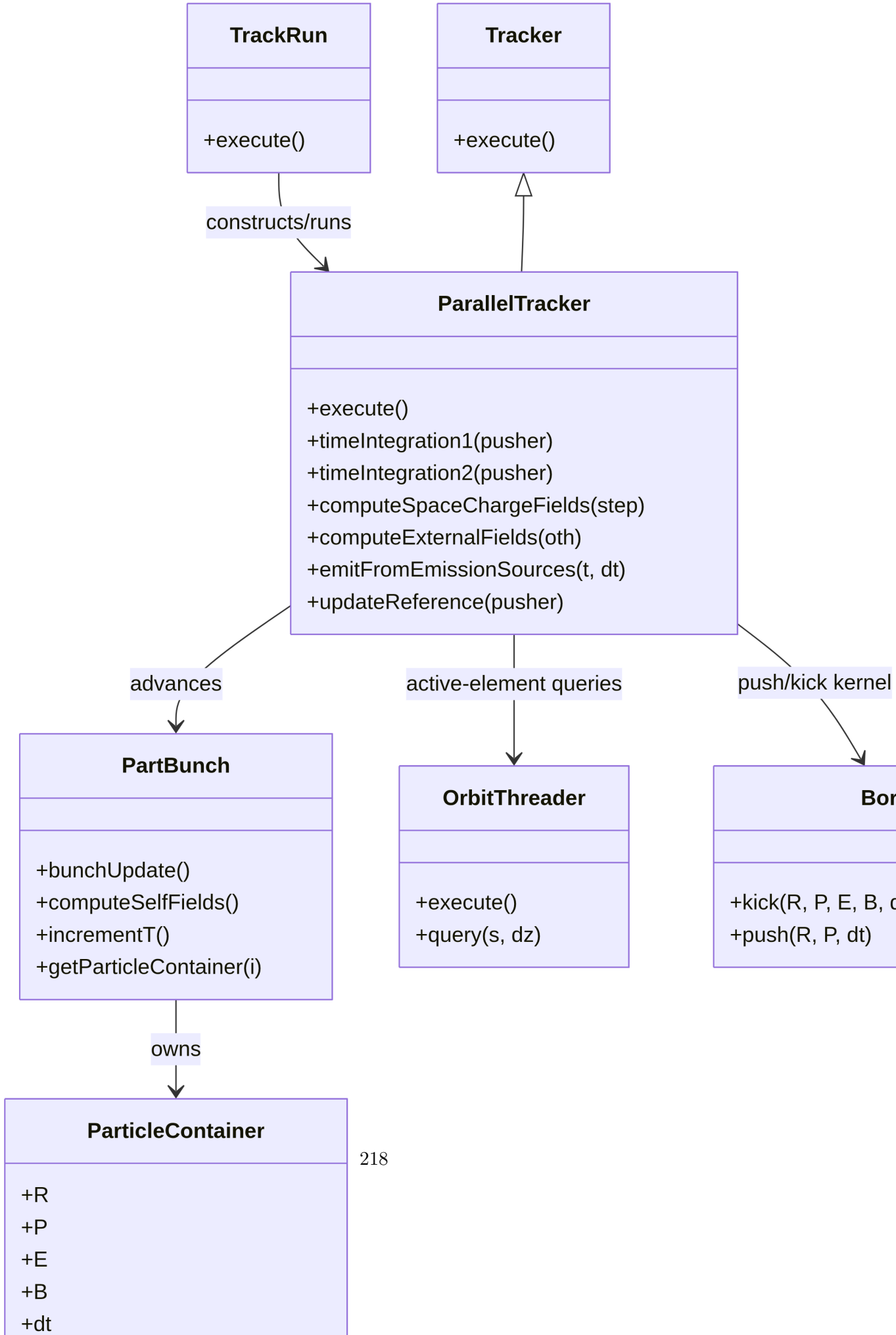
For external fields, `computeExternalFields()` queries the `OrbitThreader` active-set model and then transforms each particle container from the current reference frame into the local frame of each active element. The element field is applied there, and the bunch is transformed back afterwards. This is why OPALX can handle overlapping field supports and explicit element placement without reducing the machine to a purely one-dimensional owner-in-s model.

29.3.3 Boris momentum update

The implemented Boris kernel follows the standard relativistic Birdsall-Langdon form. In the notation of the code:

1. half electric impulse
$$\mathbf{P}^{-} = \mathbf{P}^{n-1/2} + \frac{\Delta t}{2}\frac{qc}{m}\mathbf{E},$$
2. magnetic rotation with
$$\gamma^{-} = \sqrt{1 + |\mathbf{P}^{-}|^2}, \qquad \mathbf{t} = \frac{\Delta t}{2}\frac{qc^2}{\gamma^{-} m}\mathbf{B}, \qquad \mathbf{s} = \frac{2\mathbf{t}}{1 + |\mathbf{t}|^2},$$
and
$$\mathbf{w} = \mathbf{P}^{-} + \mathbf{P}^{-} \times \mathbf{t}, \qquad \mathbf{P}^{+} = \mathbf{P}^{-} + \mathbf{w} \times \mathbf{s},$$
3. second half electric impulse
$$\mathbf{P}^{n+1/2} = \mathbf{P}^{+} + \frac{\Delta t}{2}\frac{qc}{m}\mathbf{E}.$$

This is the update implemented in the `BorisPusher` used by `ParallelTracker`.


TrackRun
+execute()
Tracker
+execute()
constructs/runs
ParallelTracker
+execute()
+timeIntegration1(pusher)
+timeIntegration2(pusher)
+computeSpaceChargeFields(step)
+computeExternalFields(oth)
+emitFromEmissionSources(t, dt)
+updateReference(pusher)
advances
active-element queries
push/kick kernel
PartBunch
+bunchUpdate()
+computeSelfFields()
+incrementT()
+getParticleContainer(i)
OrbitThreader
+execute()
+query(s, dz)
Bor
+kick(R, P, E, B, d
+push(R, P, dt)
owns
ParticleContainer
+R
+P
+E
+B
+dt

29.3.4 Integration class structure

The current integration stack is shown below.

The main point is that `ParallelTracker` orchestrates the full step, while `BorisPusher` only provides the local momentum and position update kernel. Field assembly, coordinate-frame changes, emission, and reference-particle updates are handled outside the pusher in the tracker loop itself.

29.4 Positioning of Elements

OPALX places elements in three-dimensional laboratory space and evaluates each active field in the element's local frame. Explicit placement uses translations and rotations; reference-line attributes may be compiled into the same spatial representation where supported by the element.

At runtime `OrbitThreader` queries the active supports near the reference particle. Each particle container is transformed into an active element's local frame for field evaluation and then transformed back. This permits overlapping field supports without assigning every longitudinal position to a single element.

Element length, field support, and aperture are distinct concepts. Models that need a finite transverse extent must define it through their supported geometry or aperture attributes rather than relying on an implicit beamline tube.

29.5 Coordinate Systems

The motion of a charged particle in an accelerator can be described by relativistic Hamilton mechanics. A particular motion is that of the reference particle, having the central energy and traveling on the so-called reference trajectory. Motion of a particle close to this fictitious reference particle can be described by linearized equations for the displacement of the particle under study, relative to the reference particle. In OPALX, the time t is used as independent variable instead of the path length s. The relation between them can be expressed as

$$\frac{\mathrm{d}}{\mathrm{d}t} = \frac{\mathrm{d}}{\mathrm{d}\mathbf{s}}\frac{\mathrm{d}\mathbf{s}}{\mathrm{d}t} = c\frac{\mathrm{d}}{\mathrm{d}\mathbf{s}}.$$

29.5.1 Global Cartesian Coordinate System

We define the global cartesian coordinate system, also known as floor coordinate system with K, a point in this coordinate system is denoted by $(X, Y, Z) \in K$. In Figure 29.2 the geometry of the accelerator is uniquely defined by the sequence of physical elements in K. The beam elements are numbered $e_0, \ldots, e_i, \ldots e_n$.

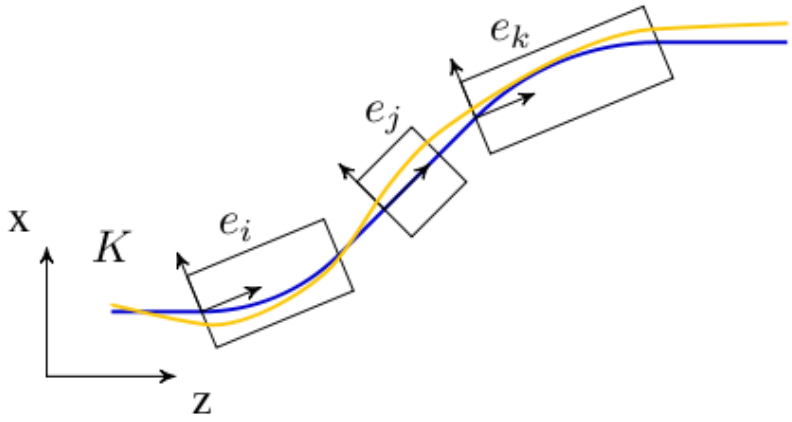


Figure 29.2: Illustration of local and global coordinates.

29.5.2 Local Cartesian Coordinate System

A local coordinate system K_i' is attached to each element e_i. This is simply a frame in which $(0,0,0)$ is at the entrance of each element. For an illustration see Figure 29.2. The local reference system $(x,y,z) \in K_n'$ may thus be referred to a global Cartesian coordinate system $(X,Y,Z) \in K$.

The local coordinates (x_i, y_i, z_i) at element e_i with respect to the global coordinates (X,Y,Z) are defined by three displacements (X_i, Y_i, Z_i) and three angles $(\Theta_i, \Phi_i, \Psi_i)$.

Ψ is the roll angle about the global Z-axis. Φ is the yaw angle about the global Y-axis. Lastly, Θ is the pitch angle about the global X-axis. All three angles form right-handed screws with their corresponding axes. The angles (Θ, Φ, Ψ) are the Tait-Bryan angles [13].

The displacement is described by a vector $\mathbf{v}$ and the orientation by a unitary matrix $\mathcal{W}$. The column vectors of $\mathcal{W}$ are unit vectors spanning the local coordinate axes in the order (x,y,z). $\mathbf{v}$ and $\mathcal{W}$ have the values:

$$\mathbf{v} = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}, \qquad \mathcal{W} = \mathcal{S}\mathcal{T}\mathcal{U}$$

where

$$\mathcal{S} = \begin{pmatrix} \cos\Theta & 0 & \sin\Theta \\ 0 & 1 & 0 \\ -\sin\Theta & 0 & \cos\Theta \end{pmatrix}, \quad \mathcal{T} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\Phi & \sin\Phi \\ 0 & -\sin\Phi & \cos\Phi \end{pmatrix}, \quad \mathcal{U} = \begin{pmatrix} \cos\Psi & -\sin\Psi & 0 \\ \sin\Psi & \cos\Psi & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

We take the vector $\mathbf{r}_i$ to be the displacement and the matrix $\mathcal{S}_i$ to be the rotation of the local reference system at the exit of the element i with respect to the entrance of that element.

Denoting with i a beam line element, one can compute $\mathbf{v}_i$ and $\mathcal{W}_i$ by the recurrence relations

$$\mathbf{v}_i = \mathcal{W}_{i-1}\mathbf{r}_i + \mathbf{v}_{i-1}, \qquad \mathcal{W}_i = \mathcal{W}_{i-1}\mathcal{S}_i,$$

where $\mathbf{v}_0$ corresponds to the origin of the `LINE` and $\mathcal{W}_0$ to its orientation. In OPALX they can be defined using either `X`, `Y`, `Z`, `THETA`, `PHI` and `PSI` or `ORIGIN` and `ORIENTATION`, see Simple Beam Lines.

29.5.3 Space Charge Coordinate System

In order to calculate space charge in the electrostatic approximation, we introduce a co-moving coordinate system K_{sc}, in which the origin coincides with the mean position of the particles and the mean momentum is parallel to the z-axis.

29.5.4 Curvilinear Coordinate System

In order to compute statistics of the particle ensemble, K_s is introduced. The accompanying tripod (Dreibein) of the reference orbit spans a local curvilinear right handed system (x, y, s). The local s-axis is the tangent to the reference orbit. The two other axes are perpendicular to the reference orbit and are labelled x (in the bend plane) and y (perpendicular to the bend plane).

Both coordinate systems are described in Figure 29.3.

29.5.5 Design or Reference Orbit

The reference orbit consists of a series of straight sections and circular arcs and is **computed** by the Orbit Threader i.e. deduced from the element placement in the floor coordinate system.

29.5.6 Compatibility Mode

To facilitate the change for users we will provide a compatibility mode. The idea is that the user does not have to change the input file. Instead OPALX will compute the positions of the elements. For this it uses the bend angle and chord length of the dipoles and the position of the elements along the trajectory. The user can choose whether effects due to fringe fields are considered when computing the path length of dipoles or not. The option to toggle OPALX's behavior is called `IDEALIZED`. OPALX assumes per default that provided

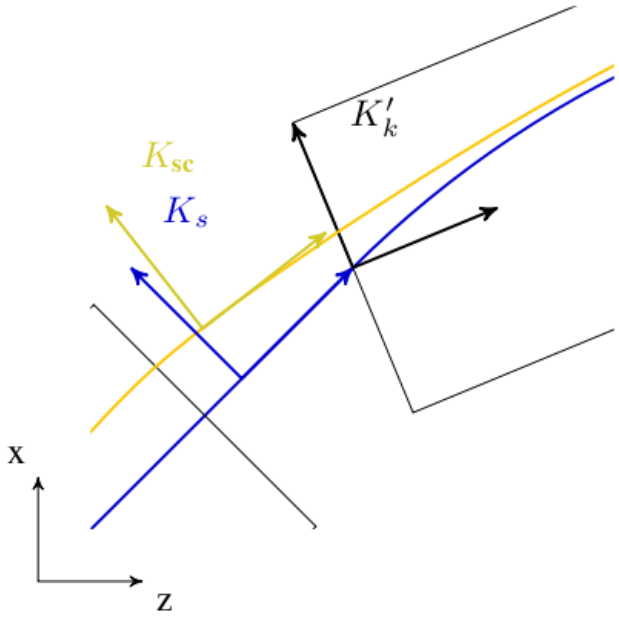


Figure 29.3: Illustration of K_{sc} and K_s

`ELEMEDGE` for elements downstream of a dipole are computed without any effects due to fringe fields.

Elements that overlap with the fields of a dipole have to be handled separately by the user to position them in 3D.

We split the positioning of the elements into two steps. In a first step we compute the positions of the dipoles. Here we assume that their fields don't overlap. In a second step we can then compute the positions and orientations of all other elements.

The accuracy of this method is good for all elements except for those that overlap with the field of a dipole.

29.5.7 Orbit Threader and Autophasing

The `OrbitThreader` integrates a design particle through the lattice and setups up a multi map structure (`IndexMap`). Furthermore when the reference particle hits an rf-structure for the first time then it auto-phases the rf-structure, see Appendix Auto-phasing Algorithm. The multi map structure speeds up the search for elements that influence the particles at a given position in 3D space by minimizing the looping over elements when integrating an ensemble of particles. For each time step, `IndexMap` returns a set of elements $\mathcal{S}_{\mathrm{e}} \subset e_0 \ldots e_n$ in case of the example given in Figure 29.2. An implicit drift is modelled as an empty set $\emptyset$. The `IndexMap` therefore represents an occupancy model on the reference coordinate: each interval stores the set of elements whose field-support extents are active on that interval. This is distinct from the geometric placement of the corresponding element bodies. In

overlapping regions the design particle sees the superposition of all active fields, and the reference path is determined by integrating this combined field.

29.6 Flow Diagram of OPALX

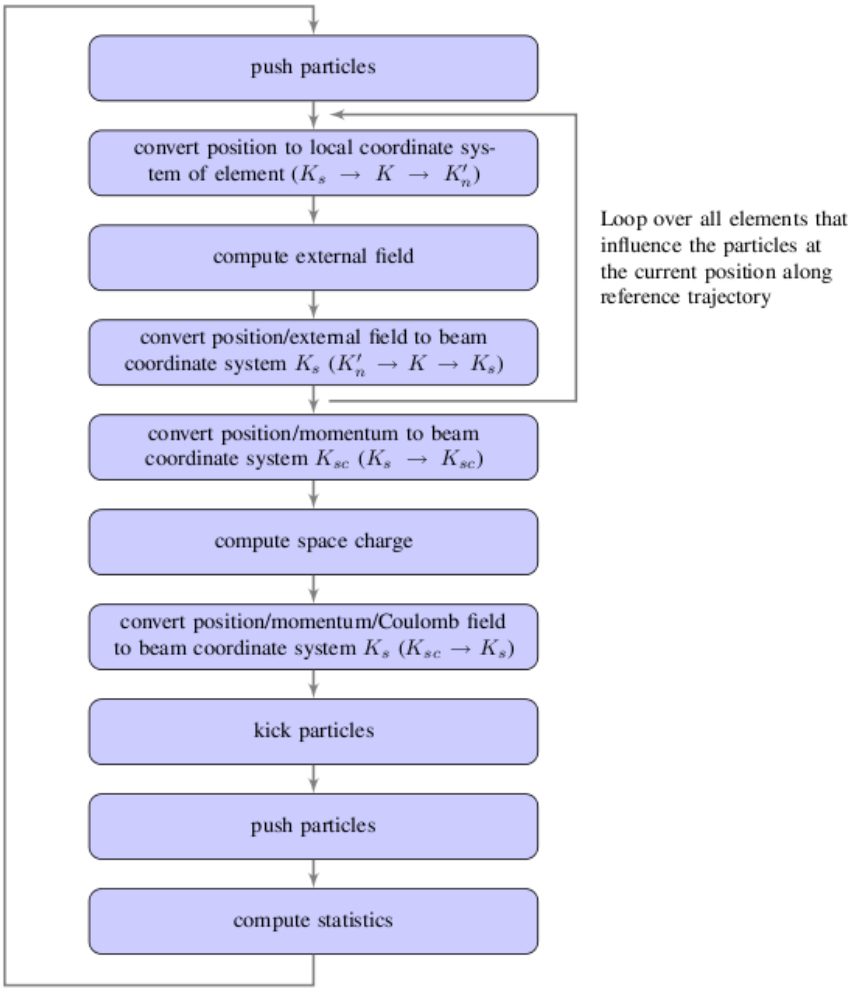


Figure 29.4: Schematic workflow of OPALX's execute method.

A regular time step in OPALX is sketched in Figure 29.4. In order to compute the coordinate system transformation from the reference coordinate system K_s to the local coordinate systems K'_n we join the transformation from floor coordinate system K to K'_n to the transformation from K_s to K. All computations of rotations which are involved in the computation of coordinate system transformations are performed using quaternions. The resulting quaternions are then converted to the appropriate matrix representation before applying the rotation operation onto the particle positions and momenta.

As can be seen from Figure 29.4 the integration of the trajectories of the particles are

integrated and the computation of the statistics of the six-dimensional phase space are performed in the reference coordinate system.

29.7 Multipoles in different Coordinate systems

In the following sections there are three models presented for the fringe field of a multipole. The first one deals with a straight multipole, while the second one treats a curved multipole, both starting with a power expansion for the magnetic field. The last model tries to be different by starting with a more compact functional form of the field which is then adapted to straight and curved geometries.

29.7.1 Fringe field models

(for a straight multipole)

Most accelerator modeling codes use the hard-edge model for magnets - constant Hamiltonian. Real magnets always have a smooth transition at the edges - fringe fields. To obtain a multipole description of a field we can apply the theory of analytic functions.

$$\begin{aligned} \nabla \cdot \mathbf{B} = 0 \Rightarrow \exists \quad \mathbf{A} \quad \text{with} \quad \mathbf{B} = \nabla \times \mathbf{A} \\ \nabla \times \mathbf{B} = 0 \Rightarrow \exists \quad V \quad \text{with} \quad \mathbf{B} = -\nabla V \end{aligned}$$

Assuming that A has only a non-zero component A_s we get

$$B_x = -\frac{\partial V}{\partial x} = \frac{\partial A_s}{\partial y}$$

$$B_y = -\frac{\partial V}{\partial y} = -\frac{\partial A_s}{\partial x}$$

These equations are just the Cauchy-Riemann conditions for an analytic function $\tilde{A}(z) = A_s(x, y) + iV(x, y)$. So the complex potential is an analytic function and can be expanded as a power series

$$\tilde{A}(z) = \sum_{n=0}^{\infty} \kappa_n z^n, \quad \kappa_n = \lambda_n + i\mu_n$$

with λ_n, μ_n being real constants. It is practical to express the field in cylindrical coordinates (r, φ, s)

$$\begin{aligned} x = r\cos\varphi \quad y = r\sin\varphi \\ z^n = r^n(\cos n\varphi + i\sin n\varphi) \end{aligned}$$

From the real and imaginary parts of equation () we obtain

$$V(r,\varphi) = \sum_{n=0}^{\infty} r^n(\mu_n \cos n\varphi + \lambda_n \sin n\varphi)$$
$$A_s(r,\varphi) = \sum_{n=0}^{\infty} r^n(\lambda_n \cos n\varphi - \mu_n \sin n\varphi)$$

Taking the gradient of $-V(r,\varphi)$ we obtain the multipole expansion of the azimuthal and radial field components, respectively

$$B_\varphi = -\frac{1}{r}\frac{\partial V}{\partial \varphi} = -\sum_{n=0}^{\infty} n r^{n-1}(\lambda_n \cos n\varphi - \mu_n \sin n\varphi)$$
$$B_r = -\frac{\partial V}{\partial r} = -\sum_{n=0}^{\infty} n r^{n-1}(\mu_n \cos n\varphi + \lambda_n \sin n\varphi)$$

Furthermore, we introduce the normal multipole coefficient b_n and skew coefficient a_n defined with the reference radius r_0 and the magnitude of the field at this radius B_0 (these coefficients can be a function of s in a more general case as it is presented further on).

$$b_n = -\frac{n\lambda_n}{B_0} r_0^{n-1} \qquad a_n = \frac{n\mu_n}{B_0} r_0^{n-1}$$

$$B_\varphi(r,\varphi) = B_0 \sum_{n=1}^{\infty}(b_n \cos n\varphi + a_n \sin n\varphi)\left(\frac{r}{r_0}\right)^{n-1}$$
$$B_r(r,\varphi) = B_0 \sum_{n=1}^{\infty}(-a_n \cos n\varphi + b_n \sin n\varphi)\left(\frac{r}{r_0}\right)^{n-1}$$

To obtain a model for the fringe field of a straight multipole, a proposed starting solution for a non-skew magnetic field is

$$V = \sum_{n=1}^{\infty} V_n(r,z) \sin n\varphi$$
$$V_n = \sum_{k=0}^{\infty} C_{n,k}(z) r^{n+2k}$$

It is straightforward to derive a relation between coefficients

$$\nabla^2 V = 0 \Rightarrow \frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial V_n}{\partial r}\right) + \frac{\partial^2 V_n}{\partial z^2} = \frac{n^2 V_n}{r^2} = 0$$

$$V_n = \sum_{k=0}^{\infty} C_{n,k}(z) r^{n+2k}$$

$$\Rightarrow \sum_{k=0}^{\infty} \left[r^{n+2(k-1)} \left[(n+2k)^2 - n^2 \right] C_{n,k}(z) + r^{n+2k} \frac{\partial^2 C_{n,k}(z)}{\partial z^2} \right] = 0$$

By identifying the term in front of the same powers of r we obtain the recurrence relation

$$C_{n,k}(z) = -\frac{1}{4k(n+k)} \frac{d^2 C_{n,k-1}}{dz^2}, k = 1, 2, ...$$

The solution of the recursion relation becomes

$$C_{n,k}(z) = (-1)^k \frac{n!}{2^{2k} k!(n+k)!} \frac{d^{2k} C_{n,0}(z)}{dz^{2k}}$$

Therefore

$$V_n = -\left(\sum_{k=0}^{\infty} (-1)^{k+1} \frac{n!}{2^{2k} k!(n+k)!} C_{n,0}^{(2k)}(z) r^{2k} \right) r^n$$

The transverse components of the field are

$$B_r = \sum_{n=1}^{\infty} g_{rn} r^{n-1} \sin n\varphi$$
$$B_\varphi = \sum_{n=1}^{\infty} g_{\varphi n} r^{n-1} \cos n\varphi$$

where the following gradients determine the entire potential and can be deduced from the function $C_{n,0}(z)$ once the harmonic n is fixed.

$$g_{rn}(r,z) = \sum_{k=0}^{\infty} (-1)^{k+1} \frac{n!(n+2k)}{2^{2k} k!(n+k)!} C_{n,0}^{(2k)}(z) r^{2k}$$
$$g_{\varphi n}(r,z) = \sum_{k=0}^{\infty} (-1)^{k+1} \frac{n! n}{2^{2k} k!(n+k)!} C_{n,0}^{(2k)}(z) r^{2k}$$

The z-directed component of the filed can be expressed in a similar form

$$B_z = -\frac{\partial V}{\partial z} = \sum_{n=1}^{\infty} g_{zn} r^n \sin n\varphi$$
$$g_{zn} = \sum_{k=0}^{\infty} (-1)^{k+1} \frac{n!}{2^{2k} k!(n+k)!} C_{n,0}^{2k+1} r^{2k}$$

The gradient functions $g_{rn}, g_{\varphi n}, g_{zn}$ are obtained from

$$B_{r,n} = -\frac{\partial V_n}{\partial r}\sin n\varphi = g_{rn} r^{n-1}\sin n\varphi$$
$$B_{\varphi,n} = -\frac{n}{r}V_n \cos n\varphi = g_{\varphi n} r^{n-1}\cos n\varphi$$
$$B_{z,n} = -\frac{\partial V_n}{\partial z}\sin n\varphi = g_{zn} r^{n}\sin n\varphi$$

One preferred model to approximate the gradient profile on the central axis is the k-parameter Enge function

$$C_{n,0}(z) = \frac{G_0}{1 + exp[P(d(z))]}, \quad G_0 = \frac{B_0}{r_0^{n-1}}$$
$$P(d) = C_0 + C_1\left(\frac{d}{\lambda}\right) + C_2\left(\frac{d}{\lambda}\right)^2 + ... + C_{k-1}\left(\frac{d}{\lambda}\right)^{k-1}$$

where $d(z)$ is the distance to the field boundary and λ characterizes the fringe field length.

29.7.2 Fringe field of a curved multipole

(fixed radius)

We consider the Frenet-Serret coordinate system $(\hat{\mathbf{x}}, \hat{\mathbf{s}}, \hat{\mathbf{z}})$ with the radius of curvature ρ constant and the scale factor $h_s = 1 + x/\rho$. A conversion to these coordinates implies that

$$\nabla \cdot \mathbf{B} = \frac{1}{h_s}\left[\frac{\partial(h_s B_x)}{\partial x} + \frac{\partial B_s}{\partial s} + \frac{\partial(h_s B_z)}{\partial z}\right]$$

$$\nabla \times \mathbf{B} = \frac{1}{h_s}\left[\frac{\partial B_z}{\partial s} - \frac{\partial(h_s B_s)}{\partial z}\right]\hat{\mathbf{x}} + \left[\frac{\partial B_x}{\partial z} - \frac{\partial B_z}{\partial x}\right]\hat{\mathbf{s}} + \frac{1}{h_s}\left[\frac{\partial(h_s B_s)}{\partial x} - \frac{\partial B_x}{\partial s}\right]\hat{\mathbf{z}}$$

To simplify the problem, consider multipoles with mid-plane symmetry, i.e.

$$b_z(z) = B_z(-z) \qquad B_x(z) = -B_x(-z) \qquad B_s(z) = -B_s(-z)$$

The most general form of the expansion is

$$B_z = \sum_{i,k=0}^{\infty} b_{i,k} x^i z^{2k}$$
$$B_x = z\sum_{i,k=0}^{\infty} a_{i,k} x^i z^{2k}$$
$$B_s = z\sum_{i,k=0}^{\infty} c_{i,k} x^i z^{2k}$$

Maxwell's equations $\nabla \cdot \mathbf{B} = 0$ and $\nabla \times \mathbf{B} = 0$ in the above coordinates yield

$$\frac{\partial}{\partial x}\left((1+x/\rho)B_x\right) + \frac{\partial B_s}{\partial s} + (1+x/\rho)\frac{\partial B_z}{\partial z} = 0$$

$$\frac{\partial B_z}{\partial s} = (1+x/\rho)\frac{\partial B_s}{\partial z}$$

$$\frac{\partial B_x}{\partial z} = \frac{\partial B_z}{\partial s}$$

$$\frac{\partial B_x}{\partial s} = \frac{\partial}{\partial x}\left((1+x/\rho)B_s\right)$$

The substitution of the general-form equation into Maxwell's equations allows for the derivation of recursion relations. Maxwell's equations gives

$$\sum_{i,k=0}^{\infty} a_{i,k}(2k+1)x^i z^{2k} = \sum_{i,k=0}^{\infty} b_{i,k} i x^{i-1} z^{2k}$$

Equating the powers in $x^i z^{2k}$

$$a_{i,k} = \frac{i+1}{2k+1} b_{i+1,k}$$

A similar result is obtained from Maxwell's equations

$$\sum_{i,k=0}^{\infty} \partial_s b_{i,k} x^i z^{2k} = \left(1+\frac{x}{\rho}\right) \sum_{i,k=0}^{\infty} c_{i,k}(2k+1)x^i z^{2k}$$
$$\Rightarrow c_{i,k} + \frac{1}{\rho} c_{i-1,k} = \frac{1}{2k+1}\partial_s b_{i,k}$$

The last equation from $\nabla \times \mathbf{B} = 0$ should be consistent with the two recursion relations obtained. Maxwell's equations implies

$$\sum_{i,k=0}^{\infty} \left[\frac{i+1}{\rho} c_{i,k} x^i + c_{i,k} i x^{i-1}\right] z^{k+1} = \sum_{i,k=0}^{\infty} \partial_s a_{i,k} x^i z^{2k}$$

$$\Rightarrow \frac{\partial_s a_{i,k}}{i+1} = \frac{1}{\rho} c_{i,k} + c_{i+1,k}$$

This results follows directly from the a-factor relation and the c-factor relation; therefore the relations are consistent. Furthermore, the last required relations is obtained from the divergence of $\mathbf{B}$

$$\sum_{i,k=0}^{\infty}\left[\frac{a_{i,k}x^i z^{2k+1}}{\rho}+ia_{i,k}x^{i-1}z^{2k+1}+\frac{ia_{i,k}x^i z^{2k+1}}{\rho}+\partial_s c_{i,k}x^i z^{2k+1}+2kb_{i,k}x^i z^{2k-1}\right]=0$$

$$\Rightarrow \partial_s c_{i,k}+\frac{2(k+1)}{\rho}b_{i-1,k+1}+2(k+1)b_{i,k+1}+\frac{1}{\rho}a_{i,k}+(i+1)a_{i+1,k}+\frac{1}{\rho}a_{i,k}=0$$

Using the relation (the a-factor relation) to replace the a coefficients with b's we arrive at

$$\partial_s c_{i,k}+\frac{(i+1)^2}{\rho(2k+1)}b_{i+1,k}+\frac{(i+1)(i+2)}{2k+1}b_{i+2,k}+\frac{2(k+1)}{\rho}b_{i-1,k+1}+2(k+1)b_{i,k+1}=0$$

All the coefficients above can be determined recursively provided the field B_z can be measured at the mid-plane in the form

$$B_z(z=0)=B_{0,0}+B_{1,0}x+B_{2,0}x^2+B_{3,0}x^3+...$$

where $B_{i,0}$ are functions of s and they can model the fringe field for each multipole term x^n. As an example, for a dipole magnet, the $B_{1,0}$ function can be model as an Enge function or $tanh$.

29.7.3 Fringe field of a curved multipole

(variable radius of curvature)

The difference between this case and the above is that ρ is now a function of s, $\rho(s)$. We can obtain the same result starting with the same functional forms for the field (the general-form equation). The result of the previous section also holds in this case since no derivative $\frac{\partial}{\partial s}$ is applied to the scale factor h_s. If the radius of curvature is set to be proportional to the dipole filed observed by some reference particle that stays in the centre of the dipole

$$\rho(s)\propto B(z=0,x=0,s)=B_x(z=0,x=0)=b_{0,0}(s)$$

29.7.4 Fringe field of a multipole

This is a different, more compact treatment The derivation is more clear if we gather the variables together in functions. We assume again mid-plane symmetry and that the z-component of the field in the mid-plane has the form

$$B_z(x, z = 0, s) = T(x)S(s)$$

where $T(s)$ is the transverse field profile and $S(s)$ is the fringe field. One of the requirements of the symmetry is that $B_z(z) = B_z(-z)$ which using a scalar potential ψ requires $\frac{\partial \psi}{\partial z}$ to be an even function in z. Therefore, ψ is an odd function in z and can be written as

$$\psi = z f_0(x, s) + \frac{z^3}{3!} f_1(x, s) + \frac{z^5}{5!} f_3(x, s) + ...$$

The given transverse profile requires that $f_0(x, s) = T(x)S(s)$, while $\nabla^2 \psi = 0$ follows from Maxwell's equations as usual, more explicitly

$$\frac{\partial}{\partial x}\left(h_s \frac{\partial \psi}{\partial x}\right) + \frac{\partial}{\partial s}\left(\frac{1}{h_s}\frac{\partial \psi}{\partial s}\right) + \frac{\partial}{\partial z}\left(h_s \frac{\partial \psi}{\partial z}\right) = 0$$

For a straight multipole $h_s = 1$. Laplace's equation becomes

$$\sum_{n=0} \frac{z^{2n+1}}{(2n+1)!}\left[\partial_x^2 f_n(x, s) + \partial_s^2 f_n(x, s)\right] + \sum_{n=1} f_n(x, s)\frac{z^{n-1}}{(n-1)!} = 0$$

By equating powers of z we obtain the recursion relation

$$f_{n+1}(x, s) = -\left(\partial_x^2 + \partial_s^2\right) f_n(x, s)$$

The general expression for any $f_n(x, s)$ is then obtained from the mid-plane field by

$$f_n(x, s) = (-1)^n \left(\partial_x^2 + \partial_s^2\right)^n f_0(x, s)$$

$$f_n(x, s) = (-1)^n \sum_{i=0}^{n} \binom{n}{i} T^{(2i)}(x) S^{(2n-2i)}(s)$$

For a curved multipole of constant radius $h_s = 1 + \frac{x}{\rho}$ with $\rho = const.$ The corresponding Laplace's equation is

$$\left(\frac{1}{\rho h_s}\partial_x + \partial_x^2 + \partial_z^2 + \frac{\partial_s^2}{h_s^2}\right)\psi = 0$$

Again we substitute with the functional form of the potential to get the recursion

$$f_{n+1}(x,s) = -\left[\frac{1}{\rho+x}\partial_x + \partial_x^2 + \frac{\partial_s^2}{(1+x/\rho)^2}\right] f_n(x,s)$$

$$f_{n+1}(x,s) = (-1)^n \left[\frac{1}{\rho+x}\partial_x + \partial_x^2 + \frac{\partial_s^2}{(1+x/\rho)^2}\right]^n f_0(x,s)$$

Finally consider what changes for $\rho = \rho(s)$. Laplace's equation is

$$\left[\frac{1}{\rho h_s}\partial_x + \partial_x^2 + \partial_z^2 + \frac{\partial_s^2}{h_s^2} + \frac{x}{\rho^2 h_s^3}(\partial_s\rho)\partial_s\right]\psi = 0$$

The last step is again the substitution to get

$$f_{n+1}(x,s) = -\left[\frac{\partial_x}{\rho h_s} + \partial_x^2 + \partial_z^2 + \frac{1}{h_s^2}\partial_s^2 + \frac{x}{\rho^2 h_s^3}(\partial_s\rho)\partial_s\right] f_n(x,s)$$

$$f_n(x,s) = (-1)^n \left[\frac{\partial_x}{\rho h_s} + \partial_x^2 + \partial_z^2 + \frac{\partial_s^2}{h_s^2} + \frac{x}{\rho^2 h_s^3}(\partial_s\rho)\partial_s\right]^n f_0(x,s)$$

If the radius of curvature is proportional to the dipole field in the centre of the multipole (the dipole component of the transverse field is a constant $T_{dipole}(x) = B_0$ and

$$\rho(s) = B_0 \times S(s)$$

This expression can be substituted into the preceding recursion to obtain a more explicit equation.

30 Coordinate Systems

This chapter collects the current coordinate-system and placement language for OPALX. It has been migrated from the OPALX physics notes and is being checked against the orbit-threading and element-placement implementation.

30.1 Quick Reader

30.1.1 Symbols and notation

To keep the formulas uniform, this chapter uses the following notation.

Symbol	Meaning
K	Laboratory or floor Cartesian frame with coordinates $\mathbf{r} = (X, Y, Z)$.
$\mathcal{B}$	Reference base of the lattice, typically an interval for a transfer line or a closed path for a ring.
s	Reference or reporting coordinate on $\mathcal{B}$; in an s-integrator it is also the independent variable.
t	Physical time and the independent variable of the current OPALX tracker.
U_i, u_i	Canonical local chart of element i and the corresponding local coordinates.
$\psi_i : U_i \to \mathbb{R}^3$	Intrinsic geometry map of element i in its canonical frame.
$\Omega_i \subset U_i$	Local support region on which the element field, aperture, or material model is defined.
$\Sigma_{i,\alpha} \subset \partial\Omega_i$	Marked boundary chart of element i; this is the geometric origin of a port.
$p_{i,\alpha} = (\Sigma_{i,\alpha}, F_{i,\alpha})$	Port α of element i, consisting of a boundary chart together with an adapted frame $F_{i,\alpha}$.
$P_i(u_i, t)$	Placement map that embeds the local chart of element i into the laboratory frame.
$\mathcal{S}_i(t) = P_i(\Omega_i, t)$	Placed support of element i in laboratory space at time t.

Symbol	Meaning
$T_i(t)$	Rigid placement transform of element i at time t; it may be written as a rotation matrix $R_i(t)$ together with a translation vector $\mathbf{a}_i(t)$.
$\Gamma_i(\ell)$	Optional local reference path of element i, parameterized by the local longitudinal coordinate $\ell \in [0, L_i]$.
C_{pq}	Rigid transition or patch transform from port p to port q.
g_{ij}	Transition map between neighboring local charts.

30.1.2 A geometric viewpoint

Following Forest and Hirata [14], one should distinguish the placed physical object from the coordinates used to describe it. In that spirit, an OPALX element is not first of all an interval in one global coordinate s. It is a local chart U_i, an intrinsic geometry map ψ_i, a support region $\Omega_i \subset U_i$, and a finite set of marked interfaces through which that local chart is connected to neighboring charts.

This is the origin of the *port* notion used throughout this chapter. A port of element i is a marked boundary chart

$$\Sigma_{i,\alpha} \subset \partial\Omega_i.$$

together with an adapted frame $F_{i,\alpha}$. In ordinary beamline language these are simply the entrance face, exit face, branch face, fiducial interface, or closure interface of the object in case of a circular machine. The advantage of introducing the word "port" is that the same notion works for a straight drift, a curved bend, a patch, a branch, or a ring-closing connection.

When two ports are connected, OPALX stores a transition datum between them. In differential-geometric language this is the chart-to-chart handoff across an interface. In fibre-bundle language it is the transition function relating two local trivializations [14]. The port notion therefore does not introduce new physics. It is the implementation-level representative of how local charts are glued together.

A transfer line is then an open chain of ports, while a ring is a closed chain, possibly only up to a prescribed symmetry transform. That is the geometric reason the same placement model can cover both linear and circular machines.

30.1.3 Why the independent variable matters

Comparisons with established accelerator codes provide a useful baseline: MAD-X describes the machine primarily as an ordered sequence of elements along a reference orbit, and survey or optics calculations are performed relative to that backbone [15], [16]. The later code survey then showed four distinct placement philosophies relative to that baseline: Elegant remains strongly sequence-centric, MARYLIE is more map-first, IMPACT-T uses ordered field regions with local frame changes, and Bmad exposes geometry operators such as patches and fiducials [17], [18], [19], [20].

In an ***s*-based code**, placement is fundamentally one-dimensional. An element is placed by specifying where it begins and ends on the design curve. Once that curve is known, global survey coordinates are derived by transporting a local frame along it. The tracker can usually decide which element is active by asking where the particle is in s.

In a ***t*-based code**, placement is fundamentally spatial. A particle is advanced in laboratory time, self-fields are evaluated on a common time slice, and different particles in the bunch may simultaneously occupy different parts of the machine. Therefore the code cannot rely on a single global "current element" inferred from one common s. Each element must instead have a placed spatial support, a local coordinate frame, ports that describe how it connects to neighbours, and usually timing metadata such as RF phase origin or trigger offset.

This leads to the main OPALX conclusion:

1. topology,
2. survey or reference-curve metadata,
3. rigid placement in laboratory space, and
4. physics field evaluation

should be treated as separate concepts, even if they are often conflated in traditional s-based lattice descriptions.

30.2 Geometry Design Principles

30.2.1 Forest-Hirata bundles versus differential geometry

The Forest-Hirata appendix gives a clean abstract reformulation of local element placement [14]. The lattice is treated as a fibre bundle over the machine loop; local neighbourhoods are element intervals; the fibres may be coordinate frames, particle coordinates, maps, or envelopes; and the matching block between local descriptions is a transition function $g_{ij} \in G$.

The present OPALX note uses a more concrete differential-geometric language. Each element may carry a reference path $\Gamma_i(\ell)$, a moving orthonormal frame, marked interface charts, and a placed support region in laboratory space. This is closer to implementation because it directly supports field evaluation, aperture checks, and coordinate conversion inside an element.

The two viewpoints answer different questions:

1. use differential geometry locally inside an element whenever a curved or adapted chart is numerically useful;
2. use fibre-bundle-style transition maps to connect neighboring charts, branches, anchors, and closures without pretending that there must be one globally preferred frame.

30.2.2 Current OPALX implementation context

The current tracker advances particles in physical time with a Boris-Buneman update. Elements have spatial placement and local field frames, and the orbit-threading layer queries active supports during field assembly. The design described here extends those implemented concepts into one geometry representation without importing a second machine model from historical documentation.

30.2.3 A geometry graph and placement tuple

The cleanest generalization is to represent machine geometry by a directed placement graph

$$\mathcal{G} = (\mathcal{N}, \mathcal{E}).$$

Here $\mathcal{N}$ is the set of named ports and free anchors, and $\mathcal{E}$ is the set of placed objects that span those ports. A placed object may be a physics element, a pure drift, or a tracking-neutral geometry operator such as a patch or survey shift.

This representation unifies linear and circular machines immediately:

1. a transfer line is an open connected path in $\mathcal{G}$ with two exposed boundary ports,
2. a ring is a closed cycle in $\mathcal{G}$,
3. a sector machine is a subgraph together with one or more symmetry transforms that replicate it.

Every placed object i should be representable by a tuple

$$\mathfrak{E}_i = (M_i,\, U_i,\, \Omega_i,\, \psi_i,\, p_i^{\mathrm{in}},\, p_i^{\mathrm{out}},\, A_i,\, B_i,\, \Delta T_i^{\mathrm{err}},\, \Delta T_i^{\mathrm{dyn}}(t),\, \tau_i).$$

Here M_i is the physics model, U_i the canonical local chart, $\Omega_i \subset U_i$ the local support, ψ_i the intrinsic geometry map, p_i^{in} and p_i^{out} the named ports, A_i and B_i the nominal entrance and exit transforms, $\Delta T_i^{\mathrm{err}}$ the static alignment-error transform, $\Delta T_i^{\mathrm{dyn}}(t)$ any time-dependent motion, and τ_i timing metadata.

If $T(p_i^{\mathrm{in}})$ denotes the laboratory-frame pose of the entrance port, then the runtime pose of the canonical element frame is

$$T_i(t) = T(p_i^{\mathrm{in}})\, A_i\, \Delta T_i^{\mathrm{err}}\, \Delta T_i^{\mathrm{dyn}}(t).$$

The corresponding exit-port pose is

$$T(p_i^{\mathrm{out}}) = T_i(t)\, B_i.$$

30.2.4 Transition maps as first-class objects

Whenever two local charts meet, OPALX should store two closely related pieces of information:

1. the rigid port-to-port patch transform C_{pq} attached to the named interface;
2. the induced chart-change map $g_{ij}(t) = P_i^{-1}(\cdot, t) \circ P_j(\cdot, t)$ when an actual local chart conversion is needed.

If the common region is only a glued entrance or exit face, g_{ij} reduces to the familiar rigid handoff encoded by C_{pq}. If there is volumetric overlap, g_{ij} provides a precise way to translate diagnostics, apertures, or field-evaluation points from one local chart to the other.

30.2.5 Two complementary coordinate charts

For OPALX it is useful to distinguish explicitly between:

1. a **reference chart** s for lattice description, sequence order, optics-style diagnostics, and survey reports;
2. a **world chart** $(\mathbf{r}, t)$ for actual field evaluation, particle pushing, aperture collisions, and self-field coupling.

In a purely MAD-X-like mode the two charts nearly collapse onto one another. In a time-based OPALX mode they do not. The code should therefore allow both to coexist without forcing one to masquerade as the other.

30.2.6 Minimal front-end semantics

The front-end syntax should compile to the graph-based placement model above. The intent is to extend the present manual concepts rather than replace them with an unrelated syntax.

Input notion	Keep or extend	Proposed semantics
`LINE`	Keep	Ordered open path in the placement graph; not a separate machine type.

Input notion	Keep or extend	Proposed semantics
Anchors and explicit placement data	Keep	Explicit anchor pose for a named frame or for the root of a line.
`ELEMEDGE`	Keep as compatibility mode	Reference coordinate only; compile once into explicit transforms and do not use it directly for runtime ownership.
Alignment errors	Keep	Populate the static error transform $\Delta T_i^{\mathrm{err}}$.
Named ports and patches	Add or expose explicitly	Declare interface charts, pure geometry transforms, and closure constraints independently of the physics elements.
Chart and support attributes	Add	Declare `STRAIGHT`, `ARC`, or `USER` intrinsic charts, explicit support regions, and timing metadata.

30.3 Geometric Background

This section records the small amount of differential-geometric and fibre-bundle language used in the placement discussion. The point, following Forest and Hirata [14], is to separate the placed object from any one chosen coordinate description.

30.3.1 The bookkeeping base

Let $\mathcal{B}$ denote the *reference base* used for ordering and reporting. For a transfer line, $\mathcal{B}$ may be an interval. For a ring, $\mathcal{B}$ may be a circle. For a machine whose reference-order topology contains a junction, an extraction line, a splitter, or a switchyard, $\mathcal{B}$ may be a graph with labeled edges and junction nodes.

The important point is that $\mathcal{B}$ is a bookkeeping base, not the full physical geometry.

30.3.2 Local charts and supports

An individual element is described locally by a chart

$$\psi_i : U_i \to \mathbb{R}^3,$$

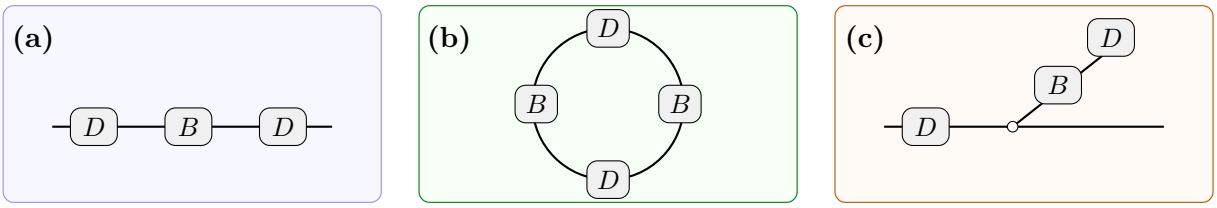


Figure 30.1: Examples of the bookkeeping base $\mathcal{B}$: (a) interval, the reference base for an open line; (b) circle, the reference base for a ring; and (c) graph with junctions, appropriate for a switchyard, splitter, or extraction topology. The same placement formalism can sit over all three.

where U_i is the canonical local coordinate domain of element i. The subset

$$\Omega_i \subset U_i$$

is the local support on which the field, aperture, material, or map model is defined. The local block picture is not yet the lab-frame placement. A placement map P_i rotates and translates that local description into the laboratory frame.

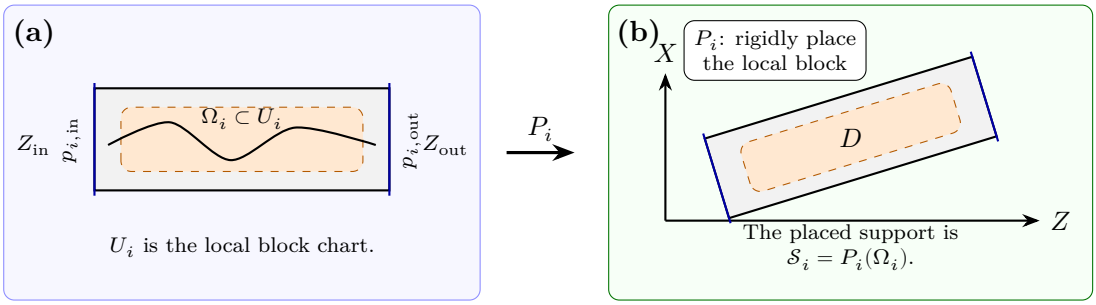


Figure 30.2: Local versus placed descriptions of an element: (a) a local chart with support $\Omega_i \subset U_i$ and ports $p_{i,\text{in}}$ and $p_{i,\text{out}}$; and (b) the corresponding placement in laboratory space obtained by the rigid placement map P_i.

30.3.3 Ports

A *port* is a distinguished interface chart on the boundary of that support. If

$$\Sigma_{i,\alpha} \subset \partial\Omega_i$$

is a marked boundary patch and $F_{i,\alpha}$ is an adapted frame attached to that patch, then

$$p_{i,\alpha} = (\Sigma_{i,\alpha}, F_{i,\alpha})$$

is a port of the element.

In the simplest case the ports are just the entrance and exit faces. Junctions, splitters, extraction devices, or ring closures introduce additional named interface charts, but they do not change the basic definition.

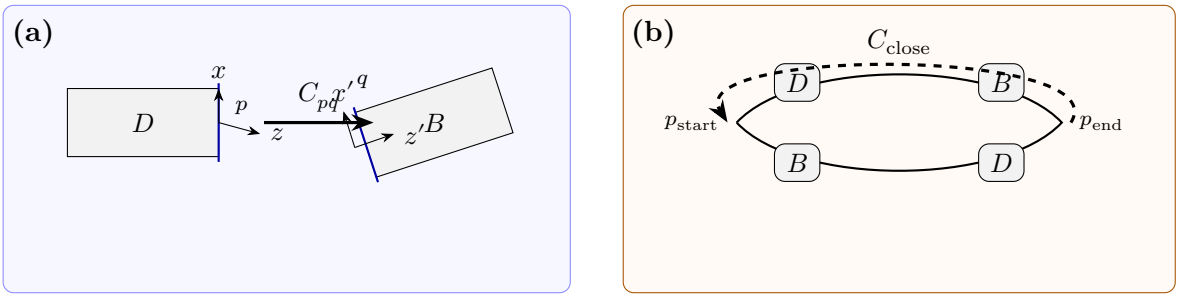


Figure 30.3: Ports, gluing, and closure: (a) port-to-port gluing via the transition map or patch transform C_{pq}; and (b) transition from port p to port q or closure around a ring, where the residual closure transform may satisfy $C_{\text{close}} \neq I$.

30.3.4 Transition maps and closure

When two local descriptions are joined, the relevant datum is a transition map between charts. On an overlap or a common interface one may write

$$g_{ij} = P_i^{-1} \circ P_j.$$

If the connection is localized to named ports p and q, the same datum can be stored as a rigid port-to-port patch transform C_{pq}. In that sense, the port notion is not an additional physical ingredient. It is the implementation-level representative of a chosen interface chart.

This language is particularly useful for rings. Transporting local frames around a full loop need not return one to the original frame with the identity transform. The residual closure transform is part of the geometry. This is the fibre-bundle point emphasized by Forest and Hirata: one should not force one global chart when the machine is naturally assembled from local ones [14].

31 Elements

This chapter documents element models whose geometry, field support, or runtime semantics require more than the generic placement conventions.

31.1 Analytic sector and rectangular bends

The current OPALX-native analytic `SBEND` and `RBEND` implementation is a field-based bend model with explicit body placement, explicit field-support extent, and first-order fringe optics. The implementation does not replace bend tracking by transfer matrices. Instead, the reference particle and the bunch are integrated through the magnetic field that is evaluated in the local bend chart.

31.1.1 Body extent and field-support extent

The **nominal body extent** is the placed hardware interval

$$z \in [0, L],$$

where L is the body length used for placement, body ports, and geometry export.

The **field-support extent** may extend beyond the body through entry and exit fringe regions,

$$z \in [-\ell_{\text{entry}},\, L + \ell_{\text{exit}}],$$

with

$$\ell_{\text{entry}} = \frac{h_{\text{gap}} F_{\text{int}}}{|\cos E_1|}, \qquad \ell_{\text{exit}} = \frac{h_{\text{gap}} F_{\text{int}}}{|\cos E_2|}.$$

Here h_{gap} is the half gap (`HGAP`), F_{int} is the fringe integral (`FINT`), and E_1, E_2 are the entrance and exit pole-face angles. If `HGAP` is not given, OPALX uses the fallback

$$h_{\text{gap}} = \frac{\text{GAP}}{2}.$$

The body extent drives placement and visualization. The field-support extent drives field evaluation, active-set occupancy, and timestep sanity checks. This is why OPALX can export both body markers (`ENTRY EDGE`, `EXIT EDGE`) and field-support markers (`FIELD BEGIN`, `FIELD END`) for the same bend.

31.1.2 Effective field length and bend normalization

The implemented fringe-support model uses a linear hard-edge ramp at both faces. The effective field length is therefore defined as

$$L_{\mathrm{eff}} = L_{\mathrm{ref}} + \frac{\ell_{\mathrm{entry}} + \ell_{\mathrm{exit}}}{2},$$

where L_{ref} is the body length measured along the design reference path.

For `SBEND`, the design reference-path body length equals the sector-bend body length,

$$L_{\mathrm{ref}}^{\mathrm{SBEND}} = L.$$

For `RBEND`, the design reference-path body length is the curved reference-path length between the rotated entrance and exit faces,

$$L_{\mathrm{ref}}^{\mathrm{RBEND}} = L_{\mathrm{arc}},$$

not the straight body chord. This distinction is required to normalize the analytic rectangular-bend field on the actual design orbit rather than on the mechanical chord.

`ANGLE` is the primary geometric input. The analytic dipole coefficient is normalized against the effective field length,

$$k_0 = \frac{\theta}{L_{\mathrm{eff}}},$$

with total bend angle θ. `K0` is treated only as a secondary consistency-check quantity.

Internally, OPALX stores the analytic bend strengths in normalized lattice form and converts them to physical Tesla coefficients only when a reference rigidity is available. If `DESIGNENERGY` is given, that kinetic energy is used together with the bunch species mass and charge. Otherwise the scaling is taken from the bunch reference momentum. This avoids tying bend tracking strength to the parser-global quantity P_0.

31.1.3 Longitudinal field profile

The body field is multiplied by a longitudinal scale function $\lambda(z)$ defined by

$$\lambda(z) = \begin{cases} 0, & z < -\ell_{\mathrm{entry}}, \\ \dfrac{z + \ell_{\mathrm{entry}}}{\ell_{\mathrm{entry}}}, & -\ell_{\mathrm{entry}} \le z \le 0, \\ 1, & 0 \le z \le L, \\ \dfrac{L + \ell_{\mathrm{exit}} - z}{\ell_{\mathrm{exit}}}, & L \le z \le L + \ell_{\mathrm{exit}}, \\ 0, & z > L + \ell_{\mathrm{exit}}. \end{cases}$$

This is a field-support model, not a body-length remapping. Particles begin to see the bend field at `FIELD BEGIN`, not necessarily at `ENTRY EDGE`.

31.1.4 First-order fringe optics

In addition to the longitudinal field ramp, the implementation includes a first-order edge-focusing model. With signed curvature

$$h = \frac{\theta}{L_{\text{eff}}},$$

the horizontal hard-edge relation is

$$x'_{\text{out}} = x'_{\text{in}} + h \tan(E)\, x.$$

The vertical edge focusing includes the standard fringe correction

$$\psi = h\, h_{\text{gap}}\, F_{\text{int}} \frac{1 + \sin^2(E)}{\cos(E)},$$

so the implemented vertical law is

$$y'_{\text{out}} = y'_{\text{in}} - h \tan(E - \psi)\, y.$$

In OPALX these optics effects are realized through an equivalent distributed fringe-field correction across the entry and exit support regions. The model is therefore field-based and first-order accurate in the edge optics, but it is not yet a full three-dimensional fringe-field model.

31.1.5 Sliced Bend Model

The current analytic `SBEND` and `RBEND` field model is already expressed in the local bend chart and is used consistently for the reference particle. The remaining many-particle tracking problem is geometric rather than magnetic: a quadrupole can be tracked in one rigid local frame, while a bend requires a local chart whose tangent direction rotates along the design path. The planned many-particle bend tracker therefore replaces a single rigid bend frame by a set of short rigid **tracking slices**.

The guiding idea is to make bend tracking follow the same Kokkos execution pattern as a straight multipole:

- transform the particle container into one local frame,
- evaluate the field locally with a Kokkos kernel,
- transform back.

For a quadrupole this is done once. For a bend it is done slice by slice along the reference path. The field law stays the analytic bend field described above; only the particle-level geometric approximation is changed.

31.1.5.1 Slice geometry

Let s denote the longitudinal coordinate along the bend design reference path. The reference path is split into N slices with intervals

$$I_j = [s_j, s_{j+1}], \qquad j = 0, \dots, N-1,$$

with slice length

$$\Delta s_j = s_{j+1} - s_j.$$

Each slice is represented by a rigid local frame attached at the slice center

$$s_{j+\frac{1}{2}} = \frac{s_j + s_{j+1}}{2}.$$

Let $\mathbf{r}_{\mathrm{ref}}(s)$ be the reference-path position and $(\mathbf{e}_x(s), \mathbf{e}_y(s), \mathbf{e}_s(s))$ the local right-handed bend basis consisting of horizontal transverse, vertical transverse, and tangent directions. The slice frame is then

$$\mathcal{F}_j = \left(\mathbf{r}_{\mathrm{ref}}(s_{j+\frac{1}{2}}), \mathbf{e}_x(s_{j+\frac{1}{2}}), \mathbf{e}_y(s_{j+\frac{1}{2}}), \mathbf{e}_s(s_{j+\frac{1}{2}})\right).$$

Within the slice, particle coordinates are approximated by the straight local chart

$$\mathbf{r} \approx \mathbf{r}_{\mathrm{ref}}(s_{j+\frac{1}{2}}) + x\,\mathbf{e}_x(s_{j+\frac{1}{2}}) + y\,\mathbf{e}_y(s_{j+\frac{1}{2}}) + z\,\mathbf{e}_s(s_{j+\frac{1}{2}}),$$

with $z \in [-\Delta s_j/2, \Delta s_j/2]$. The approximation becomes exact in the limit $\max_j \Delta s_j \to 0$.

This means that many-particle bend tracking is planned as a **slice-wise rigid tangent-frame approximation** to the curved local chart. It is not a transfer matrix model and it is not a replacement of the bend field by a different straight-element field law.

31.1.5.2 Body, entry-fringe, and exit-fringe slices

The slice construction must respect the field-support extent rather than only the placed body extent. In the local bend chart the three tracking regions are

$$[-\ell_{\mathrm{entry}}, 0], \qquad [0, L_{\mathrm{ref}}], \qquad [L_{\mathrm{ref}}, L_{\mathrm{ref}} + \ell_{\mathrm{exit}}],$$

corresponding to:

1. entry fringe,
2. body,
3. exit fringe.

The body slices carry the full field scale $\lambda = 1$. The fringe slices carry the longitudinal ramp already defined in the analytic bend model. Using the local slice coordinate z, the slice-local field scale is

$$\lambda_j(z) = \begin{cases} \dfrac{z - z_{j,\mathrm{begin}}}{z_{j,\mathrm{end}} - z_{j,\mathrm{begin}}}, & \text{entry fringe slice,} \\ 1, & \text{body slice,} \\ \dfrac{z_{j,\mathrm{end}} - z}{z_{j,\mathrm{end}} - z_{j,\mathrm{begin}}}, & \text{exit fringe slice,} \end{cases}$$

with the obvious clipping to $[0, 1]$.

This preserves the currently implemented body-vs-field-support split:

- body geometry, ports, and visualization are still tied to the placed body,
- field activity and many-particle tracking support extend over the fringe intervals.

31.1.5.3 Pole-face rotations and slice support lengths

The pole-face angles E_1 and E_2 do **not** rotate the slice frames. Each slice frame remains tangent to the design path. The pole-face angles only determine:

1. how far the entry and exit fringe regions extend in s,
2. what first-order edge focusing strength is distributed over those fringe slices.

The support lengths remain

$$\ell_{\mathrm{entry}} = \frac{h_{\mathrm{gap}} F_{\mathrm{int}}}{|\cos E_1|}, \qquad \ell_{\mathrm{exit}} = \frac{h_{\mathrm{gap}} F_{\mathrm{int}}}{|\cos E_2|}.$$

Therefore the slice builder must explicitly create entry-fringe slices over $[-\ell_{\mathrm{entry}}, 0]$ and exit-fringe slices over $[L_{\mathrm{ref}}, L_{\mathrm{ref}} + \ell_{\mathrm{exit}}]$. A uniform body slicing alone is not sufficient.

31.1.5.4 `HGAP` and fringe optics inside slices

The half gap h_{gap} remains a physical parameter of the field support and edge optics, not a geometrical rotation parameter. It enters the sliced model in exactly the same two ways as in the unsliced analytic model:

1. through the fringe support lengths ℓ_{entry} and ℓ_{exit},
2. through the vertical fringe correction

$$\psi = h\, h_{\mathrm{gap}}\, F_{\mathrm{int}} \frac{1 + \sin^2(E)}{\cos(E)}.$$

With signed curvature

$$h = \frac{\theta}{L_{\text{eff}}},$$

the edge strengths are

$$k_{x,\text{entry}} = h\tan(E_1), \qquad k_{x,\text{exit}} = h\tan(E_2),$$

and

$$k_{y,\text{entry}} = -h\tan(E_1 - \psi_1), \qquad k_{y,\text{exit}} = -h\tan(E_2 - \psi_2),$$

with

$$\psi_1 = h\,h_{\text{gap}}\,F_{\text{int}}\frac{1+\sin^2(E_1)}{\cos(E_1)}, \qquad \psi_2 = h\,h_{\text{gap}}\,F_{\text{int}}\frac{1+\sin^2(E_2)}{\cos(E_2)}.$$

In the sliced model these thin-lens strengths are distributed over the fringe slice lengths. A natural first-order choice is

$$G_{x,\text{entry}} = \frac{k_{x,\text{entry}}}{\ell_{\text{entry}}}, \qquad G_{y,\text{entry}} = \frac{k_{y,\text{entry}}}{\ell_{\text{entry}}},$$

and likewise for the exit face. The slice-local field correction then acts as an equivalent normal quadrupole contribution over the fringe slices,

$$B_x^{\text{edge}} = G_y\,y, \qquad B_y^{\text{edge}} = G_x\,x,$$

so that integration through the slice stack reproduces the first-order edge optics in the limit of sufficient slicing.

31.1.5.5 `SBEND` versus `RBEND`

For `SBEND`, the body reference-path length equals the sector-bend body length,

$$L_{\text{ref}}^{\text{SBEND}} = L.$$

For `RBEND`, the slice construction must use the curved reference-path length between the rotated faces,

$$L_{\text{ref}}^{\text{RBEND}} = L_{\text{arc}},$$

not the mechanical body chord. This is the same distinction already required for the effective field length and dipole normalization, and it must also be used consistently when building the body and fringe slice intervals.

31.1.5.6 Numerical interpretation

The sliced bend model is therefore a controlled geometric approximation for many-particle tracking:

- the **field law** remains the analytic OPALX bend field,
- the **curved chart** is approximated by a sequence of short rigid tangent frames,
- the approximation error decreases as the slice size decreases,
- the OpenMP/Kokkos execution model remains the same as for straight local elements.

This makes `SBEND` and `RBEND` many-particle tracking structurally consistent with the straight-element tracker while preserving the bend-specific body placement, field-support extent, pole-face angles, and `HGAP`/`FINT` fringe semantics.

31.1.6 Code-to-code and analytic comparisons

The current implementation has been benchmarked against Bmad for the two simplest no-fringe constant-field cases:

- a proton `SBEND` with kinetic energy `590 MeV`, body length `1 m`, and bend angle `45^\circ`,
- an electron `RBEND` with kinetic energy `1 GeV`, rectangular body length `1 m`, and bend angle `45^\circ`.

For the `SBEND` benchmark, the OPALX trajectory has also been compared with the analytic circular-orbit solution. The comparison plot below shows the current code-to-code agreement. The `SBEND` benchmark agrees with the analytic endpoint at the exported-design-path level to within approximately 2×10^{-5} m for `DT = 10^{-10}` and improves below 10^{-6} m for smaller time steps. The `RBEND` benchmark now agrees with Bmad to within the displayed plotting resolution for the simple constant-field case.

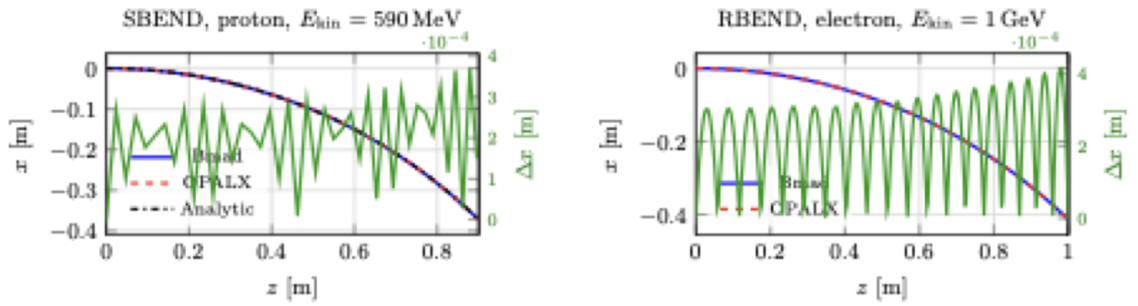


Figure 31.1: Comparison of the current OPALX `SBEND` and `RBEND` benchmarks against Bmad. The `SBEND` panel also includes the analytic circular-orbit solution. The right axis in each panel shows $\Delta x = x_{\mathrm{OPALX}} - x_{\mathrm{Bmad}}$.

The timestep study for the simple `SBEND` case is shown below. The reported metric is the discrete path-wise L^2 error between the OPALX design path and the analytic reference

curve, computed over the full bend interval. The dashed reference line shows the expected second-order decay $\propto \Delta t^2$. The measured convergence is second order over the practical range until it reaches the numerical floor of the current benchmark setup.

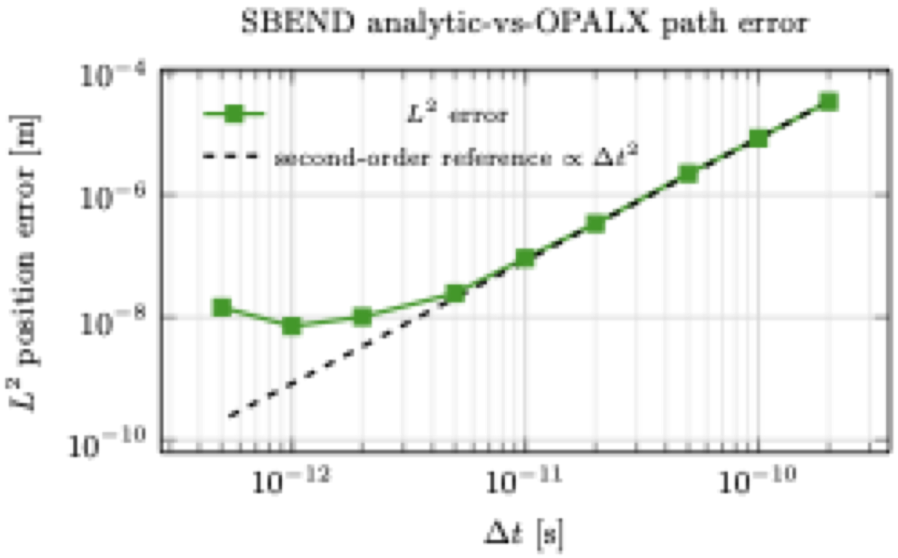


Figure 31.2: `SBEND` timestep study using the path-wise L^2 error between OPALX and the analytic circular-orbit solution. The dashed line shows the theoretical second-order reference decay.

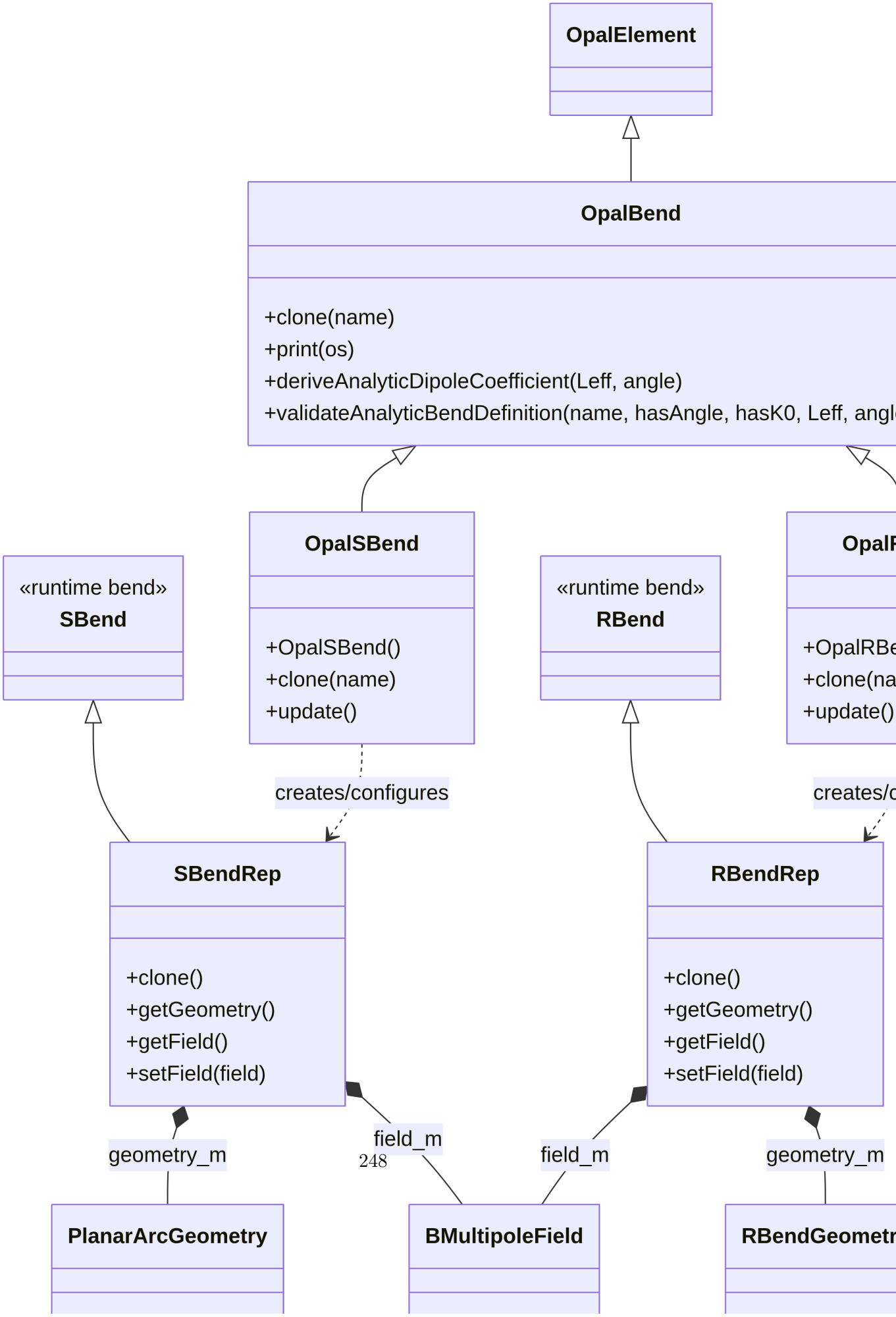
31.1.7 `SBEND` and `RBEND` class structure

The current implementation splits bend handling into a parser-facing object layer and a runtime representation layer. `OpalSBend` and `OpalRBend` inherit the common bend attributes and update logic from `OpalBend`, then instantiate and configure `SBendRep` and `RBendRep` respectively.

The key distinction is that `OpalSBend` and `OpalRBend` are the input and update front ends, while `SBendRep` and `RBendRep` own the actual geometry and local normalized multipole field used during tracking.

31.1.8 Example: C-Type Chicane

A useful multi-element benchmark for explicit bend placement is a symmetric C-type four-dipole chicane. The example below is deliberately kept at the analytic hard-edge `SBEND` level: no field maps are used, the four bend bodies are posed explicitly with `X`, `Y`, `Z`, `THETA`, `PHI`, and `PSI`, and the output is checked through the exported element positions and a small off-momentum R_{56} probe.


OpalElement
OpalBend
+clone(name)
+print(os)
+deriveAnalyticDipoleCoefficient(Leff, angle)
+validateAnalyticBendDefinition(name, hasAngle, hasK0, Leff, angl
OpalSBend
+OpalSBend()
+clone(name)
+update()
OpalR
+OpalRBe
+clone(na
+update()
«runtime bend»
SBend
«runtime bend»
RBend
creates/configures
creates/c
SBendRep
+clone()
+getGeometry()
+getField()
+setField(field)
RBendRep
+clone()
+getGeometry()
+getField()
+setField(field)
geometry_m
field_m
field_m
geometry_m
PlanarArcGeometry
BMultipoleField
RBendGeometr

The complete runnable input and local post-processing scripts are stored with this chapter:

- test-chicane-1.in
- test-chicane-1_distribution.txt
- check-r56.py

The benchmark uses four equal hard-edge sector bends with physical survey bend signs

$$(+\theta,\ -\theta,\ -\theta,\ +\theta),$$

magnetic arc length L_b, outer drift D between bends 1–2 and 3–4, and central drift CS between bends 2–3. This is the standard small-angle four-dipole bunch-compressor model used in text book treatments of C-type chicanes [21], [22]. The numerical constants used here are

$$L_b = 1.0 \text{ m}, \qquad \theta = 0.1 \text{ rad}, \qquad D = 1.5 \text{ m}, \qquad C = 2.0 \text{ m}.$$

For placement, the reference entrance is centered at $(X, Y, Z) = (0, 0, 0)$, with tangent along global $+Z$. In the linear approximation we get

$$\rho = \frac{L_b}{\theta},$$

and the other defining constants are

$$\begin{aligned}
\Delta x_b &= \rho(1 - \cos\theta), & \Delta z_b &= \rho \sin\theta, \\
\Delta x_{b/2} &= \rho\left(1 - \cos\frac{\theta}{2}\right), & \Delta z_{b/2} &= \rho \sin\frac{\theta}{2}, \\
\Delta x_{r/2} &= \rho\left(\cos\frac{\theta}{2} - \cos\theta\right), & \Delta z_{r/2} &= \rho\left(\sin\theta - \sin\frac{\theta}{2}\right), \\
\Delta x_D &= D \sin\theta, & \Delta z_D &= D\cos\theta.
\end{aligned}$$

The corresponding names in the imputfile are `SB_DX`, `SB_DZ`, `SBH_DX`, `SBH_DZ`, `RBH_DX`, `RBH_DZ`, `DR_DX`, and `DR_DZ`. It then assembles the actual element body origins and monitor locations as `B1_X`, `B1_Z`, …, `MEXIT_X`, `MEXIT_Z`. This is important because OPALX places an analytic bend by its body origin, not by an entrance edge (BODY pose).

OPALX's current `SBEND` angle sign is opposite to the positive-X survey convention used in this example. Therefore the first and fourth bends use `ANGLE = -TH`, while the second and third use `ANGLE = TH`. The element placement remains survey-positive in the named `X` and `Z` variables.

The resulting design survey is:

point	Z [m]	X [m]
B1 entry	0.000000000	0.000000000
B1 exit / D12 begin	0.998334166	0.049958347
B2 entry	2.490840414	0.199708472

point	Z [m]	X [m]
B2 exit / D23 begin	3.489174581	0.249666819
B3 entry	5.489174581	0.249666819
B3 exit / D34 begin	6.487508747	0.199708472
B4 entry	7.980014995	0.049958347
B4 exit / MEXIT	8.978349162	0.000000000

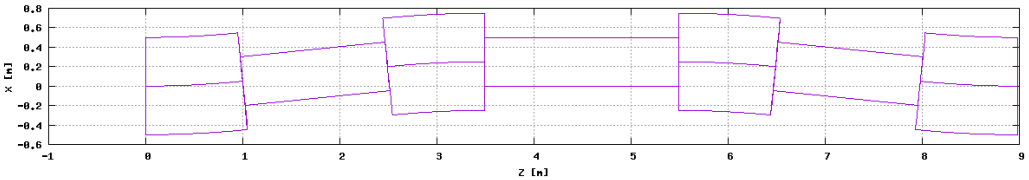


Figure 31.4: Projected OPALX element-position export for the analytic four-`SBEND` C-type chicane. The projection was made from the OPALX-generated `test-chicane-1_ElementPositions.py` script using the X-Z plane.

This test does a momentum scan with 5 particles. The launched particles have

$$\delta = \frac{p_z - p_0}{p_0} \in \{-2\times10^{-3},\, -10^{-3},\, 0,\, +10^{-3},\, +2\times10^{-3}\},$$

with $p_0 = 1957.9509$ in OPALX normalized $(\beta\gamma)$ units. The small-angle formula for the chicane is

$$R_{56} \simeq -\theta^2\left(\frac{4}{3}L_b + 2D\right) = -4.3334\times10^{-2}\ \text{m}.$$

For this monitor convention the local OPALX coordinate has the opposite sign to the usual transport R_{56} convention,

$$R_{56} = -\frac{dz_{\text{OPALX, exit}}}{d\delta}.$$

With `DT = 1.0e-11 s`, the current run gives

$$R_{56} = -4.3114\times10^{-2}\ \text{m},$$

about

$$2.19 \times 10^{-4} \text{ m}$$

difference from the analytic model.

Run the example from its local directory with:

```
cd physics/elements/examples/test-chicane-1
/path/to/opalx test-chicane-1.in --info 2
python3 dasta/test-chicane-1_ElementPositions.py --project-to-plane --normal 0 1 0
python3 check-r56.py
```

31.2 Multipole family

The OPALX `MULTIPOLE` implementation is the common straight-magnet path for normal and skew magnetic multipole expansions. The present `QUADRUPOLE` front-end is a convenience wrapper on top of that common path. A dedicated `SEXTUPOLE` parser-side wrapper is not present in this checkout; sextupole content is therefore expressed through the generic `MULTIPOLE` coefficient arrays.

31.2.1 Field expansion and conventions

The underlying magnetic field class `BMultipoleField` (`src/Fields/BMultipoleField.h`) uses the complex transverse expansion

$$B_y + iB_x = \sum_{m=0}^{N-1} (B_m + iA_m)(x + iy)^m,$$

where:

- $m = 0$ is the dipole term,
- $m = 1$ is the quadrupole term,
- $m = 2$ is the sextupole term,
- B_m are normal coefficients,
- A_m are skew coefficients.

In the abstract multipole interface `Multipole` (`src/AbsBeamline/Multipole.h`), the same indexing is used explicitly: `n = 0` corresponds to dipole, `n = 1` to quadrupole, `n = 2` to sextupole, and so on.

For the first three nontrivial components, the transverse magnetic field is:

$$\text{normal quadrupole:} \qquad B_x = B_1\, y, \qquad B_y = B_1\, x,$$

$$\text{skew quadrupole:} \qquad B_x = -A_1\, x, \qquad B_y = A_1\, y,$$

$$\text{normal sextupole:} \qquad B_x = 2B_2\,xy, \qquad B_y = B_2(x^2 - y^2),$$

$$\text{skew sextupole:} \qquad B_x = -A_2(x^2 - y^2), \qquad B_y = 2A_2\,xy.$$

These formulas match the host-side field evaluation in `src/AbsBeamline/Multipole.cpp`.

31.2.2 `MULTIPOLE`

The parser-side element `OpalMultipole` (`src/Elements/OpalMultipole.cpp`) accepts the coefficient arrays:

- `KN`: normal normalized strengths
- `DKN`: normal strength errors
- `KS`: skew normalized strengths
- `DKS`: skew strength errors

The element semantics are:

- if `L != 0`, the strengths are interpreted per unit length
- if `L == 0`, they are interpreted as integrated strengths

At update time, `OpalMultipole` creates or updates a `MultipoleRep` (`src/BeamlineCore/MultipoleRep.h`), copies the element length into its straight geometry, and fills a `BMultipoleField` plus the runtime coefficient storage in the shared `Multipole` (`src/AbsBeamline/Multipole.h`) base.

Internally the generic mapping is:

$$\texttt{KN}[0] \to \text{dipole}, \qquad \texttt{KN}[1] \to \text{quadrupole}, \qquad \texttt{KN}[2] \to \text{sextupole},$$

and likewise for `KS`.

31.2.3 `QUADRUPOLE`

The parser-side `OpalQuadrupole` (`src/Elements/OpalQuadrupole.cpp`) is not a separate runtime field model. It is a convenience wrapper that instantiates the same `MultipoleRep` (`src/BeamlineCore/MultipoleRep.h`) used by `MULTIPOLE`, but exposes scalar quadrupole attributes instead of full coefficient arrays:

- `K1`, `DK1`: normal quadrupole strength and error
- `K1S`, `DK1S`: skew quadrupole strength and error

The implementation writes those values into the multipole representation as:

$$\texttt{KN} = \{0, K_1\}, \qquad \texttt{KS} = \{0, K_{1s}\}.$$

So `QUADRUPOLE` is best understood as a named restricted view of the generic multipole interface with only the $m = 1$ term exposed.

31.2.4 `SEXTUPOLE`

There is currently no dedicated `OpalSextupole` front-end class in this checkout. The sextupole physics is nevertheless available through the generic `MULTIPOLE` representation by populating the order-2 coefficients:

$$\texttt{KN} = \{0, 0, K_2\}, \qquad \texttt{KS} = \{0, 0, K_{2s}\}.$$

That means the sextupole is currently a *configuration of `MULTIPOLE`*, not a distinct parser-side wrapper parallel to `QUADRUPOLE`.

For a nonzero body length L, K_2 and K_{2s} are interpreted as distributed sextupole strengths; for $L = 0$, they are integrated strengths, consistent with the generic `MULTIPOLE` element description.

31.2.5 Runtime geometry and current implementation status

The runtime representation `MultipoleRep` (`src/BeamlineCore/MultipoleRep.h`) inherits from `Multipole` (`src/AbsBeamline/Multipole.h`) and owns:

- a `StraightGeometry`
- a `BMultipoleField`

So the current multipole family described here is a straight-element model with field support over

$$z \in [0, L].$$

One important current detail from `src/AbsBeamline/Multipole.cpp` is that the device-side `computeField()` kernel still contains explicit special cases only for dipole and quadrupole, while the host-side `computeFieldHost()` implements sextupole and higher orders as well. For the manual this means:

- the interface is generic up to higher order,
- the host/orbit-threader path includes sextupole formulas,
- the current device kernel is still more limited than the abstract interface.

That distinction matters when documenting present implementation status versus intended generality.

31.2.6 Class structure

The current parser-side and runtime-side structure is shown below.

In other words, `QUADRUPOLE` is presently a thin parser-side specialization of the generic multipole path, and sextupole behavior is likewise represented through `MultipoleRep`, but without a dedicated `OpalSextupole` wrapper yet.

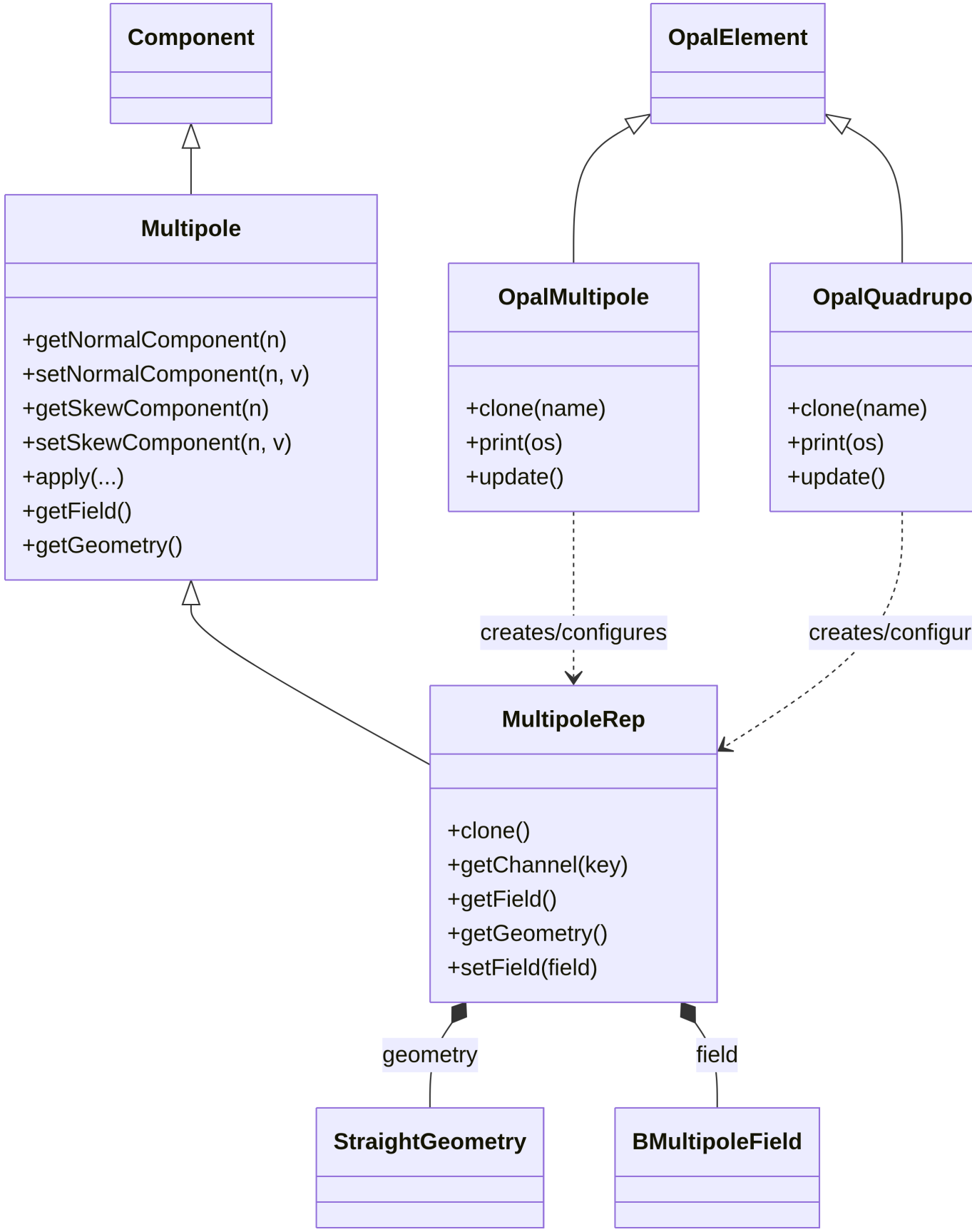


Figure 31.5: Parser-side and runtime-side class structure for the current `MULTIPOLE` and `QUADRUPOLE` implementation. Sextupole content currently enters through the generic `MULTIPOLE` coefficient path.

32 Collective Effects

Collective effects couple the tracked particles through self-consistent fields or material interactions. The current tracker assembles space-charge fields between its two half drifts, then transforms the resulting fields back into the reference frame before the momentum kick.

This chapter is intentionally marked **planned** while each implemented model is checked against the source and its verification cases. It will be expanded into separate, source-backed sections for:

- Electrostatic space charge and Green-function choices.
- Open and periodic boundary conditions.
- Wake fields.
- Particle-matter interactions.
- Mesh convergence and decomposition constraints.

For a complete source-verified example today, see the Beam-Beam manufactured solution.

33 Field Solver Physics

This chapter is the physics-level counterpart to the Field Solver user guide. The user guide documents input choices and current runtime behavior; this chapter will derive the model, state its approximations, and collect numerical validation.

> **Documentation skeleton**
>
> The headings below define the intended scope. They deliberately contain only short writing prompts so that equations and claims can be added after they are checked against the implementation and benchmark cases.

33.1 Scope and notation

Define the charge, field, coordinate, unit, and sign conventions used throughout the chapter. State which parts apply to all solvers and which are specific to space charge.

33.2 Governing electrostatic model

State the quasi-static approximation and the physical problem solved in the selected bunch frame.

33.2.1 Poisson equation

Introduce the source term, potential, electric field, and boundary data.

33.2.2 Self-field force

Connect the solved fields to the force applied by the tracker, including any frame transformation required before the particle push.

33.3 Particle-mesh discretization

Describe the complete particle-in-cell cycle and identify the discrete quantities that live on particles and on the mesh.

33.3.1 Charge deposition

Specify the particle shape, weighting rule, normalization, and treatment of ghost cells.

33.3.2 Mesh solve

Define the discrete Poisson operator or convolution used by each solver family.

33.3.3 Field reconstruction and interpolation

Explain how the electric field is reconstructed and interpolated back to particle positions.

33.4 Frames and relativistic transformations

Derive the transformations between the laboratory frame, mean rest frame, and bin-local frames. State where approximations enter.

33.5 Boundary conditions and Green functions

Define each boundary-value problem independently of its input syntax.

33.5.1 Periodic boundaries

Describe the periodic domain, compatibility conditions, and zero-mode handling.

33.5.2 Open boundaries

Derive the free-space Green-function convolution and doubled-domain method.

33.5.3 Dirichlet and image-charge boundaries

Explain explicit image-charge and shifted-Green constructions, including the geometries and assumptions for which they are valid.

33.6 Solver formulations

Relate the mathematical problem to the available numerical backends.

33.6.1 Periodic FFT solver

Document the spectral formulation, differentiation, normalization, and parallel decomposition.

33.6.2 Hockney open-boundary solver

Document domain doubling, kernel sampling, padding, and extraction of the physical-domain result.

33.6.3 Iterative solvers

Reserve this subsection for the operator, convergence criterion, preconditioning, and supported boundary conditions once an iterative backend is production-ready.

33.7 Binned rest-frame space charge

Derive the longitudinal binning approximation, bin-frame transforms, per-bin solve, and accumulation of the resulting fields.

33.8 Emission and conducting boundaries

Describe how emission time, cathode geometry, image charges, and activation of space charge interact.

33.9 Numerical accuracy and convergence

Collect the error sources that should be varied in convergence studies.

33.9.1 Mesh resolution

Define spatial refinement studies and expected convergence observables.

33.9.2 Particle noise and deposition order

Separate sampling noise from mesh and solver error.

33.9.3 Time-step coupling

Explain how field-solve frequency, particle stepping, and emission updates couple to the spatial solve.

33.10 Domain decomposition and load balancing

Explain which mathematical data are repartitioned and why the field, particle, and solver state must be refreshed in a defined order.

33.11 Current implementation architecture

The following diagram is retained as TikZ source for PDF output. HTML uses a small generated PNG of the same source so the page does not depend on a browser-side TikZ renderer.

33.12 Verification and benchmarks

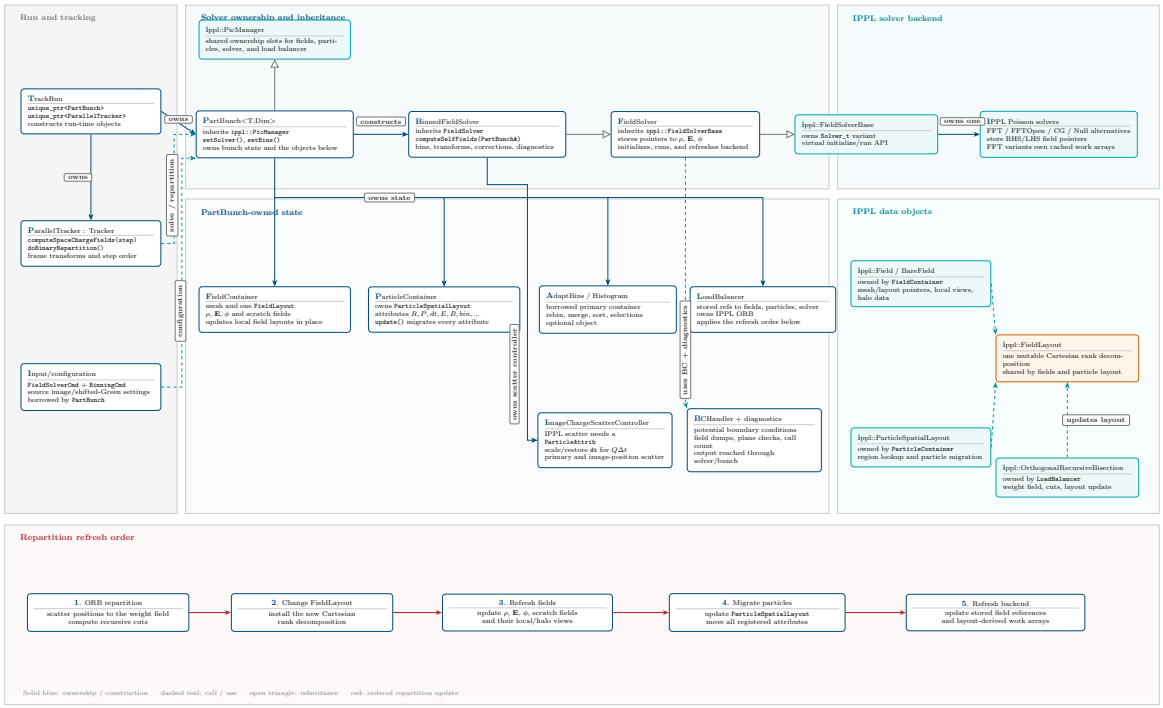
List analytic solutions, manufactured solutions, regression cases, conserved quantities, and cross-code comparisons required for each solver mode.

33.13 Assumptions and known limitations

Maintain a concise list of model assumptions, unsupported combinations, and known numerical limitations, with links to reproducible investigations.

Current OPALX space-charge class diagram

Production `master`: ownership, calls, stored references, and the separate repartition refresh order



Source trace: `ParallelTracker`, `PartBunch`, `FieldContainer`, `BinnedFieldSolver`, `FieldSolver`, `LoadBalancer`, and the referenced IPPL manager, field, particle-layout, Poisson, and ORB classes.

Figure 33.1: Current OPALX space-charge class relationships and repartition refresh order. The editable source is `figures/current-space-charge-class-diagram.tex`.

34 Beam-Beam

34.1 Theory of colliding beams in 3D

The `BEAMBEAM` element models the electromagnetic interaction of two counter-propagating bunches near an interaction point (IP). The validation problem used here isolates the field solve from the transport dynamics: the two bunches are represented in the solver frame as a static electrostatic pair of identical three-dimensional Gaussian charge densities. This is the correct manufactured solution for checking the density deposition, open-boundary Poisson solve, and electric-field reconstruction used by the beam-beam diagnostics.

Let the solver-frame coordinate be $\mathbf{r} = (x, y, z)$. The OPALX bunch is centered at $\mathbf{c}_1$, and the copied collision partner is obtained by mirror reflection about the local IP position z_{IP},

$$\mathbf{c}_2 = \begin{pmatrix} c_{1x} \\ c_{1y} \\ 2z_{\mathrm{IP}} - c_{1z} \end{pmatrix}.$$

For the static validation case the total charge density is

$$\rho(\mathbf{r}) = \sum_{a=1}^{2} \frac{Q}{(2\pi)^{3/2}\sigma^3} \exp\left[-\frac{|\mathbf{r} - \mathbf{c}_a|^2}{2\sigma^2}\right],$$

where Q is the charge of one bunch and σ is the common rms size in all three spatial directions. OPALX solves

$$\nabla^2\phi(\mathbf{r}) = -\frac{\rho(\mathbf{r})}{\epsilon_0}, \qquad \mathbf{E}(\mathbf{r}) = -\nabla\phi(\mathbf{r}),$$

with open boundary conditions. In the validation below no Lorentz transformation is applied: the comparison is deliberately performed in the solver frame against the electrostatic manufactured solution. Beam-energy parameters still appear in the input file because the normal OPALX tracking driver requires a reference beam, but the analytic field comparison is not a laboratory-frame boosted-field calculation.

34.2 Analytic solution

For a single spherical Gaussian bunch centered at $\mathbf{c}$, define

$$\mathbf{R} = \mathbf{r} - \mathbf{c}, \qquad R = |\mathbf{R}|, \qquad u = \frac{R}{\sqrt{2}\sigma}.$$

Convolving this density with the free-space Green function of Poisson's equation gives the standard electrostatic Gaussian potential [23],

$$\phi_G(\mathbf{r};\mathbf{c},Q,\sigma) = \frac{Q}{4\pi\epsilon_0}\frac{\operatorname{erf}(u)}{R},$$

with the finite limiting value

$$\phi_G(\mathbf{c};\mathbf{c},Q,\sigma) = \frac{Q}{4\pi\epsilon_0}\frac{\sqrt{2/\pi}}{\sigma}.$$

The electric field is

$$\mathbf{E}_G(\mathbf{r};\mathbf{c},Q,\sigma) = \frac{Q}{4\pi\epsilon_0}\frac{\operatorname{erf}(u) - \sqrt{\frac{2}{\pi}}\frac{R}{\sigma}\exp\!\left(-\frac{R^2}{2\sigma^2}\right)}{R^3}\mathbf{R},$$

where the field vanishes at $R = 0$ by symmetry.

The analytic two-bunch manufactured solution is the linear superposition

$$\phi(\mathbf{r}) = \phi_G(\mathbf{r};\mathbf{c}_1,Q,\sigma) + \phi_G(\mathbf{r};\mathbf{c}_2,Q,\sigma),$$

and

$$\mathbf{E}(\mathbf{r}) = \mathbf{E}_G(\mathbf{r};\mathbf{c}_1,Q,\sigma) + \mathbf{E}_G(\mathbf{r};\mathbf{c}_2,Q,\sigma).$$

For equal bunch charges and symmetric placement, the exact field at the IP is zero:

$$\mathbf{E}(0,0,z_{\mathrm{IP}}) = \mathbf{0}.$$

The useful scalar reference value is instead the longitudinal field at the center of one bunch due to the other bunch. With separation $d = |\mathbf{c}_2 - \mathbf{c}_1|$, this is

$$E_z(\mathbf{c}_1) = \frac{Q}{4\pi\epsilon_0}\frac{\operatorname{erf}\!\left(\frac{d}{\sqrt{2}\sigma}\right) - \sqrt{\frac{2}{\pi}}\frac{d}{\sigma}\exp\!\left(-\frac{d^2}{2\sigma^2}\right)}{d^2}\operatorname{sgn}(c_{1z} - c_{2z}).$$

34.3 One-volt-per-meter test case

A special manufactured solution reads

$$\sigma = 1\,\mathrm{mm}, \qquad d = 10\,\mathrm{mm}, \qquad |Q| = 1.112650055 \times 10^{-14}\,\mathrm{C}.$$

For an electron bunch, $Q < 0$. With these parameters the field at the center of one bunch due to the copied partner is approximately

$$|E_z(\mathbf{c}_1)| = 1.0\,\mathrm{V/m}.$$

At the IP the exact field remains zero by symmetry. The potential at the IP is approximately

$$\phi(0,0,z_{\mathrm{IP}}) = -3.9999977 \times 10^{-2}\,\mathrm{V}$$

for the electron-sign convention used in the OPALX input. This value is not a contradiction of the zero-field result: the electric field is the gradient of the scalar potential, so the equal and opposite gradients from the two bunches cancel at the IP, while the two scalar potential contributions add.

The OPALX run below uses a 32^3 open field-solver mesh and $8N^3 = 262144$ macroparticles. The `COPY=TRUE` option instructs the BeamBeam element to build the mirrored bunch in the interaction window.

```
OPTION, PSDUMPFREQ    = -1;
OPTION, STATDUMPFREQ  = 1;
OPTION, BOUNDPDESTROY = 10;
OPTION, AUTOPHASE     = 4;
OPTION, VERSION       = 900;

TITLE, STRING="BeamBeam static Gaussian 1 V/m analytic validation ";

REAL N                 = 32;
REAL n_particles       = 8*N*N*N;
REAL beam_bunch_charge = 1.112650055e-14;

REAL Edes  = 0.25;
REAL gamma = (Edes+EMASS)/EMASS;
REAL beta  = sqrt(1-(1/gamma^2));
REAL P0    = gamma*beta*EMASS;

DR1: DRIFT,    L = 0.008425397, ELEMEDGE = 0.0;
BB1: BEAMBEAM, L = 0.02, ELEMEDGE = 0.008425397,
               VISUALIZE=FALSE, COPY=TRUE;
DR2: DRIFT,    L = 0.031574603, ELEMEDGE = 0.028425397;

myLine: Line = (DR1,BB1,DR2);

FS1: FIELDSOLVER, TYPE=OPEN,
    NX=N, NY=N, NZ=N,
    PARFFTX=TRUE, PARFFTY=TRUE, PARFFTZ=FALSE,
    BCFFTX=OPEN, BCFFTY=OPEN, BCFFTZ=OPEN,
    BBOXINCR = 1, GREENSF = STANDARD;

Dist1: DISTRIBUTION, TYPE=GAUSS,
    SIGMAX=1e-3,  SIGMAY=1e-3,  SIGMAZ=1e-3,
    SIGMAPX=1e-12, SIGMAPY=1e-12, SIGMAPZ=1e-12,
    NPARTDIST = n_particles;

ES1: EMISSIONSOURCE, DISTRIBUTION = Dist1;
mySources: EMISSIONSOURCELIST = (ES1);
```

```
BEAM1: BEAM, PARTICLE = ELECTRON, pc = P0, NALLOC = n_particles,
       SOURCES = mySources, BCHARGE = beam_bunch_charge, CHARGE = -1;

TRACK, LINE = myLine, BEAM = BEAM1,
       MAXSTEPS = {4,15,3}, DT = {1e-11,3e-12,1e-11};
    RUN, METHOD = "PARALLEL", FIELDSOLVER = FS1;
ENDTRACK;
QUIT;
```

34.4 Results

The OPALX field diagnostics were compared directly to the analytic Gaussian pair on the same solver mesh. The active diagnostic snapshot used here has solver-frame centers

$$c_{1z} = 7.49479585743\,\mathrm{mm}, \qquad c_{2z} = 17.38792638917\,\mathrm{mm},$$

so the realized separation is

$$d_{\mathrm{OPALX}} = 9.89313053174\,\mathrm{mm}.$$

This is close to, but not exactly, the nominal 10 mm reference because the active BeamBeam window is entered on the tracker step sequence and the diagnostic samples the actual solver-frame geometry.

Quantity	Relative L_2 error	Maximum absolute error
ρ	5.221859×10^{-2}	$5.746384 \times 10^{-8}\,\mathrm{C/m^3}$
ϕ	3.042248×10^{-3}	$1.087451 \times 10^{-3}\,\mathrm{V}$
E_x	1.503024×10^{-2}	$8.676369 \times 10^{-1}\,\mathrm{V/m}$
E_y	1.582187×10^{-2}	$8.807832 \times 10^{-1}\,\mathrm{V/m}$
E_z	2.427369×10^{-2}	$1.031355\,\mathrm{V/m}$

At the nearest on-axis grid sample to the IP, the longitudinal field is

$$E_z^{\mathrm{analytic}} = 1.038298\,\mathrm{V/m}, \qquad E_z^{\mathrm{OPALX}} = 1.072345\,\mathrm{V/m}.$$

Interpolating the same on-axis samples to the mathematical IP gives the symmetry check

$$E_z^{\mathrm{analytic}}(z_{\mathrm{IP}}) \approx -5.88 \times 10^{-8}\,\mathrm{V/m}, \qquad E_z^{\mathrm{OPALX}}(z_{\mathrm{IP}}) \approx 3.77 \times 10^{-6}\,\mathrm{V/m}.$$

This confirms that the field at the interaction point is numerically consistent with the exact zero-field symmetry of the manufactured solution.

The remaining few-percent field differences are consistent with the stochastic macroparticle realization and the 32^3 mesh used in this diagnostic run. For a stricter regression test of the solver alone, the next step is to inject the analytic density directly on the mesh, bypassing particle sampling noise.

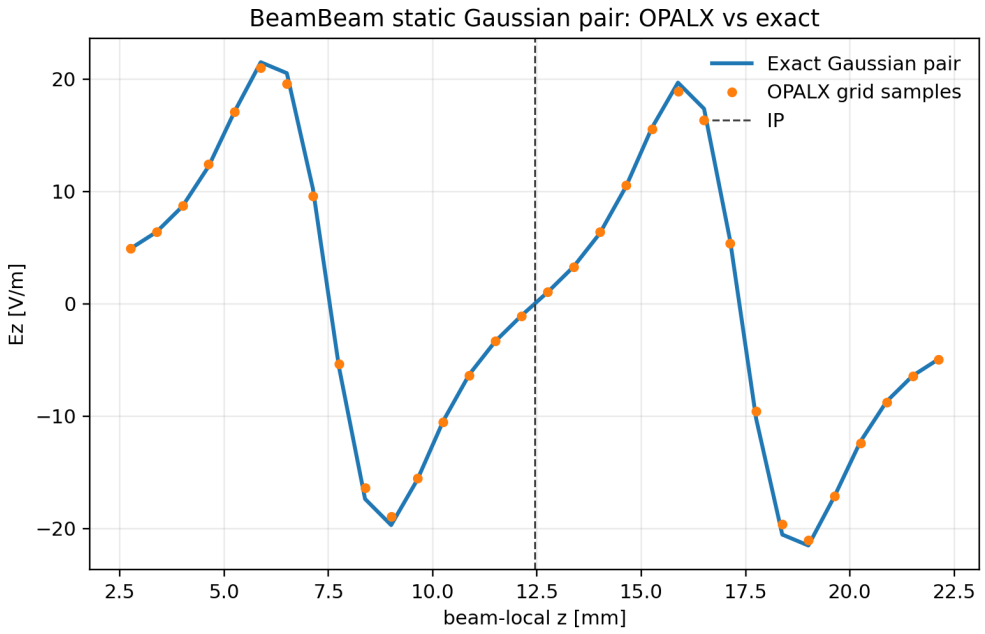


Figure 34.1: Longitudinal electric field for the static Gaussian BeamBeam validation. The markers are OPALX grid samples and the line is the exact electrostatic Gaussian-pair field.

34.4.1 Reproducibility

The files needed to reproduce Figure 34.1 and the table above are stored with this manual; the reproducer README is the entry point:

- `BeamBeam-static-1V.in`: OPALX input deck for the static Gaussian-pair run.
- `beam-beam-manufactured-solution.py`: Python comparison script for the OPALX ASCII field diagnostics.
- `README.md`: compact command-line recipe.

The assumptions are that OPALX has already been cloned and built, and that the executable is available as `build_openmp/src/opalx` from the OPALX repository root. The Python environment must provide `numpy`, `pandas`, and `matplotlib`; in the OPALX development checkout this is typically the `.venv-h6` environment.

From the OPALX repository root, copy the input deck into `sandbox/` and run the single-rank OpenMP case:

```
REPRO=~/git/physics-manual-opalx/sections/beam-beam/reproducers
IN=sandbox/BeamBeam-static-1V.in

mkdir -p sandbox data/sandbox
cp "$REPRO/BeamBeam-static-1V.in" "$IN"
mpiexec -n 1 build_openmp/src/opalx --info 1 "$IN"
```

The run writes active BeamBeam diagnostics under `data/sandbox/`. For the configuration documented here, the comparison uses

```
data/sandbox/BeamBeam-static-1V-RHO_scalar-beambeam_rho_pre-000003.dat
data/sandbox/BeamBeam-static-1V-PHI_scalar-beambeam_phi-000004.dat
data/sandbox/BeamBeam-static-1V-EF_vector-beambeam_e-000004.dat
```

Run the analytic comparison with

```
REPRO=~/git/physics-manual-opalx/sections/beam-beam/reproducers
D=data/sandbox
SCRIPT="$REPRO/beam-beam-manufactured-solution.py"
RHO="$D/BeamBeam-static-1V-RHO_scalar-beambeam_rho_pre-000003.dat"
PHI="$D/BeamBeam-static-1V-PHI_scalar-beambeam_phi-000004.dat"
EF="$D/BeamBeam-static-1V-EF_vector-beambeam_e-000004.dat"

python "$SCRIPT" \
  --compare-rho-dump "$RHO" \
  --compare-phi-dump "$PHI" \
  --compare-e-dump "$EF" \
  --sigma 1e-3
```

The relevant output is the set of relative L_2 errors for ρ, ϕ, E_x, E_y, and E_z, plus the on-axis value nearest to the IP. The reference values for this run are those listed in the results table above: ρ at about 5.22×10^{-2}, ϕ at about 3.04×10^{-3}, and the field components at about 1.5×10^{-2} to 2.4×10^{-2} relative error. The line `Ez(nearest grid sample to IP)` should report approximately 1.038298 V/m for the analytic value and 1.072345 V/m for OPALX.

If the tracker step sequence or diagnostic naming changes, the numeric suffixes of the dump files may change. In that case choose the active `beambeam_rho_pre`, `beambeam_phi`, and `beambeam_e` files with matching diagnostic headers and with a solver-frame copied-bunch separation closest to 10 mm.

35 Decay Processes

This section describes how *OPALX* models the spontaneous decay of unstable charged particles along the trajectory. Two channels are currently implemented: charged pion decay $\pi^{\pm} \to \mu^{\pm} + \nu_{\mu}$ (two-body, Section 35.2) and muon decay $\mu^{\pm} \to e^{\pm} + \nu + \bar{\nu}$ (three-body, polarized, Section 35.3). Both samplers share a common timing kernel (Section 35.1) and a common Lorentz boost to the lab frame (Section 35.4).

Decay is implemented as a *global process* applied once per tracker step, after the momentum kick. Particles flagged for decay in a given step are marked for deletion from the parent container and replaced by daughters created in a paired container; downstream tracking continues on the daughters with their new species mass, charge, and, where applicable, polarization vector defined as in the spin-tracking section.

35.1 Decay timing: relativistic Markov law

For each particle, the per-step decay probability follows the relativistic Markov law: time dilation extends the rest-frame lifetime τ_0 to a lab-frame lifetime $\gamma\tau_0$, and the exponential survival function gives

$$p_{\text{decay}}(\Delta t) = 1 - \exp(-\Delta t / \gamma\,\tau_0), \tag{35.1}$$

where $\gamma = \sqrt{1 + |\vec{P}|^2}$ is computed per-particle from the stored momentum $\vec{P}$ in $\beta\gamma$ units. Each step OPALX draws $u \sim U(0,1)$ and marks the particle for decay if $u < p_{\text{decay}}$; the parent is then collected into a compact view (alongside its position, momentum, and — when spin tracking is active — polarization) before daughters are sampled.

Rest-frame lifetimes used in OPALX (`Physics.h`):

Species	τ_0 [s]	Source
$\mu^{\pm}$	2.1969811×10^{-6}	[24]
$\pi^{\pm}$	2.6033×10^{-8}	[25]

Note that Δt in Equation 35.1 is the *tracker* step in lab time, not a proper-time interval; the time-dilation factor γ performs the conversion. For the small per-step decay probabilities typical in storage-ring or beamline applications, $1 - e^{-x} \approx x$, so this reduces to the familiar $p \approx \Delta t/(\gamma\tau_0)$ — but the exponential form remains correct for the rare large-Δt steps near a stopper or beam dump.

35.2 Pion decay: $\pi^{\pm} \to \mu^{\pm} + \nu_{\mu}$

35.2.1 Two-body kinematics

The pion is a pseudoscalar (spin 0, parity $-$). In the pion rest frame, energy-momentum conservation in a two-body decay against a massless neutrino fixes the daughter muon momentum uniquely [26]:

$$p^* = \frac{M_\pi^2 - m_\mu^2}{2\,M_\pi}, \qquad E^* = \frac{M_\pi^2 + m_\mu^2}{2\,M_\pi}. \tag{35.2}$$

Numerically, with the PDG masses in `Physics.h`, $p^* \approx 29.79$ MeV/c and $E^* \approx 109.78$ MeV.

35.2.2 Isotropic direction

Since the parent pion is spin-0, the rest-frame decay direction is isotropic. OPALX samples $\cos\theta^* \sim U(-1,1)$ and $\phi^* \sim U(0,2\pi)$, then constructs the rest-frame momentum $\vec{p}_\mu^{\,*} = p^*\,(\sin\theta^*\cos\phi^*, \sin\theta^*\sin\phi^*, \cos\theta^*)$ before boosting.

35.2.3 V-A polarization of the daughter muon

The decay $\pi \to \mu\,\nu_\mu$ proceeds through the V-A charged current. In the pion rest frame the neutrino is massless and emerges with definite helicity ($-$ for ν_μ, $+$ for $\bar{\nu}_\mu$); angular-momentum conservation with the spin-0 pion then forces the muon to carry the *opposite* helicity. The resulting muon polarization vector in the muon rest frame is therefore

$$\vec{P}_\mu = h\,\hat{p}_\mu^{\,*}, \qquad h = \begin{cases} +1 & \text{for } \pi^- \to \mu^- \\ -1 & \text{for } \pi^+ \to \mu^+ \end{cases} \tag{35.3}$$

The convention follows the standard accelerator notation in which positive helicity means $\vec{P}$ parallel to the daughter momentum.

A subtle but useful point: the boost that takes the muon from the pion rest frame to the lab is along $\hat{p}_\mu^{\,*}$ itself, so the projection of $\vec{P}_\mu$ along that axis is invariant under the boost. Writing the rest-frame unit vector $\hat{p}_\mu^{\,*}$ directly into the per-particle polarization storage is therefore the exact lab-frame value of $\vec{P}_\mu$ — no additional spin rotation is required at boost time. This is what makes the daughter-muon polarization computation cheap.

35.3 Muon decay: $\mu^{\pm} \to e^{\pm} + \nu + \bar{\nu}$

Muon decay is a three-body weak process. With two unobserved neutrinos, the daughter electron is described by a non-trivial joint energy-angle distribution. We use the standard Michel parameterization in the limit of an unpolarized electron with massless neutrinos [27].

35.3.1 Energy marginal: Michel spectrum

Define the reduced electron energy $x = 2\,E_e/m_\mu \in [0,1]$. The energy marginal of the differential decay rate is the Michel spectrum,

$$f(x) = 2\,x^2\,(3-2x), \qquad x \in [x_{\min}, 1], \tag{35.4}$$

with the kinematic threshold $x_{\min} = m_e/(m_\mu/2) \approx 9.7 \times 10^{-3}$. The spectrum peaks at $x = 1$ with $f_{\max} = 2$. OPALX samples x by rejection against this upper bound.

35.3.2 Joint distribution and the V-A asymmetry

For a muon with rest-frame polarization vector $\vec{P}$, the joint differential rate factorizes in the standard V-A form [27]

$$\frac{d\Gamma}{dx\,d\cos\theta} \;\propto\; f(x)\,\big[\,1 + \alpha(x)\,\cos\theta\,\big], \tag{35.5}$$

where θ is the angle between the electron momentum and $\hat{\vec{P}}$ in the muon rest frame, and the asymmetry coefficient is

$$\alpha(x) = s\,|\vec{P}|\,\frac{2x-1}{3-2x}, \qquad s = \begin{cases} +1 & \text{for } \mu^- \\ -1 & \text{for } \mu^+ \end{cases} \tag{35.6}$$

The sign s reflects the conventional V-A coupling — for μ^- the high-energy electron is preferentially emitted *along* $\hat{\vec{P}}$, while for μ^+ the high-energy positron prefers the opposite direction. This is the V-A signature that motivates spin-dependent muon decay tracking in the first place.

35.3.3 Inverse-CDF sampling of $\cos\theta$

The conditional Equation 35.5 is linear in $\cos\theta$, so the conditional CDF is quadratic and explicitly invertible. Writing $c \equiv \cos\theta$, the normalized conditional density on $[-1,1]$ is

$$p(c \mid x, \vec{P}) = \tfrac{1}{2}\,(1 + \alpha\,c), \tag{35.7}$$

with CDF

$$F(c) = \tfrac{1}{2}(c+1) + \tfrac{\alpha}{4}(c^2 - 1). \tag{35.8}$$

For $u \sim U(0,1)$, setting $F(c) = u$ gives the quadratic

$$\frac{\alpha}{4}\,c^2 + \frac{1}{2}\,c - \Big(\frac{\alpha}{4} + \frac{1}{2} - u\Big) = 0, \tag{35.9}$$

solved analytically and the root in $[-1,1]$ selected. The degenerate limit $\alpha \to 0$ (unpolarized muon or zero V-A asymmetry at $x = \frac{1}{2}$) recovers the uniform sampling $c = 2u - 1$, as expected.

The azimuth ϕ around $\hat{\vec{P}}$ is uniform on $[0, 2\pi)$, and the rest-frame electron momentum direction in the $\hat{\vec{P}}$-aligned local frame is $(\sin\theta\cos\phi, \sin\theta\sin\phi, \cos\theta)$. A single rotation taking $\hat{z} \to \hat{\vec{P}}$ then realigns this vector with the lab-frame Cartesian axes (which are the axes the polarization vector itself is referenced to, see the spin-tracking section); the rotated rest-frame momentum is fed to the Lorentz boost of the next section.

35.4 Lorentz boost to the lab frame

Both decay samplers share the same lab-frame boost. With the parent momentum stored as $\vec{P}_{parent}$ in $\beta\gamma$ (dimensionless) units,

$$\gamma = \sqrt{1 + |\vec{P}_{parent}|^2}, \qquad \vec{\beta} = \frac{\vec{P}_{parent}}{\gamma}, \tag{35.10}$$

the spatial part of the daughter four-momentum boosts as

$$\vec{p}_{\mathrm{lab}} = \vec{p}_{\mathrm{RF}} + \left[\frac{(\gamma - 1)\,\vec{\beta}\cdot\vec{p}_{\mathrm{RF}}}{\beta^2} + \gamma\, E_{\mathrm{RF}}\right]\vec{\beta}. \tag{35.11}$$

The energy E_{RF} is the daughter rest-frame energy ($E_e = x\, m_\mu/2$ for muon decay, E^* from Equation 35.2 for pion decay). At storage, OPALX writes the daughter momentum back in $\beta\gamma$ units of the *daughter* species mass: $\vec{P}_{\mathrm{daughter}} = \vec{p}_{\mathrm{lab}}/m_d$, so that the downstream Boris pusher and T-BMT pusher consume it without unit conversion.

The polarization vector of the daughter muon from pion decay is written directly in the rest frame (Equation 35.3) and, as noted above, requires no boost rotation because the daughter momentum direction is the boost axis. For muon decay the daughter electron polarization is not stored — the rest of the simulation treats decay electrons as unpolarized — but the muon decay sampler needs the *parent* polarization, which is read from the muon container's polarization view at decay time.

35.5 Numerical implementation

The runtime cost of the decay process is dominated by the rejection sampling of the Michel spectrum; everything else is closed-form.

- **Decay flagging.** Per-particle Bernoulli draw against Equation 35.1; survivors retain their state, marked particles are compacted into a dense view of $(\vec{R}, \vec{P}, \Delta t, \vec{P}_{\mathrm{pol}})$ tuples by a Kokkos parallel scan.
- **Michel x.** Rejection against $f_{\max} = 2$; average acceptance is $\int_0^1 f(x)\,dx/f_{\max} \approx 0.41$, so the rejection loop is short and predictable.
- **Muon $\cos\theta$.** Closed-form root of Equation 35.9 — no rejection, no iteration.
- **Pion $\cos\theta$.** A single uniform draw.

- **Daughter momentum.** Stored as $\vec{P}_{\text{daughter}} = \vec{p}_{\text{lab}}/m_d$, consistent with the rest of OPALX's $\beta\gamma$ convention.

Random numbers are sourced from a Kokkos `XorShift64_Pool` seeded per-container, so each rank's decay stream is reproducible and statistically independent across MPI ranks.

35.6 Validation

The samplers are verified by distribution-level unit tests that compare 10^4–10^6 generated daughters against the analytic forms above.

- **Pion decay.** TestPionDecaySpectrum.cpp checks that every daughter muon has $|\vec{p}| = p^*$ from Equation 35.2 within machine precision, that the rest-frame $\cos\theta$ histogram is statistically uniform on $[-1, 1]$ (verifying the spin-0 assumption), and that the lab-frame energy spectrum after a boost along $\hat{z}$ is a flat box on $[E_-, E_+]$ with endpoints $E_\pm = \gamma E^* \pm \beta\gamma\, p^*$.
- **Muon decay.** TestMuonDecaySpectrum.cpp fires fully-polarized muons at rest, fixes the reduced energy x, and compares the daughter $\cos\theta$ histogram against the analytic linear-in-c slope $\alpha(x)$ from Equation 35.6. The same test is run for μ^- and μ^+ to confirm the V-A sign flip.

These tests are run automatically as part of the ctest suite and guard the kinematic and polarization correctness of the samplers against regressions.

36 Spin Tracking

This section describes how *OPALX* evolves the per-particle spin polarization vector along the trajectory. The model is the Thomas-BMT equation [23], [28], integrated as a closed-form Rodrigues rotation so that the magnitude $|\vec{P}|$ is preserved exactly per step.

36.1 Polarization vector

OPALX stores, for each tracked particle, a three-component polarization vector

$$\vec{P} = (\langle\hat{\sigma}_x\rangle,\ \langle\hat{\sigma}_y\rangle,\ \langle\hat{\sigma}_z\rangle), \tag{36.1}$$

where each component is the expectation value of the corresponding Pauli operator. The expectation is evaluated in the particle's **instantaneous (Lorentz) rest frame**, while the three Cartesian axes along which it is projected are the **lab-frame axes**. This hybrid convention is the one the BMT equation is naturally written in: the spin state lives in the rest frame, but the rotation rate $\vec{\Omega}$ is expressed using the lab-frame fields $\vec{E}, \vec{B}$ at the particle's position, so keeping the projection axes lab-aligned avoids a redundant boost rotation inside the integrator.

The magnitude $|\vec{P}|$ is bounded by the spin-$\frac{1}{2}$ Bloch sphere,

$$0 \le |\vec{P}| \le 1, \tag{36.2}$$

with $|\vec{P}| = 1$ corresponding to a fully polarized state and $|\vec{P}| = 0$ to an unpolarized state.

36.2 The Thomas-BMT equation

In the lab frame, with t the lab-frame time, $\vec{P}$ evolves as

$$\frac{d\vec{P}}{dt} = \vec{\Omega} \times \vec{P}, \tag{36.3}$$

where the angular velocity $\vec{\Omega}$ for a particle of charge q, rest mass m, and gyromagnetic anomaly $G = (g-2)/2$ in electromagnetic fields $\vec{E}, \vec{B}$ is

$$\vec{\Omega} = -\frac{q}{m}\left[\left(G + \frac{1}{\gamma}\right)\vec{B} - \frac{G\gamma}{\gamma+1}\,(\vec{\beta}\cdot\vec{B})\,\vec{\beta} - \left(G + \frac{1}{\gamma+1}\right)\frac{\vec{\beta}\times\vec{E}}{c}\right]. \tag{36.4}$$

Here $\vec{\beta} = \vec{v}/c$ is the lab-frame velocity in units of c and $\gamma = (1-\beta^2)^{-1/2}$ is the corresponding Lorentz factor. The derivation is given in [23, Sec. 11.11]; we use the standard sign convention for $\vec{\Omega}$ such that Equation 36.3 describes the time evolution of $\vec{P}$ rather than that of a rotating frame.

Term-by-term, Equation 36.4 separates into three physically distinct contributions:

- **Cyclotron-Larmor term**, $-\frac{q}{m}\left(G + \frac{1}{\gamma}\right)\vec{B}$. The $1/\gamma$ piece is the relativistic Larmor precession at the cyclotron frequency $\omega_c = qB/(\gamma m)$, common to *any* charged particle in a magnetic field. The G piece is the additional precession driven by the anomalous magnetic moment — it is the part the BMT equation contributes beyond the Dirac result $g = 2$.
- **Longitudinal-B correction**, $+\frac{q}{m}\frac{G\gamma}{\gamma+1}\left(\vec{\beta}\cdot\vec{B}\right)\vec{\beta}$. This subtracts the component of $\vec{B}$ along the motion that is ineffective at rotating $\vec{P}$. It vanishes when $\vec{\beta} \perp \vec{B}$ (the canonical storage-ring geometry).
- **Motional-electric term**, $+\frac{q}{m}\left(G + \frac{1}{\gamma+1}\right)\left(\vec{\beta}\times\vec{E}\right)/c$. This is the BMT response to electric fields. Even at $G = 0$ (Dirac) it does not vanish — the residual $1/(\gamma+1)$ is the Thomas-precession piece, which is a kinematic effect of the successive Lorentz boosts and survives in the absence of anomalous moments.

Setting $G = 0$ in Equation 36.4 reduces the rotation to pure Larmor precession of a Dirac particle: $\vec{\Omega} = -(q/m)\left(\vec{B}/\gamma\right)$, i.e. at ω_c when $\vec{\beta} \perp \vec{B}$. Any deviation of g from 2 — which is generic for composite particles and even for QED loop-corrected leptons — adds the $G\vec{B}$ contribution that physically defines the BMT regime.

OPALX currently tabulates G for the species that have a well-defined classical anomaly:

Species	$G = (g-2)/2$	Source
Electron, positron	$1.15965218128 \times 10^{-3}$	QED [23]
Muon ($\mu^{\pm}$)	$1.16592061 \times 10^{-3}$	[24]
Proton, antiproton	1.792847386	[23]

Species not listed default to $G = 0$. In the current OPALX release only the muon channel is actively spin-tracked downstream; extending the integrator itself to another species is just a matter of populating G from the species table.

36.3 OPALX unit conventions

OPALX stores rest mass as the **rest energy** in eV (numerically $m\,c^2$), and charge in **proton-charge units** (dimensionless, e.g. -1 for μ^-, $+1$ for μ^+). With these conventions, the SI ratio q/m that appears in Equation 36.4 becomes

$$\left.\frac{q}{m}\right|_{\mathrm{SI}} = \frac{\mathrm{charge}\cdot c^2}{\mathrm{mass}_{\mathrm{eV}}}, \tag{36.5}$$

because the elementary-charge factor cancels between numerator (units of e) and denominator (units of eV). This is the same idiom used by the Boris pusher [12] in OPALX; the spin and momentum updates therefore consume q, m in identical internal forms with no separate unit conversion needed.

36.4 Numerical integration: Rodrigues rotation

A naive explicit ODE solver applied to Equation 36.3 (e.g. RK4) does not preserve $|\vec{P}|$ and accumulates a slow magnitude drift over time. OPALX therefore integrates Equation 36.3 as a closed-form rotation.

For a single step of duration Δt during which $\vec{\Omega}$ is held constant, the exact solution is a rotation of $\vec{P}$ about $\hat{n} \equiv \vec{\Omega}/|\vec{\Omega}|$ by angle

$$\phi = |\vec{\Omega}|\,\Delta t. \tag{36.6}$$

The rotation is evaluated through Rodrigues' formula,

$$\vec{P}' = \cos\phi\;\vec{P} + \sin\phi\,(\hat{n}\times\vec{P}) + (1-\cos\phi)\,(\hat{n}\cdot\vec{P})\,\hat{n}. \tag{36.7}$$

By construction this preserves $|\vec{P}|$ to machine precision per step, independent of ϕ, so the magnitude drift typical of explicit non-symplectic schemes is eliminated.

The remaining accuracy concern is the **constant-$\vec{\Omega}$ assumption** across the step. For small per-step rotations the error incurred by holding $\vec{\Omega}$ frozen at its value at the start of the step is third order in ϕ. To bound this error robustly when the beamline contains strong fields or when Δt is large compared to the spin-precession period, OPALX sub-steps whenever the per-step rotation exceeds a threshold $\phi_{\max} = 0.1$ rad: the timestep is split into

$$N_{\text{sub}} = \left\lceil \frac{\phi}{\phi_{\max}} \right\rceil \tag{36.8}$$

equal sub-rotations of angle ϕ/N_{sub} each, with $\vec{\Omega}$ re-evaluated identically for each (since $\vec{E}, \vec{B}, \gamma$ do not change inside a single tracker step). The threshold $\phi_{\max} = 0.1$ rad bounds the rotation-truncation error at the 10^{-4} relative level per step, which is well below the float storage precision of $\vec{P}$.

36.5 Placement in the tracker step

The polarization update is applied after both external and space-charge contributions to $\vec{E}$ and $\vec{B}$ have been accumulated at each particle, but *before* the momentum kick that closes the time step. This ordering guarantees that the $\vec{E}, \vec{B}$ used to construct $\vec{\Omega}$ in Equation 36.4 are precisely the fields the particle experiences during the step — the same fields the Boris kick uses — so the momentum and spin halves of the step see a consistent electromagnetic environment, and no extra mid-step field-interpolation is required.

37 - Physics

This note collects the physics model and benchmarks of the laser and photon functionality in *OPALX*.

37.1 Full Workflow

The migrated top-level driver `generate-gamma-gamma-results.sh` regenerates the full published gamma-gamma benchmark state. It builds the required OPALX benchmark targets, runs the inverse-Compton and Breit-Wheeler physics tests, calls the shared CAIN helper scripts in `~/git/cain`, and rerenders the physics notes.

Use it when you want to refresh all benchmark figures and note pages in one pass:

```
cd opalx-manual/physics/gamma-gamma/scripts
./generate-gamma-gamma-results.sh --opalx-build /path/to/OPALX/build_openmp
```

The narrower CAIN-side helpers remain useful when only one of the two physics notes needs to be regenerated.

The corresponding histogram inputs are stored as the gamma-gamma reference dataset, outside the text manual.

37.2 Linear Compton Benchmark

37.2.1 Scope

This note documents the current *OPALX* validation path for linear inverse Compton scattering in the weak-field regime. The immediate goal is not yet a full tracking implementation, but a CAIN-validated benchmark path for the process

$$e^- + \gamma_L \to e^- + \gamma.$$

The benchmark follows the same staged strategy as the Breit-Wheeler note:

- start from a fixed-geometry, unpolarized, linear process,
- validate deterministic and sampled spectra against CAIN,

- then extend to broader beam models without enabling tracked interaction yet.

The baseline geometry used here is a 90° crossing between a `1 GeV` electron beam along $\hat{z}$ and a laser photon beam along $\hat{x}$ with wavelength `1030 nm`.

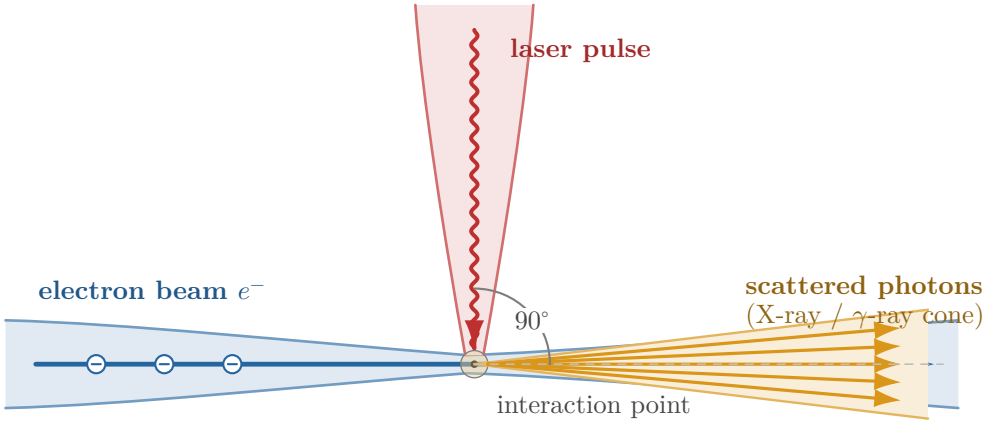


Figure 37.1: Inverse Compton scattering.

37.2.2 Physics Models

37.2.2.1 Basic Kinematics

For a laser wavelength λ_L, the photon energy is

$$\omega_L = \frac{2\pi\hbar c}{\lambda_L}. \tag{37.1}$$

For an electron with total energy E_e and momentum magnitude $p_e = \sqrt{E_e^2 - m_e^2}$, the exact forward-scattered photon energy in the `90 degree` benchmark geometry is

$$\omega'_\gamma = \frac{E_e\,\omega_L}{\omega_L + \frac{m_e^2}{E_e + p_e}}. \tag{37.2}$$

The current OPALX helper uses this numerically stable form directly in the unit benchmark.

37.2.2.2 Rest-Frame Model

The exact linear Compton kernel is evaluated in the electron rest frame. Let ω_1 be the incoming photon energy there and let Θ^* be the photon scattering angle. The scattered photon energy is

$$\omega_2(\Theta^*) = \frac{\omega_1}{1 + (\omega_1/m_e)(1 - \cos\Theta^*)}. \tag{37.3}$$

The unpolarized Klein-Nishina differential cross section is

$$\frac{d\sigma}{d\Omega^*} = \frac{r_e^2}{2}\left(\frac{\omega_2}{\omega_1}\right)^2\left(\frac{\omega_1}{\omega_2} + \frac{\omega_2}{\omega_1} - \sin^2\Theta^*\right). \tag{37.4}$$

Changing variables from $\cos\Theta^*$ to ω_2 gives the one-dimensional energy spectrum in the rest frame,

$$\frac{d\sigma}{d\omega_2} = 2\pi\frac{d\sigma}{d\Omega^*}\frac{m_e}{\omega_2^2}. \tag{37.5}$$

with exact support $\omega_2 \in [\omega_1/(1 + 2\omega_1/m_e),\, \omega_1]$.

The exact total Klein-Nishina cross section can be written with $\kappa = \omega_1/m_e$ as

$$\sigma_{\mathrm{KN}} = 2\pi r_e^2\left[\frac{1+\kappa}{\kappa^3}\left(\frac{2\kappa(1+\kappa)}{1+2\kappa} - \ln(1+2\kappa)\right) + \frac{\ln(1+2\kappa)}{2\kappa} - \frac{1+3\kappa}{(1+2\kappa)^2}\right]. \tag{37.6}$$

For $\kappa \ll 1$ this approaches the Thomson limit

$$\sigma_T = \frac{8\pi}{3}r_e^2. \tag{37.7}$$

37.2.3 Implemented OPALX Benchmark

The current *OPALX* implementation is the CAIN-validated benchmark path. The generated API reference covers the documented interfaces; the implementation and tests are in these source files:

- `src/Physics/LinearCompton.h`
- `src/Physics/LinearCompton.cpp`
- `unit_tests/Physics/TestLinearCompton.cpp`
- `unit_tests/Physics/TestLinearComptonSpectrum.cpp`
- `unit_tests/Physics/LinearComptonSpectrumBenchmark.cpp`
- `generate-gamma-gamma-results.sh`
- `~/git/cain/generate-linear-compton-results.sh`
- `~/git/cain/build-cain.sh`

The benchmark remains deliberately narrow:

- no tracked `LASER` interaction,
- no photon bunch population,
- no luminosity or overlap integration beyond the benchmark setup.

The validated observables are now:

- ☒ photon energy spectrum,
- ☒ photon laboratory polar-angle spectrum,
- ☒ joint E_γ vs. $\theta_{\gamma,\mathrm{lab}}$ map,
- ☒ finite-beam energy, angle, and joint spectra,
- ☒ finite-beam plus energy-spread energy, angle, and joint spectra,
- ☒ overlap-restricted finite-beam energy and angle spectra.

37.2.4 Results of Code Comparison

The current CAIN-backed *OPALX* benchmark covers weak-field linear inverse Compton scattering for:

- electron total energy `1 GeV`,
- laser wavelength `1030 nm`,
- crossing angle 90°,
- unpolarized laser `STOKES=(0,0,0)`,
- CAIN weak-field parameter $\xi = 0.2955 < 0.3$.

Single-electron benchmark agreement:

- ☒ energy spectrum, deterministic: mean difference about `0.5%`, L_1 about `0.0767`
- ☒ energy spectrum, sampled: mean difference about `0.45%`, L_1 about `0.0763`
- ☒ lab-angle spectrum, deterministic: mean difference about `4.0%`, L_1 about `0.0400`
- ☒ lab-angle spectrum, sampled: mean difference about `1.3%`, L_1 about `0.0250`
- ☒ joint E_γ--$\theta_{\gamma,\mathrm{lab}}$ map, deterministic: mean-energy difference about `0.48%`, mean-angle difference about `4.03%`, L_1 about `0.0849`
- ☒ joint E_γ--$\theta_{\gamma,\mathrm{lab}}$ map, sampled: mean-energy difference about `0.44%`, mean-angle difference about `1.21%`, L_1 about `0.0709`

Finite-beam benchmark agreement:

- ☒ baseline finite beam (`sigma_x = sigma_y = 1 mm`, `sigma_{x'} = sigma_{y'} = 1 mrad`): energy mean difference about `0.58%`, L_1 about `0.0316`

- ☒ baseline finite beam angle spectrum: mean difference about `0.21%`, L_1 about `0.0142`
- ☒ baseline finite-beam joint E_γ--$\theta_{\gamma,\mathrm{lab}}$ map: mean-energy difference about `0.58%`, mean-angle difference about `0.11%`, L_1 about `0.0741`
- ☒ finite beam with relative energy spread `sigma_E / E = 1.0e-3`: energy mean difference about `0.57%`, L_1 about `0.0305`
- ☒ finite beam with relative energy spread `sigma_E / E = 1.0e-3` angle spectrum: mean difference about `0.17%`, L_1 about `0.0157`
- ☒ finite beam with relative energy spread `sigma_E / E = 1.0e-3` joint E_γ--$\theta_{\gamma,\mathrm{lab}}$ map: mean-energy difference about `0.57%`, mean-angle difference about `0.14%`, L_1 about `0.0686`
- ☒ overlap-restricted finite-beam energy spectrum: mean difference about `0.71%`, L_1 about `0.0289`
- ☒ overlap-restricted finite-beam angle spectrum: mean difference about `0.084%`, L_1 about `0.0135`

Single-electron overlays:

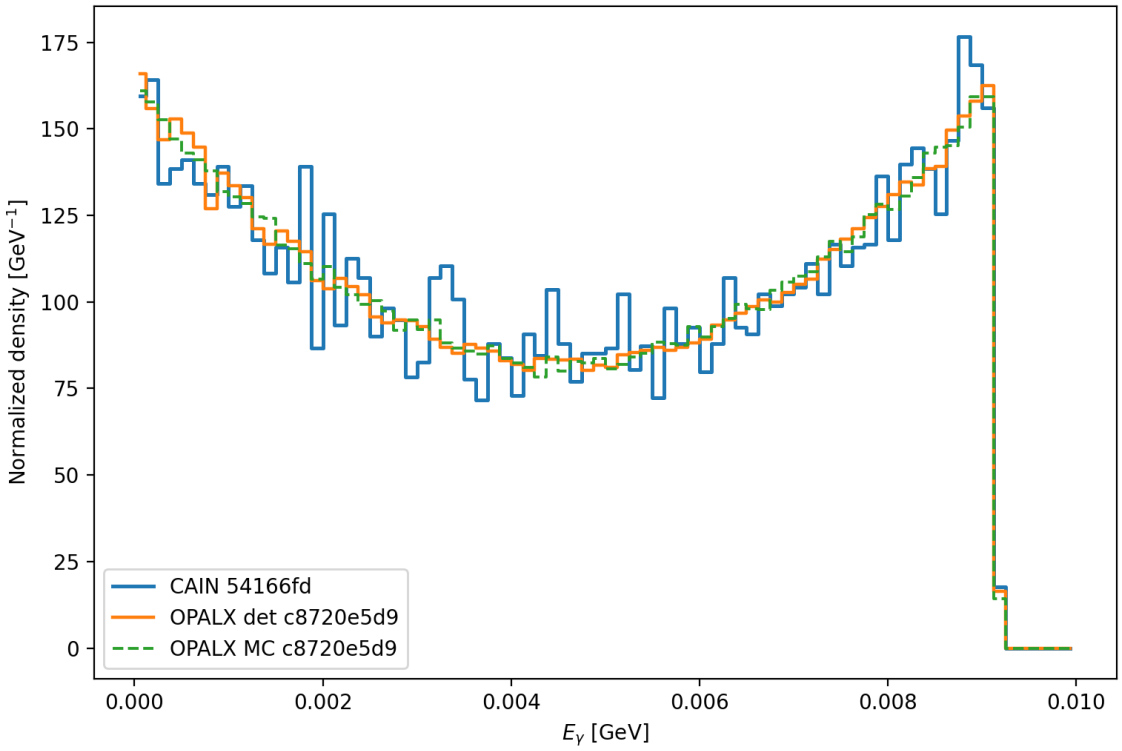


Figure 37.2: Weak-field 90-degree linear-Compton photon energy spectrum. The figure compares the CAIN reference, the OPALX deterministic benchmark, and the OPALX Monte Carlo benchmark for $E_e = 1$ GeV, $_L = 1030$ nm, and $= 0.2955$.

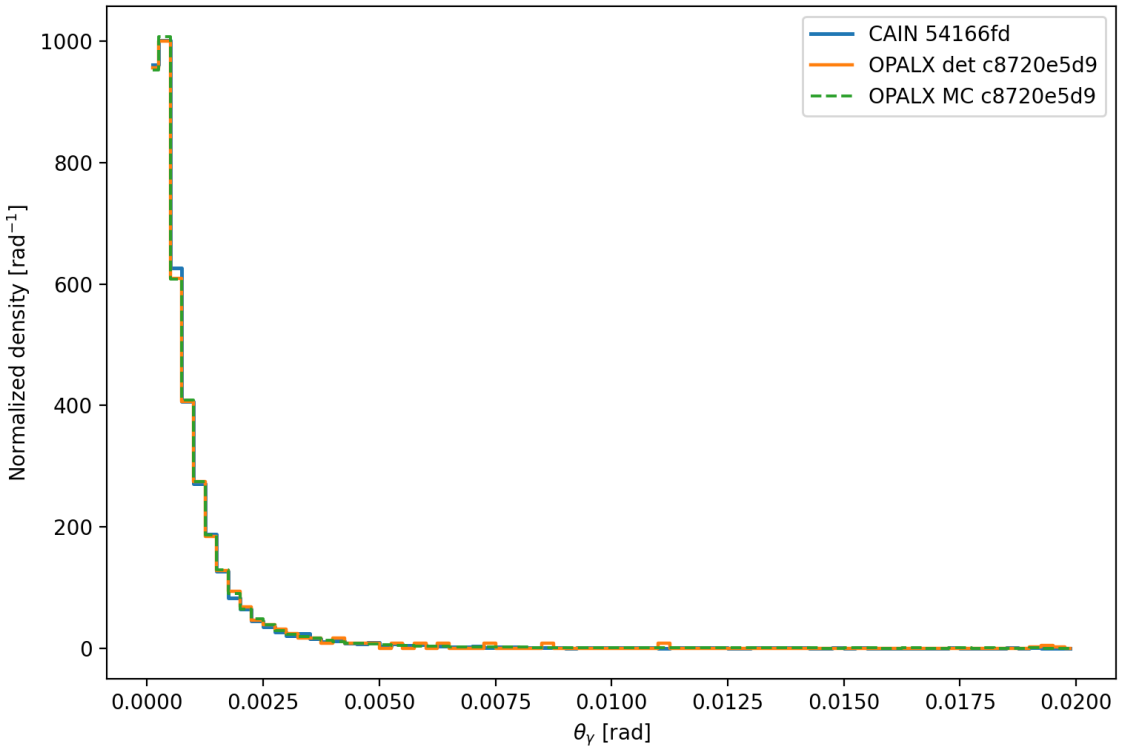


Figure 37.3: Weak-field 90-degree linear-Compton photon laboratory polar-angle spectrum. The figure compares the CAIN reference, the OPALX deterministic benchmark, and the OPALX Monte Carlo benchmark on the common angular histogram grid.

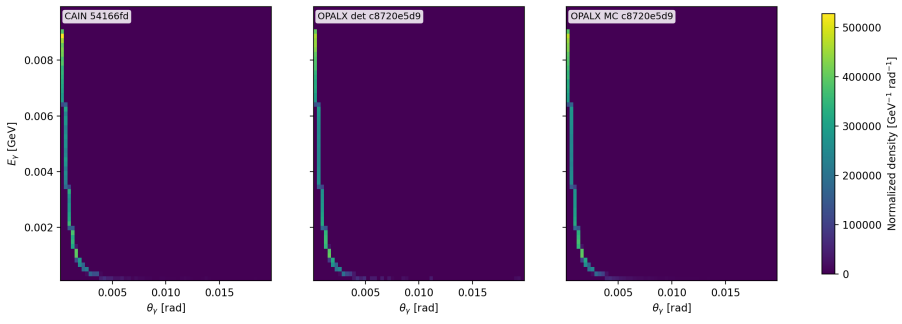


Figure 37.4: Weak-field 90-degree linear-Compton joint photon spectrum. The panels show the CAIN reference, the OPALX deterministic benchmark, and the OPALX Monte Carlo E versus density on the common 2D grid.

Finite-beam overlays:

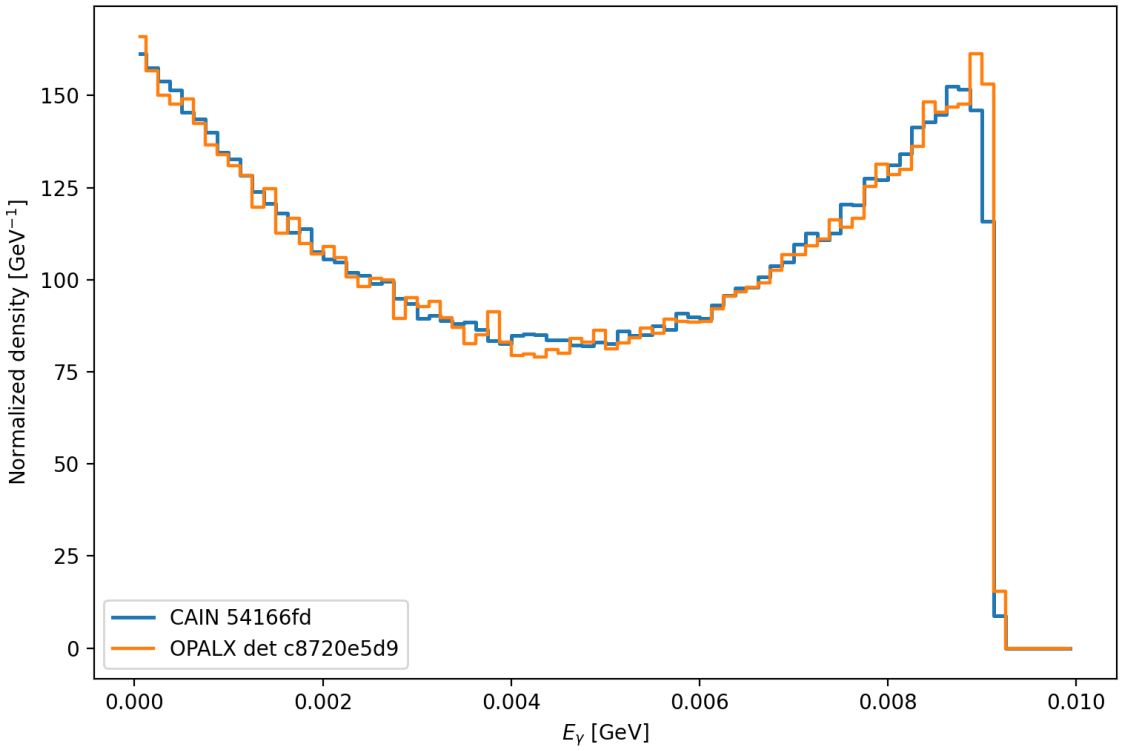


Figure 37.5: Finite-beam linear-Compton photon energy spectrum for the benchmark beam with $_x = _y = 1$ mm and $_{x'} = _{y'} = 1$ mrad. The figure compares CAIN and the OPALX Monte Carlo benchmark.

37.2.5 Build and Run

The CAIN build helper is now shared across the gamma-gamma notes in the common workspace `~/git/cain`.

From `~/git/cain`:

```
./build-cain.sh --download
./build-cain.sh --compile
```

The preferred one-shot workflow is run from the docs tree:

```
cd opalx-manual/physics/gamma-gamma/scripts
./generate-gamma-gamma-results.sh --opalx-build /path/to/opalx-laser/build_openmp
```

This top-level driver:

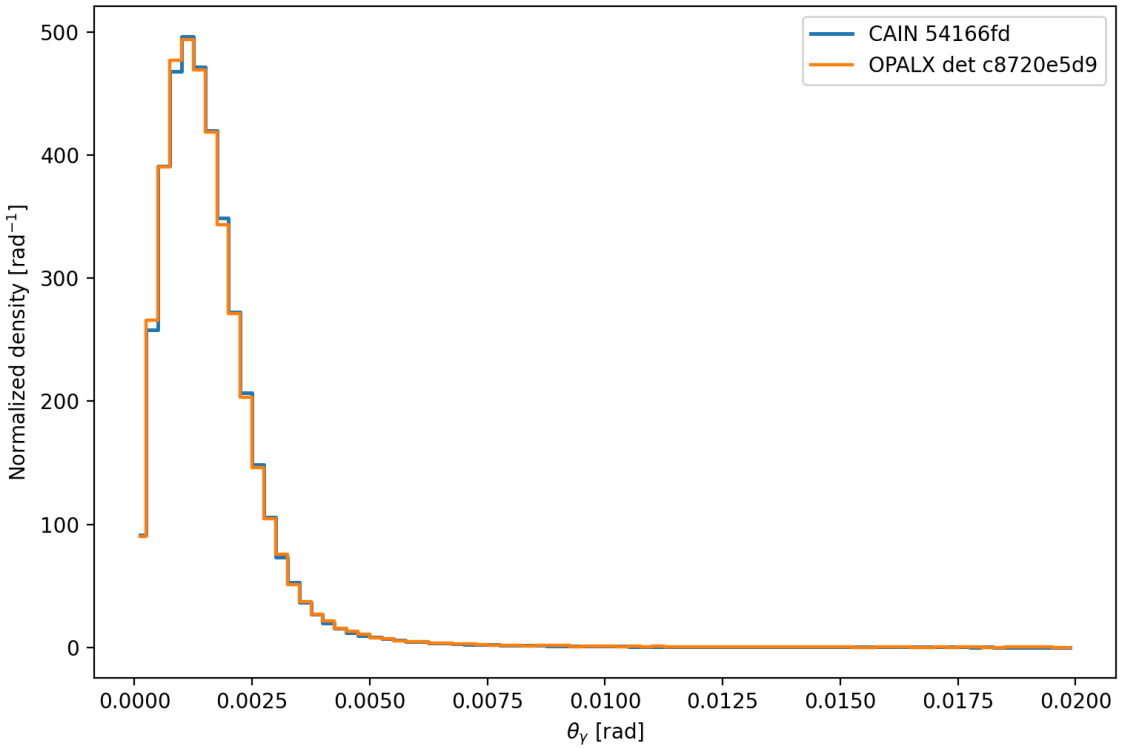


Figure 37.6: Finite-beam linear-Compton photon laboratory polar-angle spectrum for the same benchmark beam. The figure compares CAIN and the OPALX Monte Carlo benchmark on the common angular histogram grid.

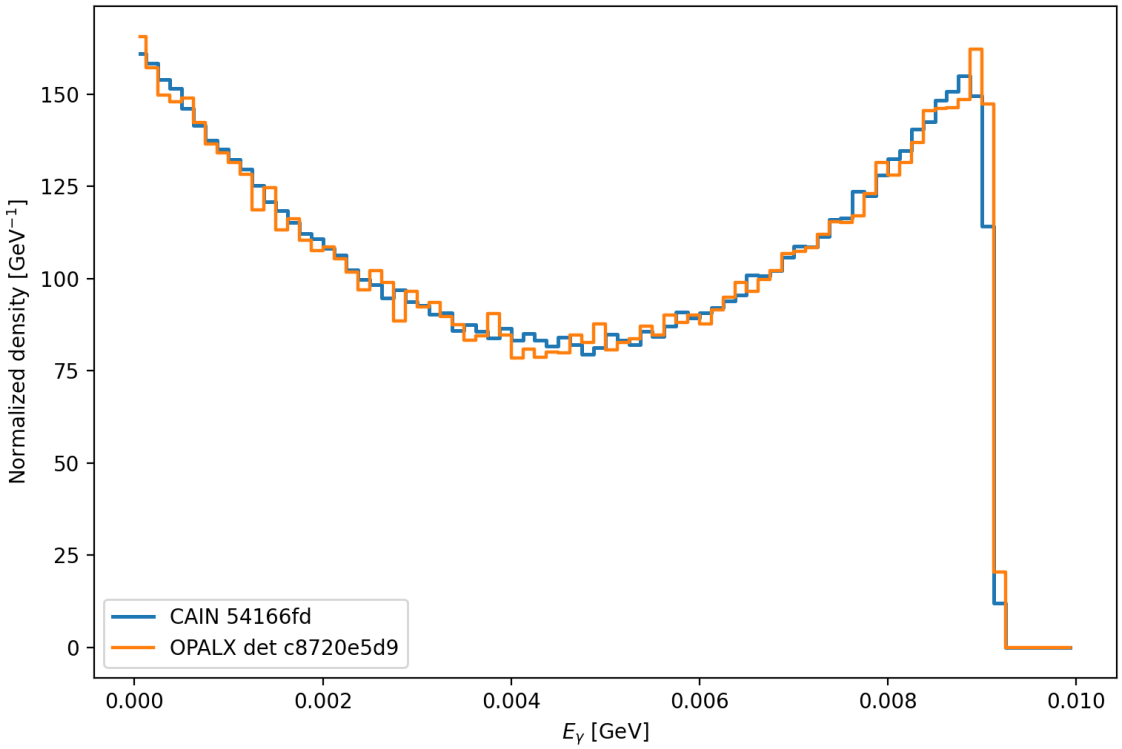


Figure 37.7: Finite-beam linear-Compton photon energy spectrum with additional electron-beam relative energy spread $_{\mathrm{E}}$/E = 1e-3. The figure compares CAIN and the OPALX Monte Carlo benchmark.

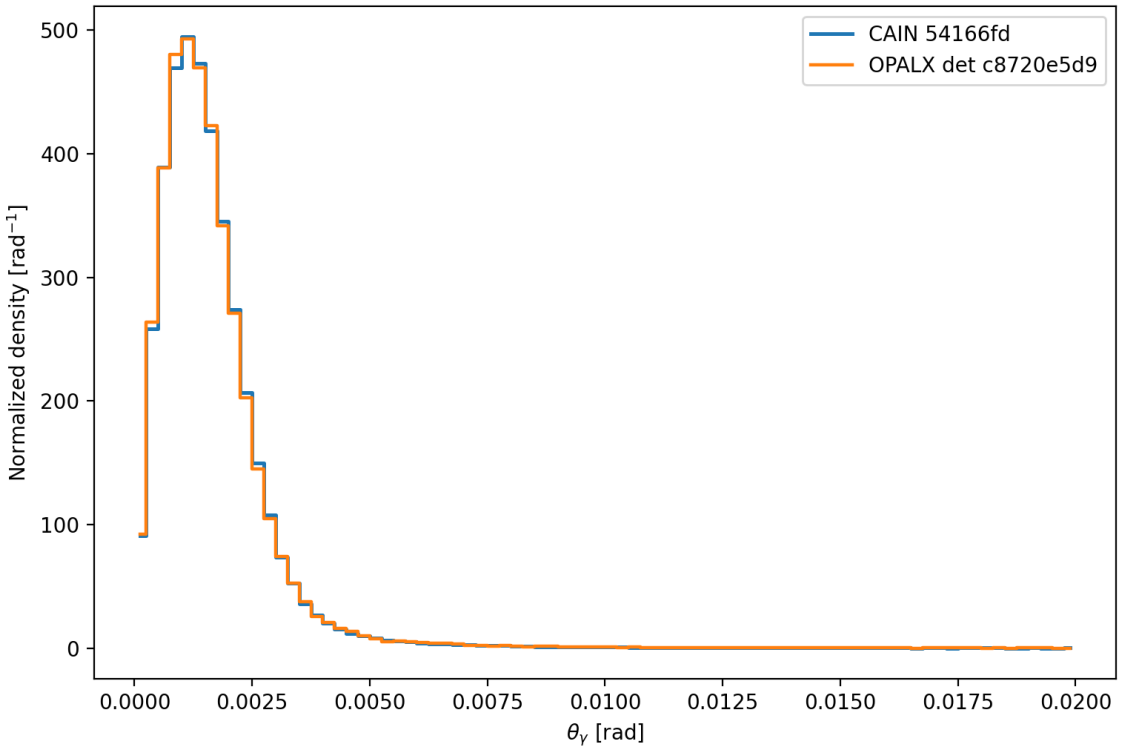


Figure 37.8: Finite-beam linear-Compton photon laboratory polar-angle spectrum with additional electron-beam relative energy spread $_{E}/E = 1e\text{-}3$. The figure compares CAIN and the OPALX Monte Carlo benchmark.

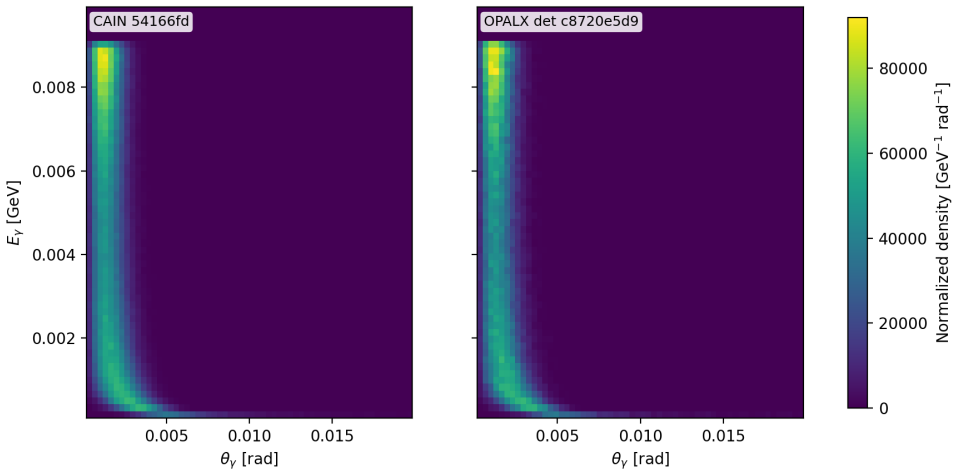


Figure 37.9: Finite-beam joint linear-Compton photon spectrum for the benchmark beam with $_{x} = _{y} = 1$ mm and $_{x'} = _{y'} = 1$ mrad. The panels compare the CAIN reference and the OPALX Monte Carlo E versus density on the common 2D grid.

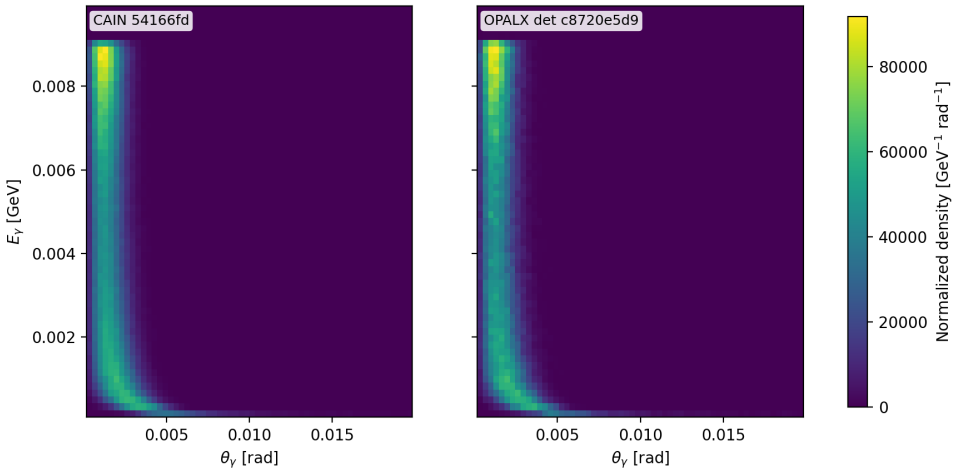


Figure 37.10: Finite-beam joint linear-Compton photon spectrum with additional electron-beam relative energy spread $_E/E = 1e\text{-}3$. The panels compare the CAIN reference and the OPALX Monte Carlo E versus density on the common 2D grid.

- builds the OPALX gamma-gamma benchmark targets,
- runs the inverse-Compton and Breit-Wheeler unit/regression suites,
- regenerates the CAIN and OPALX benchmark data for both notes,
- republishes the comparison figures, and
- renders the gamma-gamma note pages.

If only the linear-Compton assets need to be refreshed, use:

```
cd ~/git/cain
./generate-linear-compton-results.sh --opalx-build /path/to/opalx-laser/build_openmp
```

This script:

- runs the CAIN decks `~/git/cain/cain-linear-compton-90deg.i`, `~/git/cain/cain-linear-compton-90deg-finite-beam.i`, `~/git/cain/cain-linear-compton-90deg-finite-beam-energy-spread.i`, and `~/git/cain/cain-linear-compton-90deg-finite-beam-overlap.i`,
- regenerates the stored CAIN histogram references in `figures/reference-data/`,
- runs the OPALX benchmark executable `LinearComptonSpectrumBenchmark`,
- regenerates the published 1D and 2D comparison plots in `figures/reference-data/`.

The script expects:

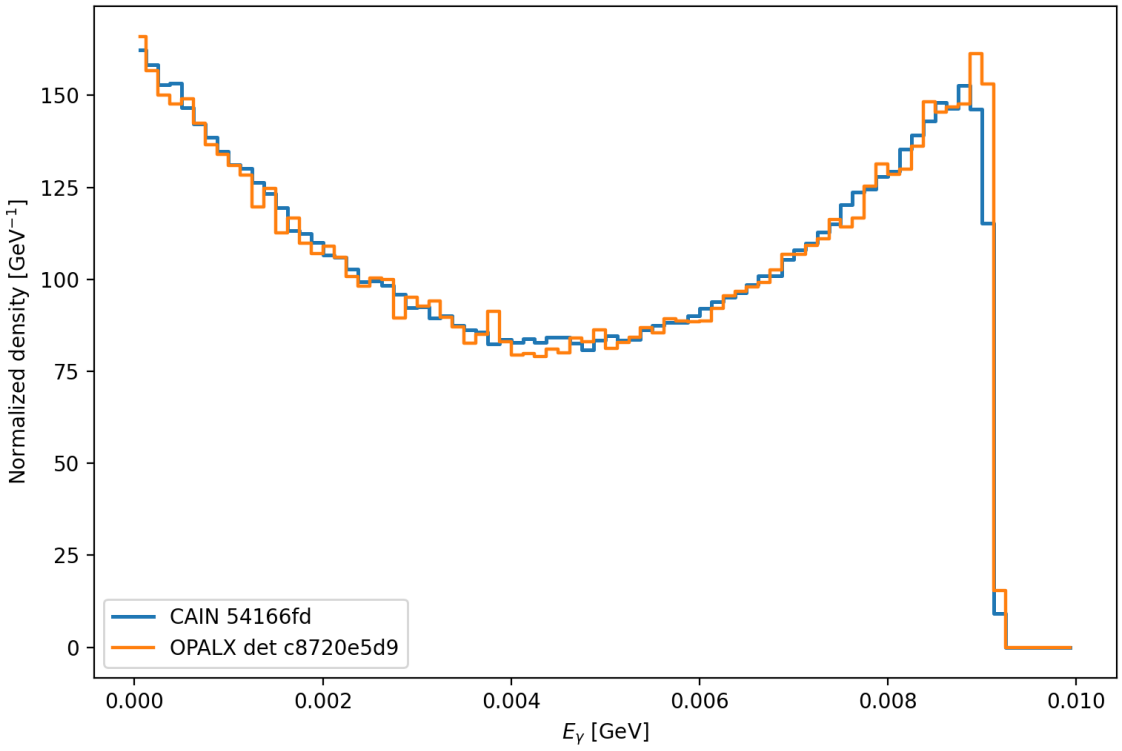


Figure 37.11: Overlap-restricted finite-beam linear-Compton photon energy spectrum. The benchmark keeps the same finite beam and conditions the interaction on the beam-laser overlap through the benchmark Rayleigh and pulse-length scales. The figure compares CAIN and the OPALX Monte Carlo benchmark.

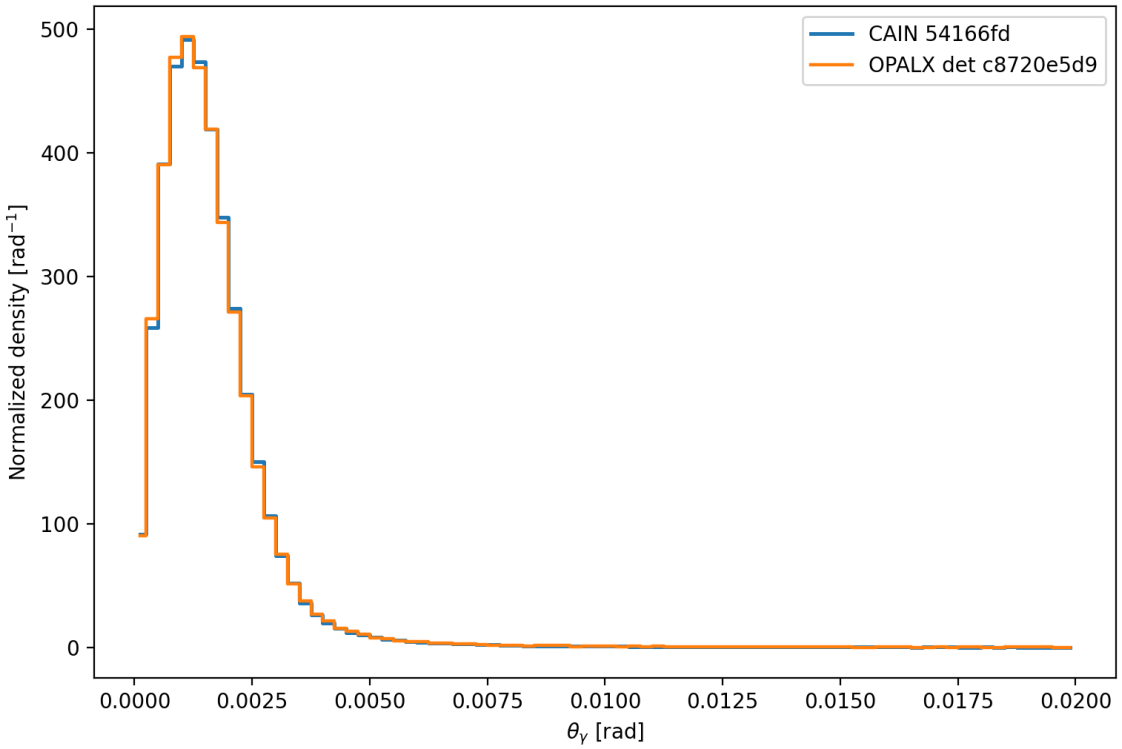


Figure 37.12: Overlap-restricted finite-beam linear-Compton photon laboratory polar-angle spectrum for the same overlap-conditioned benchmark. The figure compares CAIN and the OPALX Monte Carlo benchmark on the common angular histogram grid.

- a built OPALX benchmark executable in the supplied build directory,
- a CAIN executable at `~/git/cain/CAIN-build/cain`, unless overridden by `--cain-bin`,
- the four CAIN decks in `~/git/cain`, unless overridden by `--cain-deck-dir`.

The relevant OPALX build target is:

```
cmake --build /path/to/opalx-laser/build_openmp \
  --target LinearComptonSpectrumBenchmark TestLinearCompton TestLinearComptonSpectrum -j4
```

37.2.6 References

- CAIN linear Compton generator: `src/lncpgn.f`
- CAIN laser geometry helper: `src/lsrgeo.f`
- CAIN manual: User's Manual of CAIN

37.2.7 CAIN Mapping

CAIN evaluates the linear Compton kinematics by boosting the laser photon into the electron rest frame, applying the recoil there, and boosting the scattered photon back to the laboratory frame. In CAIN's `LNCPGN`, the incoming photon energy in the electron rest frame is

$$\omega_1 = \gamma\,\omega_L\,(1 - \beta\cos\alpha). \tag{37.8}$$

For the present benchmark $\cos\alpha = 0$, so

$$\omega_1 = \gamma\,\omega_L. \tag{37.9}$$

This is the CAIN mapping reproduced by the current OPALX helper and benchmark path.

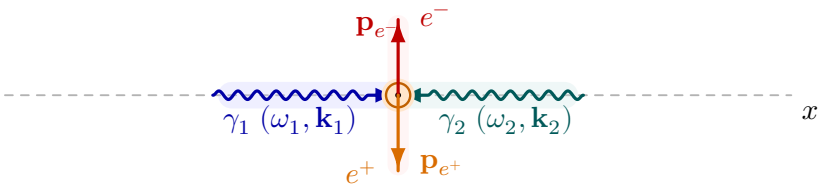


Figure 37.13: Breit-Wheeler scattering.

37.3 Breit-Wheeler Theory

37.3.1 Scope

This note describes the minimum Breit-Wheeler physics model needed for the next *OPALX* benchmark.

We follow closely the CAIN ansatz i.e. manual and source code.

The goal is not yet a full tracking implementation. The immediate target is a host-side Monte Carlo and deterministic benchmark path that reproduces CAIN for the process

$$\gamma + \gamma \to e^- + e^+.$$

The first benchmark will follow the same strategy as the linear ICS work:

- start from a fixed-geometry, unpolarized, linear process;
- implement the core event kernel in;
- validate deterministic and sampled spectra against CAIN;
- delay bunch population and tracker integration until the kernel is trusted.

37.3.2 Linear Breit-Wheeler Kinematics in CAIN

CAIN's linear generator `LNBWGN` uses two incoming photons:

- a laser photon with energy W_1 and direction $\mathbf{n}_1$,
- a high-energy photon with energy W_2 and direction $\mathbf{n}_2$.

Note: in a collider setup the laser photon is also a high-energy photon.

The opening-angle dependence enters through

$$Z_0 = \mathbf{n}_1 \cdot \mathbf{n}_2.$$

The two-photon invariant used in CAIN is

$$S_0 = 2W_1W_2(1 - Z_0). \tag{37.10}$$

Pair creation is kinematically allowed only if

$$S_0 \geq 4m_e^2. \tag{37.11}$$

This threshold appears directly in `LNBWGN`:

```
S0=2*W1*W2*(1-Z0)
IF (4*M**2.GT.S0) GOTO 900
```

In a head-on geometry, $Z_0 = -1$, so the threshold reduces to

$$W_1W_2 \geq m_e^2. \tag{37.12}$$

This is the cleanest geometry for the first benchmark.

CAIN constructs the center-of-momentum boost velocity from the two incoming photon momenta and then samples the final-state scattering variables in that frame.

The code introduces a transformed photon energy W and several auxiliary variables:

$$W = \gamma W_1 V_R, \qquad X = \frac{\sqrt{W^2 - m_e^2}}{W}, \qquad X_1 = \frac{m_e^2}{W^2 + W\sqrt{W^2 - m_e^2}}. \tag{37.13}$$

The random variable Y is used to generate the CM scattering angle through

$$\cos\theta = 1 - 2Y. \tag{37.14}$$

The resulting Mandelstam-like combinations in the CAIN implementation are

$$s_m = \frac{2m_e^2}{(1+X)X_1}\left[X(1-2Y) - 1\right], \qquad u_m = -\frac{2m_e^2}{(1+X)X_1}\left[X(1-2Y) + 1\right]. \tag{37.15}$$

CAIN then builds the angular kernel from

$$\cos\theta_1 = 1 + 2m_e^2\left(\frac{1}{u_m} + \frac{1}{s_m}\right) \tag{37.16}$$

and

$$D = -\left(\frac{s_m}{u_m} + \frac{u_m}{s_m}\right). \tag{37.17}$$

For unpolarized incoming photons, the acceptance weight reduces to the unpolarized part of the full CAIN expression.

37.3.3 OPALX Validation Path

Deterministic physics helper

- ☒ add `Physics::LinearBreitWheeler`
- ☒ implement threshold and invariant helpers
- ☒ implement deterministic spectra from the linear kernel

Host-only sampled event generator

- ☒ fixed seed through `Options::seed`
- ☒ return one sampled electron-positron pair per accepted event
- no tracker, no photon bunch population, no GPU path yet

CAIN comparison

- ☒ generate a CAIN reference deck in the same style as the linear-Compton notes
- ☒ compare OPALX and CAIN histograms first in 1D, later in 2D

37.3.4 Implemented OPALX Benchmark

The current *OPALX* implementation is covered by the generated API reference. The implementation and tests are in these source files:

- src/Physics/LinearBreitWheeler.h
- src/Physics/LinearBreitWheeler.cpp
- unit_tests/Physics/TestLinearBreitWheeler.cpp
- unit_tests/Physics/LinearBreitWheelerBenchmarkCommon.h
- unit_tests/Physics/LinearBreitWheelerBenchmark.cpp
- unit_tests/Physics/TestLinearBreitWheelerSpectrum.cpp
- generate-gamma-gamma-results.sh
- `~/git/cain/linear-breit-wheeler-head-on.i`

- `~/git/cain/generate-linear-breit-wheeler-results.sh`

The benchmark remains deliberately narrow:

- no tracker integration,
- no bunch population,
- no GPU path.

The validated observables are now:

- ☒ electron energy spectrum,
- ☒ positron energy spectrum,
- ☒ electron angle spectrum,
- ☒ positron angle spectrum,
- ☒ joint E vs. θ spectrum for the electron,
- ☒ joint E vs. θ spectrum for the positron.

Finite incoming-photon-beam benchmark:

- ☒ electron energy spectrum with Gaussian photon-beam divergence,
- ☒ positron energy spectrum with Gaussian photon-beam divergence,
- ☒ electron angle spectrum with Gaussian photon-beam divergence,
- ☒ positron angle spectrum with Gaussian photon-beam divergence.

The finite-photon-beam extension is still intentionally narrow:

- the high-energy photon beam is sampled only in momentum space,
- the reference beam axis is the head-on direction,
- the angular spread is Gaussian with $\sigma_{\theta x} = \sigma_{\theta y} = 1\,\mathrm{mrad}$,
- the current published benchmark keeps the photon-beam relative energy spread at zero,
- there is still no transverse position spread or laser-overlap weighting in the *OPALX* benchmark path.

37.3.4.1 Local Kernel Tests

The current *OPALX* implementation now exposes the same proposal-coordinate view used by the CAIN rejection sampler. Two helper functions are validated directly in the unit tests:

- the map from the proposal variable $z \in [-1, 1]$ to the center-of-momentum scattering cosine $\cos\theta$,
- the corresponding unpolarized local angular weight in the same proposal coordinate.

This gives a direct check of the differential kernel itself, not only of its integrated or sampled consequences. The pointwise tests use an independent reference implementation of the CAIN-aligned auxiliary variables $Y(z)$, s_m, u_m, and the local acceptance weight.

37.3.4.2 Finite-Photon-Beam Folding Algorithm

For the finite incoming-photon-beam benchmark, *OPALX* keeps the laser photon fixed and samples only the high-energy photon beam. For each event:

1. a high-energy photon direction is drawn from a Gaussian angular spread around the head-on reference axis,
2. the photon energy is either kept fixed or, in the extended code path, drawn from a Gaussian relative energy spread,
3. a per-event linear Breit-Wheeler sampling kernel is built from that sampled incoming photon state and the fixed laser photon,
4. the electron-positron event is generated with the same host-side rejection sampler used in the fixed-geometry benchmark,
5. the requested observable is histogrammed and compared to the matching CAIN reference.

The present published finite-photon-beam result validates only the divergence broadening. The next benchmark extensions are:

- joint E--θ maps for the finite incoming photon beam,
- finite-photon-beam benchmarks with nonzero photon-beam relative energy spread.

37.3.5 Results of Code Comparison

The current CAIN-backed *OPALX* benchmark compares the linear unpolarized process `LASERQED BREITWHEELER, NPH=0` for a weak-field setup with $\xi = 0.25$. The benchmark observables are:

- electron energy spectrum,
- positron energy spectrum,
- electron angle spectrum,
- positron angle spectrum,
- joint E vs. θ maps for electron and positron,
- and, in the extended benchmark, the same 1D observables for a finite incoming photon beam.

The first fixed-geometry comparison point uses:

- head-on geometry,
- $E_\gamma = 0.5\,\mathrm{GeV}$,
- $\lambda_L = 1\,\mathrm{nm}$,
- unpolarized photons.

With the current *OPALX* Monte Carlo benchmark and the stored CAIN references, the agreement is already tight.

Electron:

- ☒ energy spectrum: mean-energy difference about `0.09%`, L_1 about `0.0179`
- ☒ angle spectrum: mean-angle difference about `0.02%`, L_1 about `0.0176`
- ☒ joint E--θ map: mean-energy difference about `0.10%`, mean-angle difference about `0.02%`, L_1 about `0.0281`

Positron:

- ☒ energy spectrum: mean-energy difference about `0.10%`, L_1 about `0.0160`
- ☒ angle spectrum: mean-angle difference about `0.08%`, L_1 about `0.0173`
- ☒ joint E--θ map: mean-energy difference about `0.10%`, mean-angle difference about `0.08%`, L_1 about `0.0266`

So the present benchmark already supports both 1D and 2D CAIN-backed regression coverage for the linear unpolarized kernel.

Finite incoming-photon-beam point:

- head-on reference geometry,
- $E_\gamma = 0.5\,\mathrm{GeV}$,
- $\lambda_L = 1\,\mathrm{nm}$,
- Gaussian photon-beam divergence with $\sigma_{\theta x} = \sigma_{\theta y} = 1\,\mathrm{mrad}$,
- zero photon-beam energy spread.

Finite-photon-beam agreement:

Electron:

- ☒ energy spectrum: mean-energy difference about `0.12%`, L_1 about `0.0536`
- ☒ angle spectrum: mean-angle difference about `2.49%`, L_1 about `0.0784`
- ☒ joint E--θ map: mean-energy difference about `0.11%`, mean-angle difference about `2.77%`, L_1 about `0.4416`

Positron:

- ☒ energy spectrum: mean-energy difference about `0.13%`, L_1 about `0.0535`
- ☒ angle spectrum: mean-angle difference about `2.89%`, L_1 about `0.0696`
- ☒ joint E--θ map: mean-energy difference about `0.07%`, mean-angle difference about `2.58%`, L_1 about `0.4230`

Finite-photon-beam plus energy-spread agreement:

Electron:

- ☒ energy spectrum with photon-beam relative energy spread `1.0e-3`: mean-energy difference about `0.55%`, L_1 about `0.0573`
- ☒ angle spectrum with photon-beam relative energy spread `1.0e-3`: mean-angle difference about `2.36%`, L_1 about `0.0822`

Positron:

- ☒ energy spectrum with photon-beam relative energy spread `1.0e-3`: mean-energy difference about `0.55%`, L_1 about `0.0601`
- ☒ angle spectrum with photon-beam relative energy spread `1.0e-3`: mean-angle difference about `2.59%`, L_1 about `0.0721`

The finite-photon-beam angle benchmarks are intentionally windowed to $0 \leq \theta \leq 6\,\mathrm{mrad}$, so the stored CAIN angular histograms do not integrate to exactly one inside that clipped range.

The fixed-geometry head-on comparisons are shown for the electron in Figure 37.14, Figure 37.15, and Figure 37.16, and for the positron in Figure 37.25, Figure 37.26, and Figure 37.27.

The finite-photon-beam comparisons with Gaussian divergence and zero energy spread are shown for the electron in Figure 37.17, Figure 37.18, and Figure 37.19, and for the positron in Figure 37.28, Figure 37.29, and Figure 37.30.

The finite-photon-beam plus energy-spread comparisons are shown for the electron in Figure 37.20, Figure 37.21, and Figure 37.22, and for the positron in Figure 37.31, Figure 37.32, and Figure 37.33.

The overlap-restricted comparisons are shown for the electron in Figure 37.23 and Figure 37.24, and for the positron in Figure 37.34 and Figure 37.35.

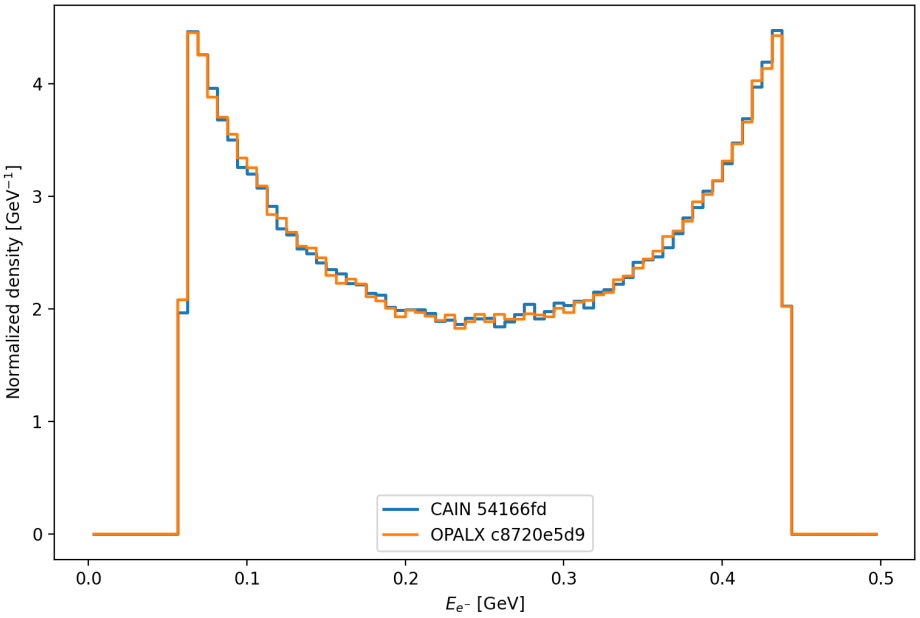


Figure 37.14: Fixed-geometry head-on outgoing electron energy spectrum.

37.3.6 Build and Run

The preferred one-shot workflow is run from the docs tree:

```
cd opalx-manual/physics/gamma-gamma/scripts
./generate-gamma-gamma-results.sh \
  --opalx-build ~/git/opalx-laser/build_openmp
```

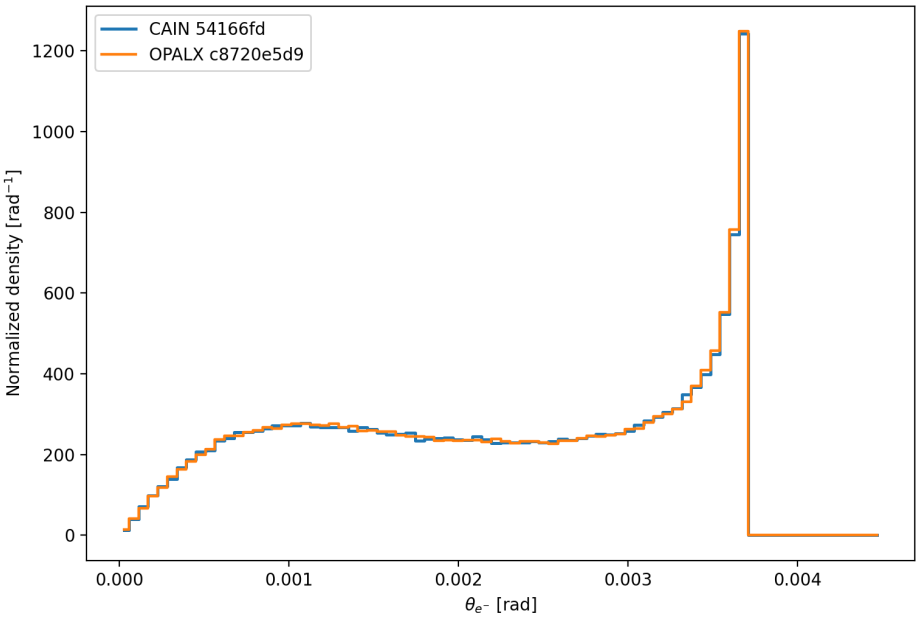


Figure 37.15: Fixed-geometry head-on outgoing electron polar-angle spectrum.

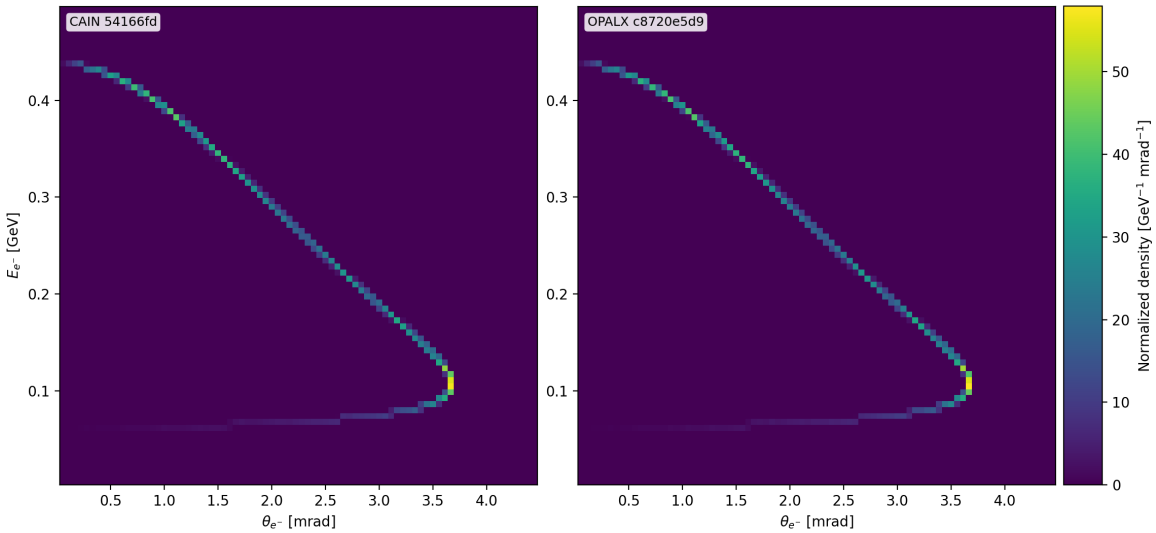


Figure 37.16: Fixed-geometry head-on outgoing electron joint E_{e^-}-- $_{e^-}$ spectrum.

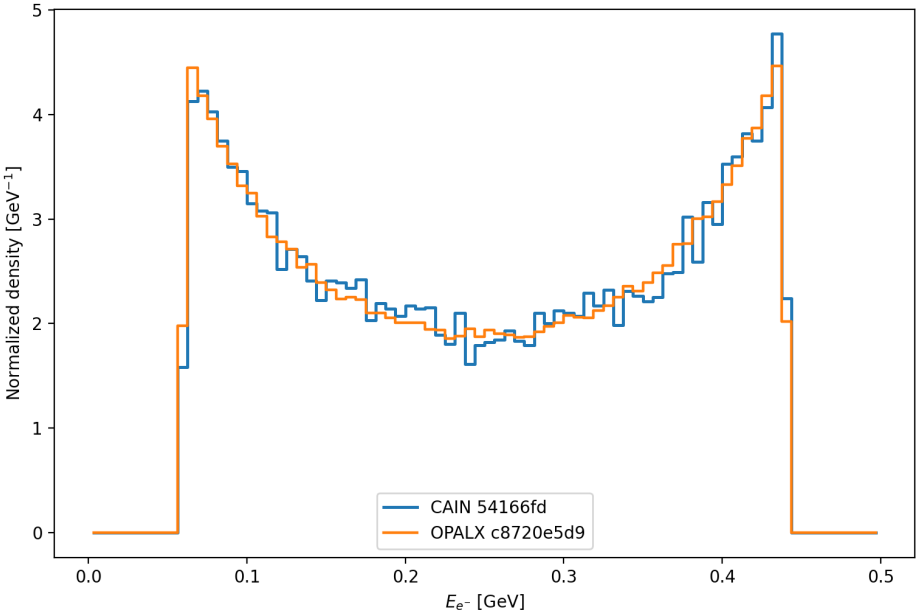


Figure 37.17: Finite-photon-beam outgoing electron energy spectrum.

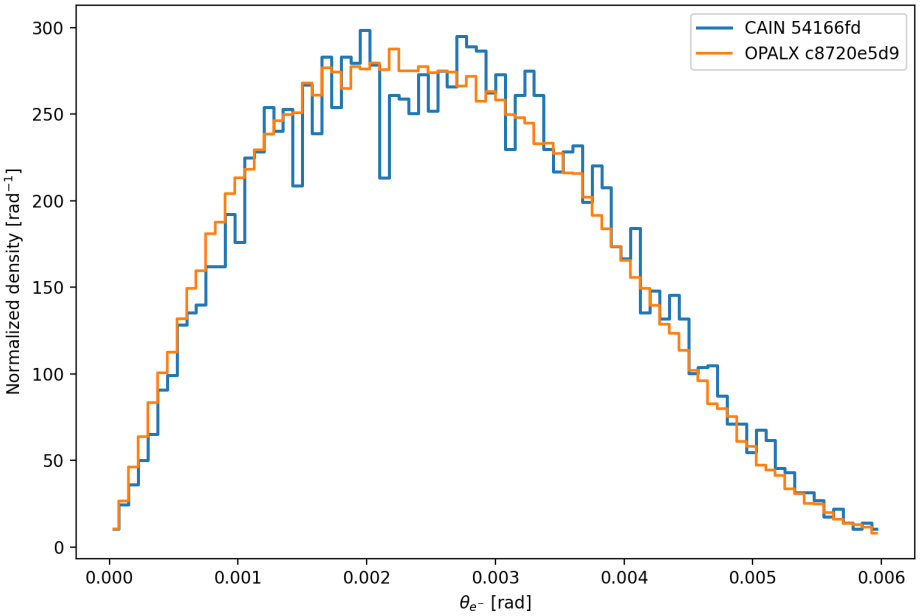


Figure 37.18: Finite-photon-beam outgoing electron polar-angle spectrum.

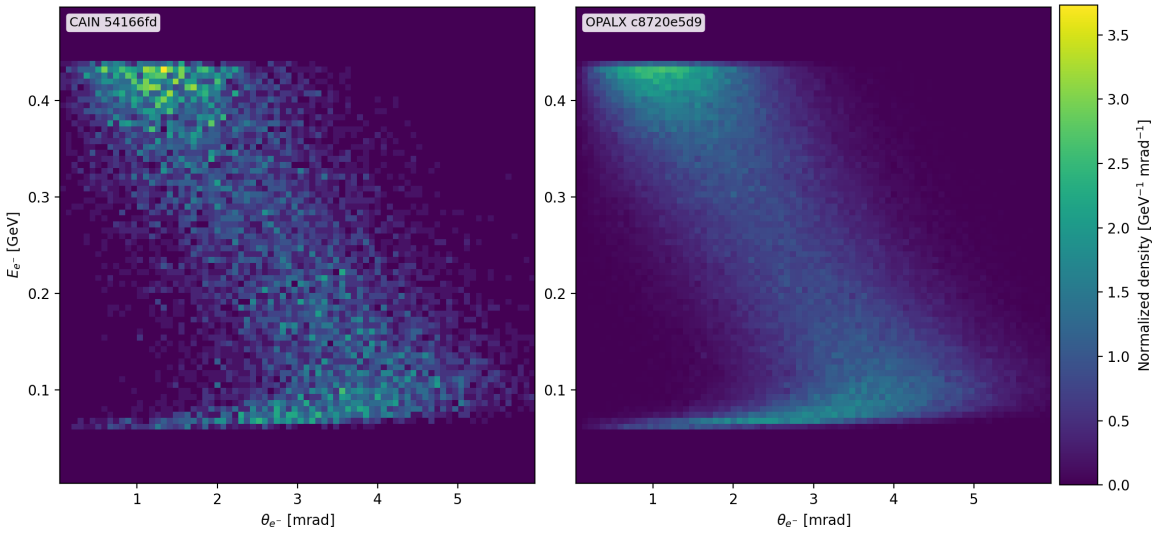


Figure 37.19: Finite-photon-beam outgoing electron joint E_{e^-}-- $_{e^-}$ spectrum.

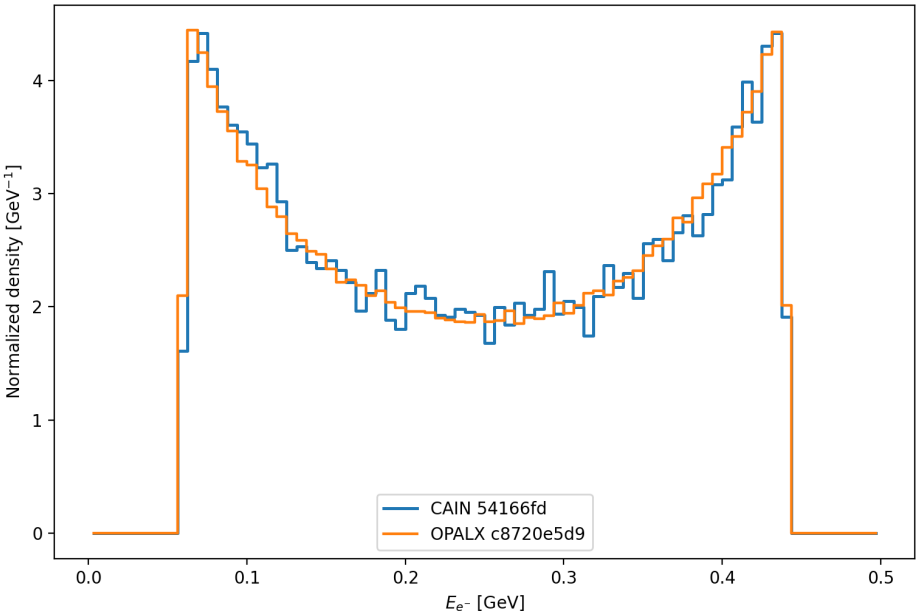


Figure 37.20: Finite-photon-beam plus energy-spread outgoing electron energy spectrum.

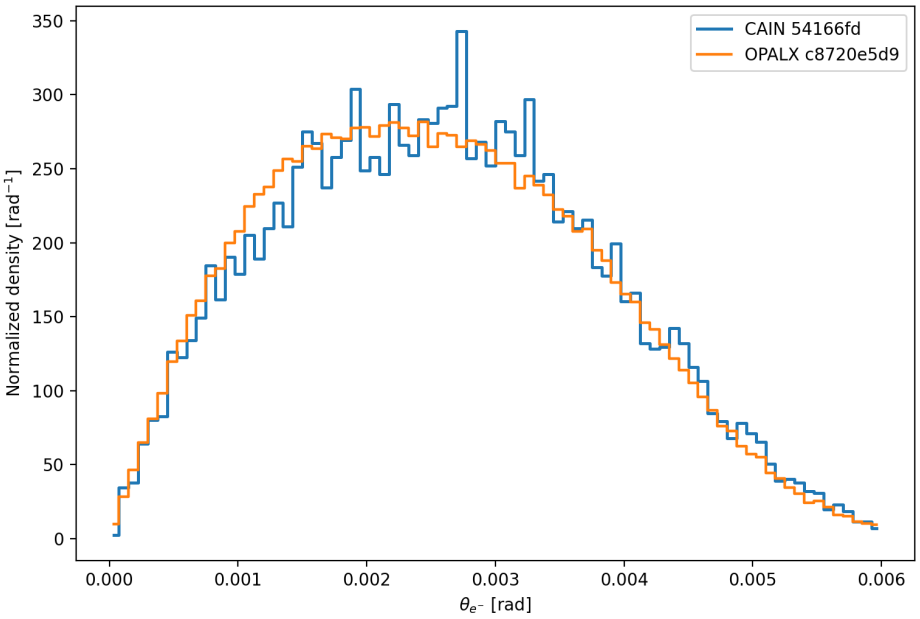


Figure 37.21: Finite-photon-beam plus energy-spread outgoing electron polar-angle spectrum.

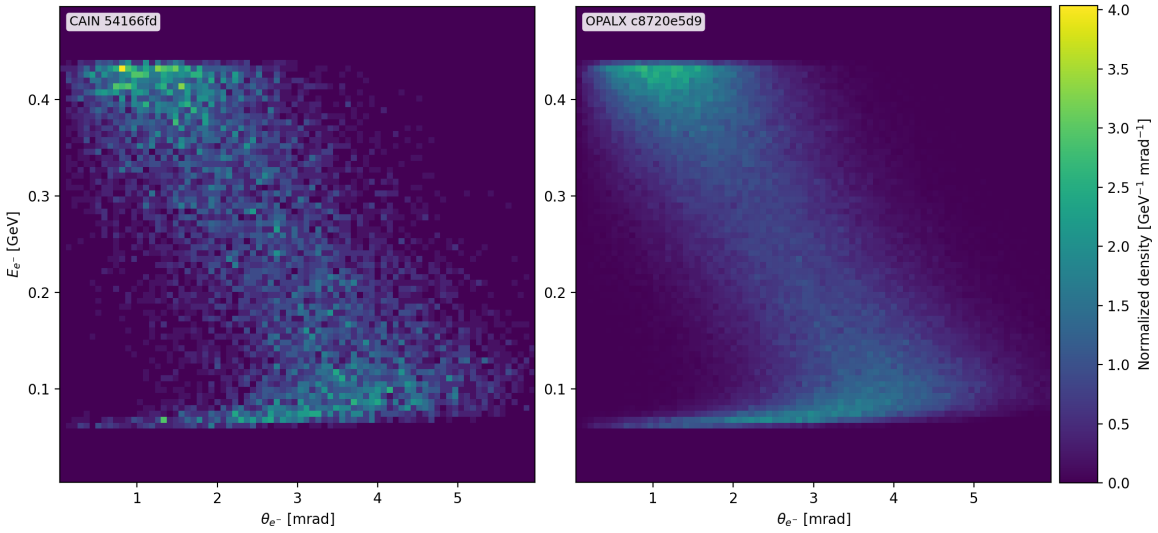


Figure 37.22: Finite-photon-beam plus energy-spread outgoing electron joint E_{e^-}-- $_{e^-}$ spectrum.

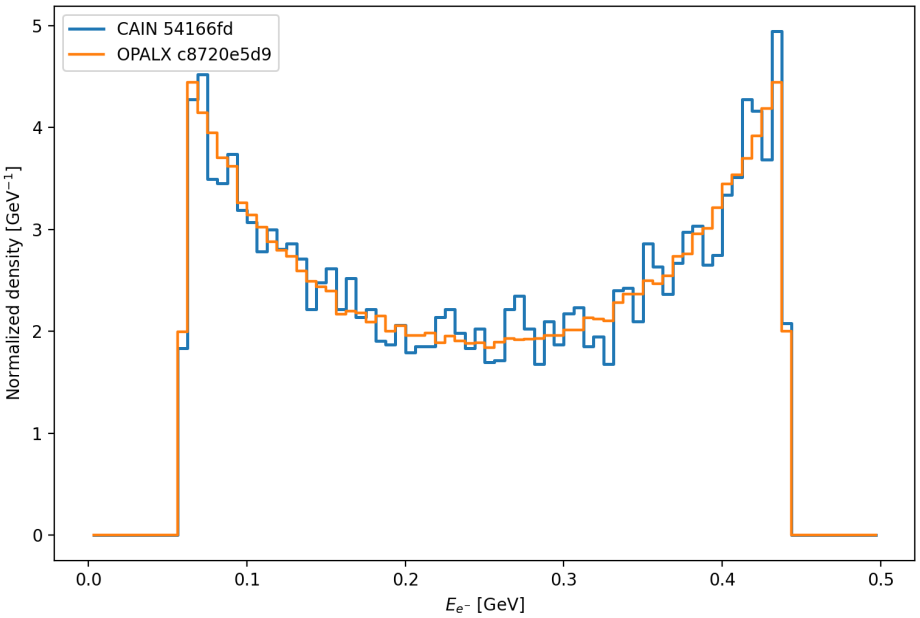


Figure 37.23: Overlap-restricted outgoing electron energy spectrum.

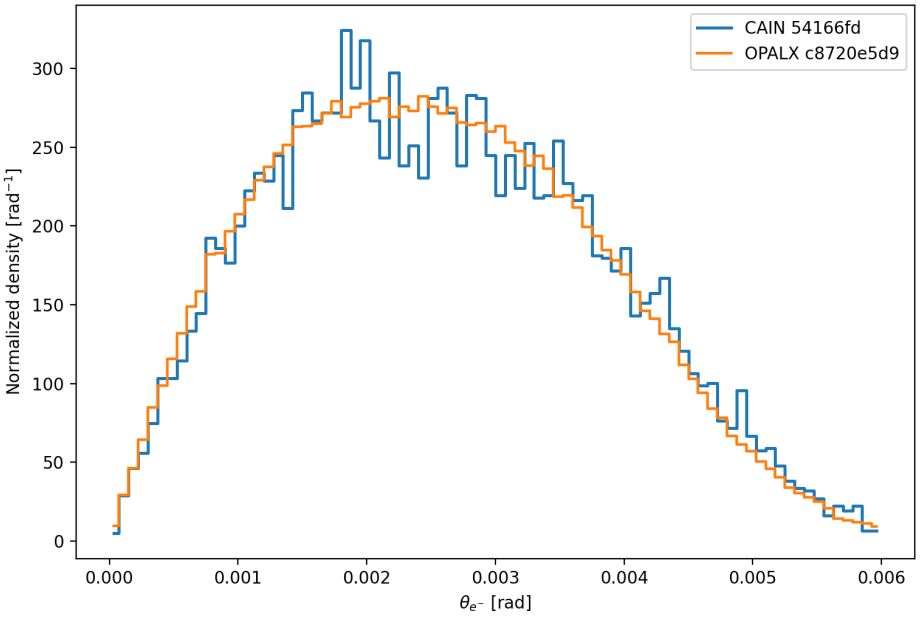


Figure 37.24: Overlap-restricted outgoing electron polar-angle spectrum.

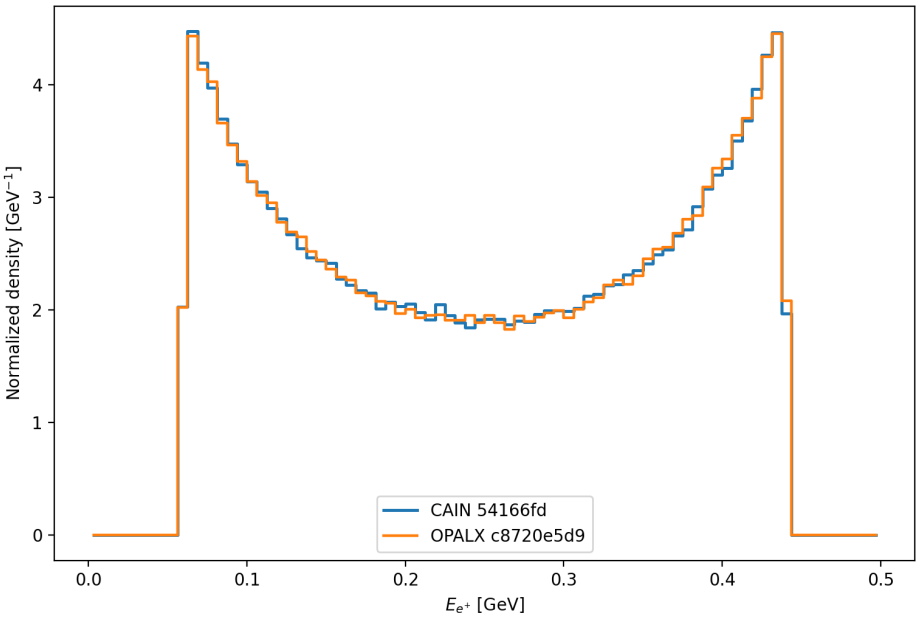


Figure 37.25: Fixed-geometry head-on outgoing positron energy spectrum.

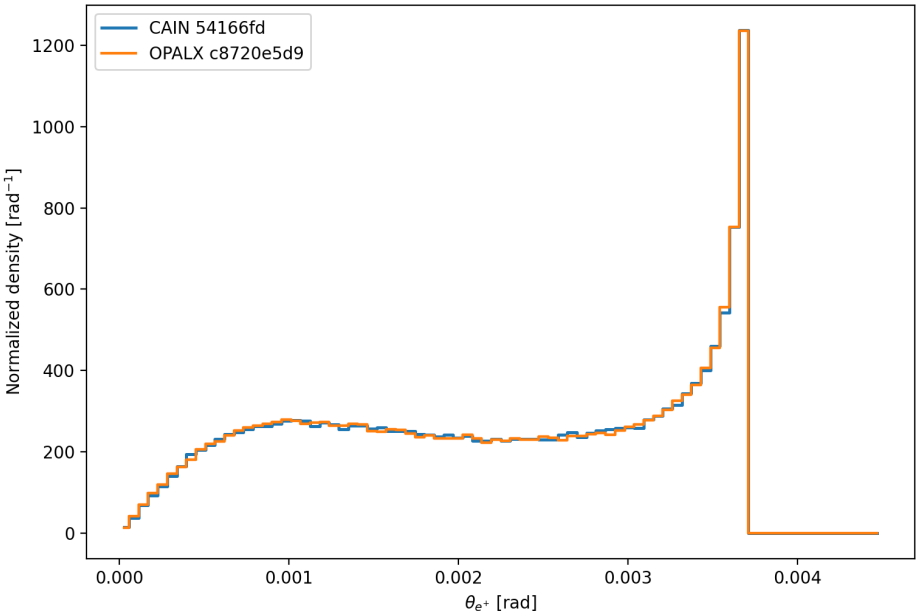


Figure 37.26: Fixed-geometry head-on outgoing positron polar-angle spectrum.

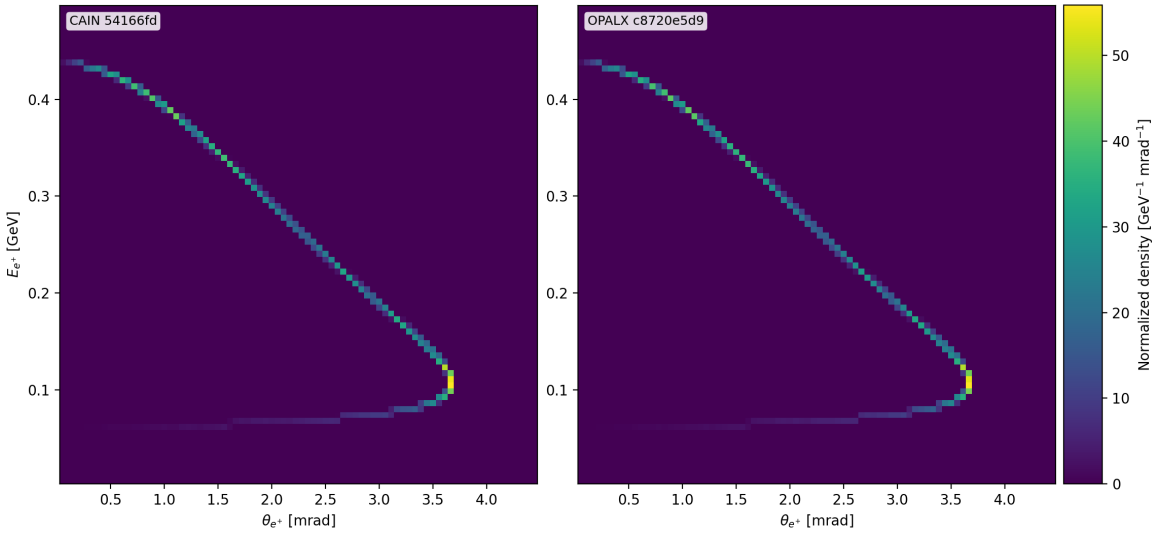


Figure 37.27: Fixed-geometry head-on outgoing positron joint E_{e+}-- $_{e+}$ spectrum.

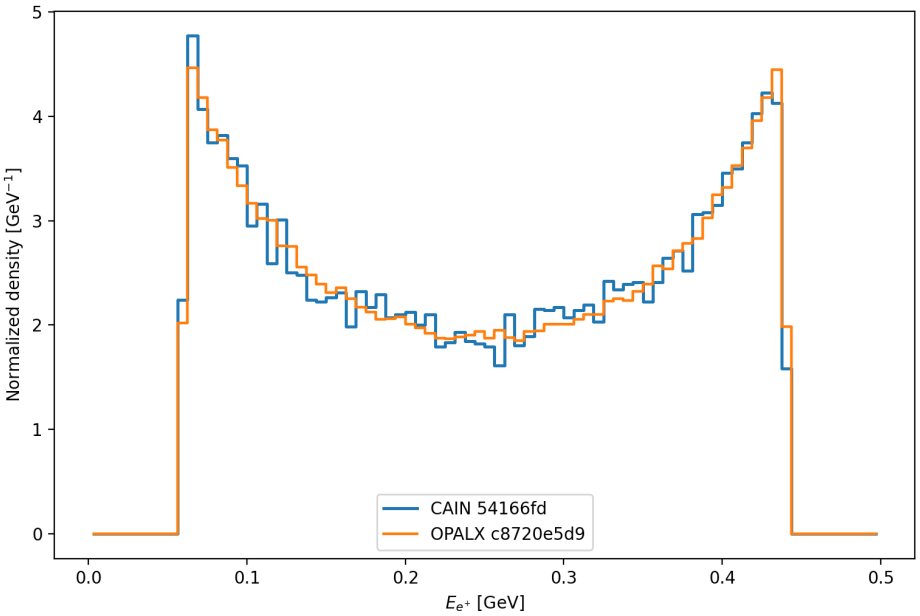


Figure 37.28: Finite-photon-beam outgoing positron energy spectrum.

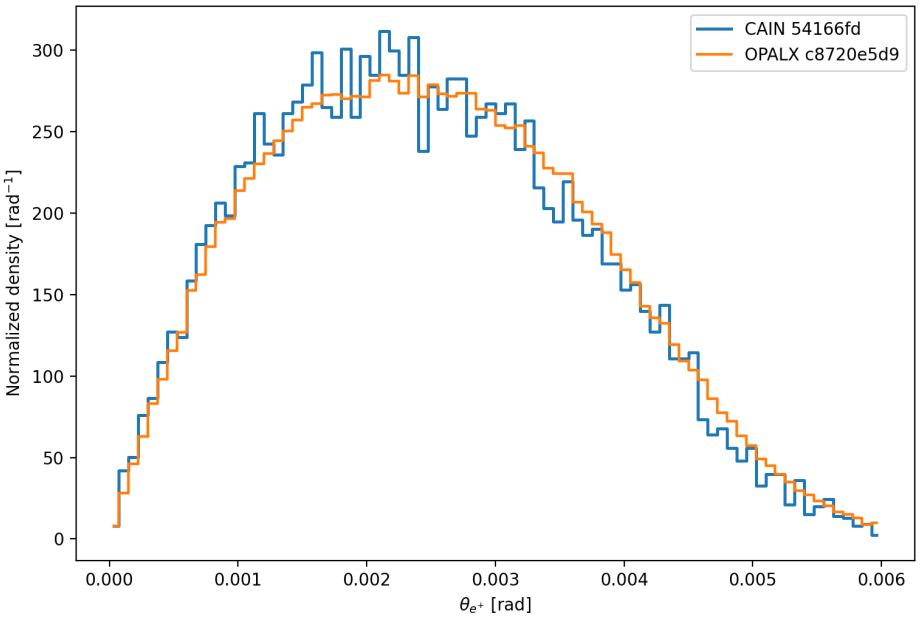


Figure 37.29: Finite-photon-beam outgoing positron polar-angle spectrum.

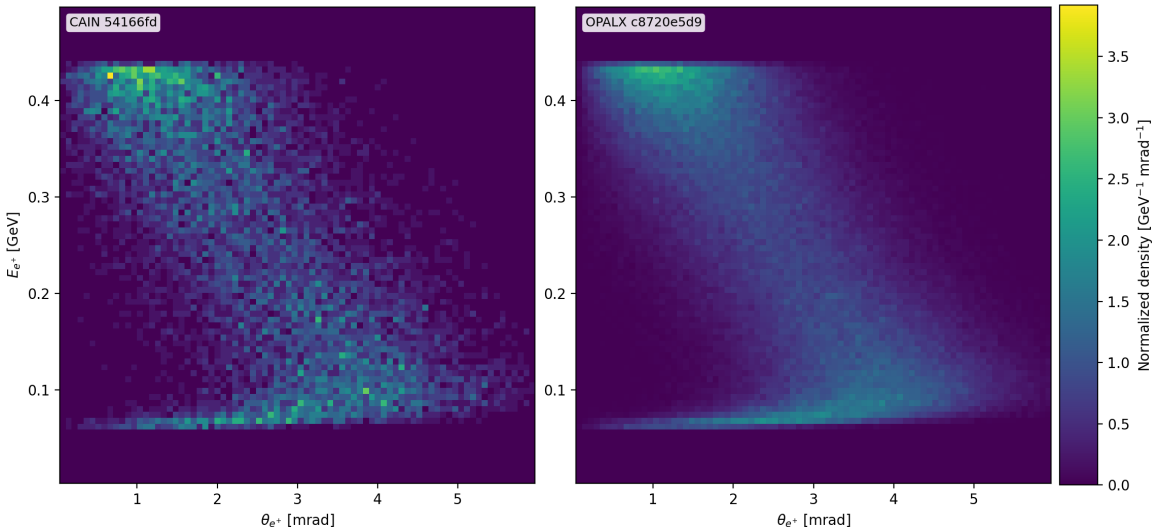


Figure 37.30: Finite-photon-beam outgoing positron joint E_{e+}-- $_{e+}$ spectrum.

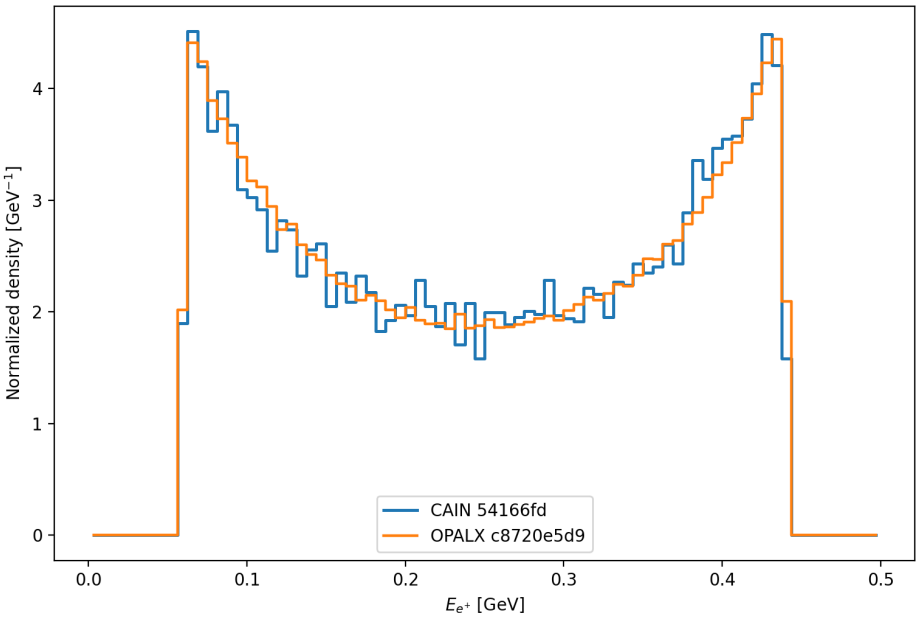


Figure 37.31: Finite-photon-beam plus energy-spread outgoing positron energy spectrum.

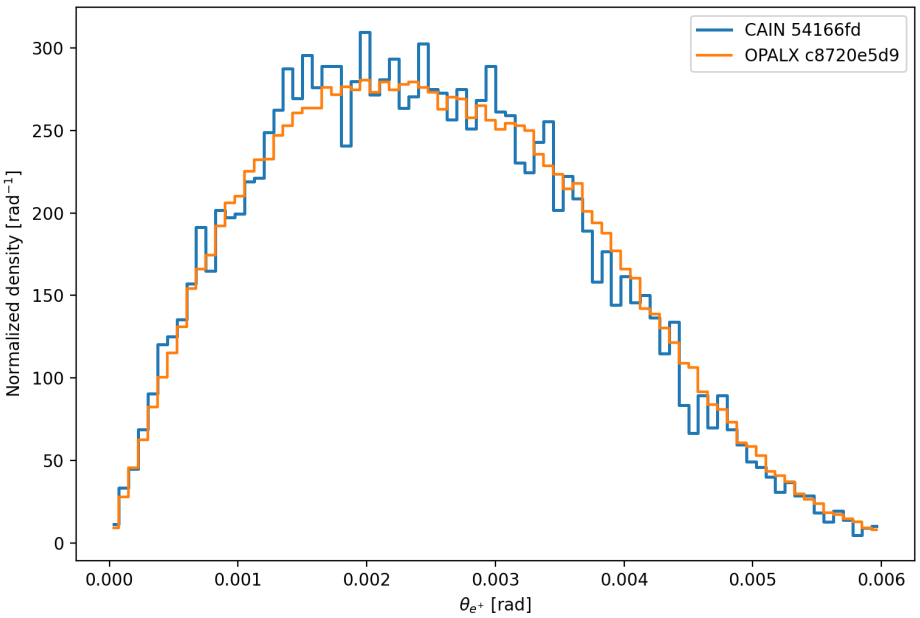


Figure 37.32: Finite-photon-beam plus energy-spread outgoing positron polar-angle spectrum.

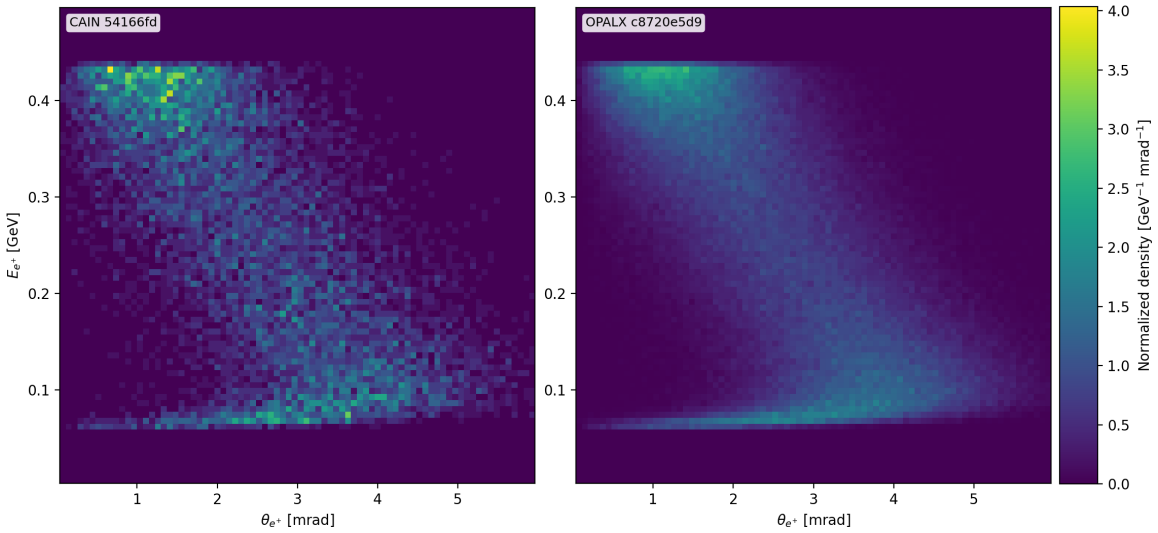


Figure 37.33: Finite-photon-beam plus energy-spread outgoing positron joint E_{e+}-- $_{e+}$ spectrum.

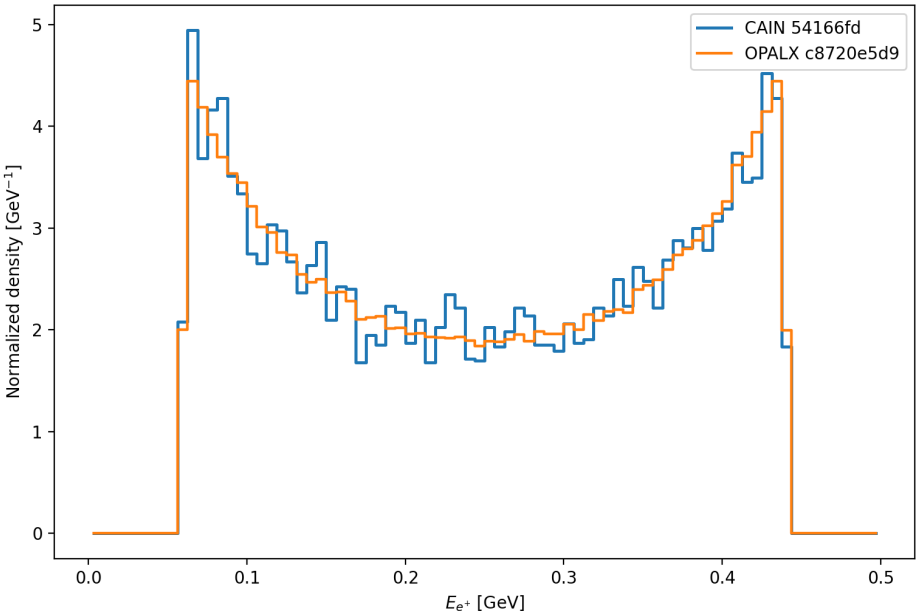


Figure 37.34: Overlap-restricted outgoing positron energy spectrum.

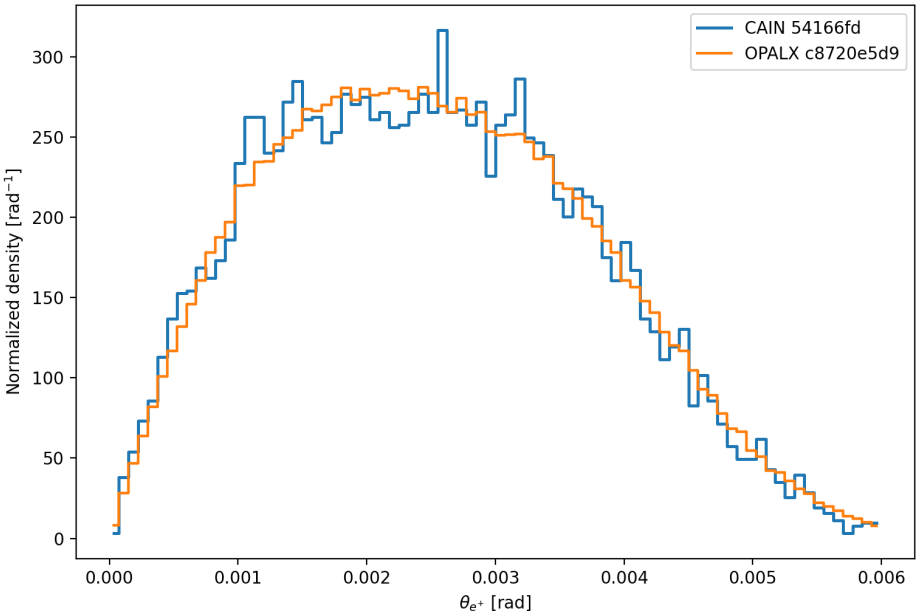


Figure 37.35: Overlap-restricted outgoing positron polar-angle spectrum.

This top-level driver:

- builds the OPALX gamma-gamma benchmark targets,
- runs the inverse-Compton and Breit-Wheeler unit/regression suites,
- regenerates the CAIN and OPALX benchmark data for both notes,
- republishes the comparison figures, and
- renders the gamma-gamma note pages.

If only the Breit-Wheeler assets need to be refreshed, use:

```
cd ~/git/cain
./generate-linear-breit-wheeler-results.sh \
  --opalx-build ~/git/opalx-laser/build_openmp
```

This script:

- runs the CAIN decks `~/git/cain/linear-breit-wheeler-head-on.i`, `~/git/cain/linear-breit-wheeler-finite-photon-beam.i`, `~/git/cain/linear-breit-wheeler-finite-photon-beam-energy-spread.i`, and `~/git/cain/linear-breit-wheeler-overlap.i`,
- regenerates the stored CAIN histogram references in `~/git/opalx-laser/unit_tests/Physics/data`,

- runs the OPALX benchmark executable `LinearBreitWheelerBenchmark` for fixed-geometry, finite-photon-beam, finite-photon-beam-plus-energy-spread, and overlap cases,
- regenerates the 1D and 2D comparison plots in `~/git/cain/reference-data`.

The script expects:

- a built OPALX benchmark executable in the supplied build directory,
- a CAIN executable at `~/git/cain/CAIN-build/cain`, unless overridden by `--cain-bin`.

The relevant unit-test targets are:

```
cmake --build ~/git/opalx-laser/build_openmp \
  --target TestLinearBreitWheeler TestLinearBreitWheelerSpectrum LinearBreitWheelerBenchma

~/git/opalx-laser/build_openmp/unit_tests/Physics/TestLinearBreitWheeler
~/git/opalx-laser/build_openmp/unit_tests/Physics/TestLinearBreitWheelerSpectrum
```

37.3.7 References

- CAIN source parser: `src/rdlqed.f`
- CAIN Breit-Wheeler runtime loop: `src/lsrqedbw.f`
- CAIN linear Breit-Wheeler generator: `src/lnbwgn.f`
- CAIN nonlinear Breit-Wheeler generator: `src/nlbwgn.f`
- CAIN weighted event insertion: `src/lbwevent.f`
- CAIN manual: User's Manual of CAIN

37.3.8 CAIN Structure

CAIN separates Breit-Wheeler physics into three layers.

Input and model selection

- `LASERQED BREITWHEELER` is parsed in `src/rdlqed.f`.
- `NPH = 0` selects the linear Breit-Wheeler model.
- `NPH >= 1` selects the nonlinear model.

Runtime driver

- The laser-particle loop is handled by `src/lsrqedbw.f`.
- Local laser geometry and power density are obtained from `LSRGEO`.
- The linear event generator is `LNBWGN`.
- The nonlinear event generator is `NLBWGN`.
- If an event happens, the produced electron and positron are inserted by `LBWEVENT`.

Event generation

- Linear Breit-Wheeler: `src/lnbwgn.f`
- Nonlinear Breit-Wheeler: `src/nlbwgn.f`
- Weighted particle insertion: `src/lbwevent.f`

For the first *OPALX* benchmark only the linear event kernel is needed.

38 Sandbox

This chapter is reserved for exploratory notes, draft diagrams, implementation sketches, and benchmark-driven documentation that is not yet ready to be folded into the formal theory chapters.

The intended use is:

- keep work-in-progress material visible in the rendered manual,
- document code-facing conventions before promoting them into the final text,
- collect diagrams and benchmark notes that support later physics sections.

38.1 OPAL - OPALX Redesign notes

This is where work in progress is shown to the comunity.

10.1.1 are notes and comments to ideas from Jon/Chris.

10.2 - 10.6 expain implmentations alread in *391-placement-of-sbend-and-rbend-analytic-fields-including-fringe-fields*.

38.1.1 Comments to the proposels of Jon/Chris

Jon's OPAL element layout proposal.

38.1.1.1 OpalElemen vs. Element / ElemenBase

Could OpalElement absorb Element? Technically possible only with a broader refactor, but not directly, because Line and Sequence also depend on Element. We would need to move Element's responsibilities somewhere else.

Could OpalElement absorb ElementBase? That would couple parser objects to the tracking/beamline runtime model and force either parser classes to implement geometry/tracking/visitor APIs or runtime classes to become OPAL directory objects. That would make ownership, cloning, sharing, and beamline traversal substantially worse.

38.1.1.2 TBeamline is a template and has std::list as base class. Is it safe to use std::list as base?

Short answer: it is legal C++, but it is, indeed, not a good or safe design.

A better shape would be composition:

```
template <class T>
class TBeamline : public Beamline {
public:
    using container_type = std::list<T>;
    using iterator = container_type::iterator;
    using const_iterator = container_type::const_iterator;

    iterator begin();
    iterator end();
    const_iterator begin() const;
    const_iterator end() const;

    void append(const T&);
    void prepend(const T&);
    iterator erase(iterator);
    iterator insert(iterator, const T&);

private:
    std::list<T> elements_;
};
```

That preserves most call-site ergonomics while giving `TBeamline` control over mutation.

38.1.1.3 The whole Beamline, TBeamlinem TLine and FlaggedBeamline Hierarchie is complext. Is it nessesary?

Mostly no: the concepts are necessary, but the current hierarchy is more complex than it needs to be.

What is actually needed:

- A runtime beamline object that is an `ElementBase`, so a beamline can appear anywhere an element can appear. That is `Beamline`.
- An ordered container of beamline entries. That is currently `TBeamline<T>`.
- A beamline entry type that points to an `ElementBase`. That is `ElmPtr`.
- Extra entry metadata for `LINE`: reflected/selected flags. That is `FlaggedElmPtr`.
- Extra entry metadata for `SEQUENCE`: position, relation to previous/next/begin/end, generated drift info, parser-side `Element` pointer. That is `SequenceMember`.

So the data model is defensible. But the implementation is too indirect:

The weak part is not that `LINE` and `SEQUENCE` need different metadata. They do. The weak part is encoding this through public inheritance from `std::list<T>` plus several inherited wrapper objects.

A simpler design would be:

```
class Beamline : public ElementBase {
public:
    using EntryList = std::list<BeamlineEntry>;

    iterator begin();
    iterator end();

    void append(BeamlineEntry);
    void insert(iterator, BeamlineEntry);
    iterator erase(iterator);

private:
    EntryList entries_;
};
```

with one entry type:

```
struct BeamlineEntry {
    std::shared_ptr<ElementBase> element;

    bool reflected = false;
    bool selected = false;

    std::optional<SequencePlacement> sequencePlacement;
    std::shared_ptr<Element> parserElement;
};
```

or, cleaner, separate concrete types:

```
BeamlineBase
  -> LineBeamline      owns list<LineEntry>
  -> SequenceBeamline  owns list<SequenceEntry>
```

We could keep the conceptual split between runtime beamline, line entries, and sequence entries. But remove `TBeamline<T> : public std::list<T>` over time. `TLine` and `FlaggedBeamline` are not really independent abstractions; they are type aliases created because `TBeamline` is templated. That is a sign the implementation is serving the template, not the model.

38.1.1.4 Three more remarks w.r.t. Element Placing, Origin etc

See OPALX has 4 layers of structure in the next Sandbox chapter for the related structure and class-level context.

38.1.1.5 Why are bend element split into seperate loops in the compute3DLattice function?

This answer already is base on the branch `391-placement-of-sbend-and-rbend-analytic-fields-including-fringe-fields`.

They are split because bends need two different pieces of placement state:

1. The bend body/entry pose must be stored explicitly.

   In the first loop in `OpalBeamline.cpp`, the code scans only unpositioned bends. For each `SBEND`/`RBEND`, it computes the desired entry pose, then converts that into the element body pose:

   ```
   const CoordinateSystemTrafo desiredEntryPose =
           fromEndLastToBeginThis * currentCoordTrafo;
   const CoordinateSystemTrafo bodyFromEntry = element->getEdgeToBegin().inverted();
   setNominalPlacement(element, bodyFromEntry * desiredEntryPose);
   ```

   That is bend-specific because a bend's useful reference frame is not just "translate by length along z". It has chord length, arc length, bend angle, entrance angle, rotation axis, and sometimes a true design path.

2. The second loop advances the global beamline chain for all elements.

   The second loop then walks every element in beamline order. Straight elements get placed directly with `setNominalPlacement(...)`. Bends are already positioned by the first loop, so the second loop uses them mainly to advance `currentCoordTrafo` and `endPriorPathLength` across the curved/chord geometry.

So the split is basically a compatibility bridge:

- first pass: pre-place bends using bend-entry/body geometry;
- second pass: place straight elements and advance the running lattice transform through both straight and bend elements.

Is it necessary structurally? Probably not. It is necessary for the current implementation because bend placement is side-effectful and special-cased. A cleaner design would have one placement algorithm where every element exposes:

```
entryPort()
exitPort()
bodyPoseFromEntryPose()
advanceReferencePose()
```

Then bends and straight elements would differ behind the element interface, not through separate loops in `OpalBeamline`. The current split is a symptom that bend geometry is not yet fully hidden behind a uniform placement/port contract. And I guess is a hint that we need to think about this in a very general way … as we do now!

38.1.1.6 The bend element

The current enum checks should be treated as legacy compatibility logic, not as the design model. The code often does this pattern:

```
if (element->getType() == ElementType::SBEND ||
    element->getType() == ElementType::RBEND) {
    ...
}
```

and then applies bend-specific logic. That is brittle because it confuses two questions:

1. Is this element named/type-tagged as `SBEND` or `RBEND`?
2. Does this element provide bend geometry/placement behavior?

Those are not the same. `MultipoleT::bends()` can return true when `bendAngle_m != 0.0`, while old-style `Multipole::bends()` returns false. `ScalingFFAMagnet` and `Ring` also bend. So `ElementType::SBEND || ElementType::RBEND` is not a general "bending element" test. It is really a test for "OPALX analytic `BendBase` element with the current SBEND/RBEND placement model."

The better design is capability-based:

```
if (auto* bend = dynamic_cast<BendBase*>(element.get())) {
    // use BendBase geometry API
}
```

or better still, avoid the branch entirely by giving every element a common placement/port API:

```
element->getEntryPort();
element->getExitPort();
element->advanceReferencePose(...);
element->getPlacementGeometry();
```

Then `SBEND`, `RBEND`, curved `MultipoleT`, straight elements, and future elements implement their own geometry. `OpalBeamline` should not need to know the enum except for diagnostics, serialization, or legacy compatibility.

So the comment is absolutly valid we have to replace these checks with capability-based polymorphism: either a `BendBase` interface for analytic bend-specific operations, or preferably

a general placement/port interface implemented by all elements. This avoids hard-coded type lists and makes the placement algorithm open to new element classes.

38.1.1.7 Nested LINE statements will call recursive calls

Yes. A named nested `LINE` will cause recursive visitor calls.

There is an important distinction in `Line::parseList()`:

- Anonymous nested lists like `(A, B, C)` are flattened immediately into the parent line.
- Named `LINE` objects are inserted as an `ElementBase*` pointing to their own `FlaggedBeamline`, so they remain nested and are expanded later by recursive visitor traversal.

This is another reason the `Beamline / TBeamline / FlaggedBeamline / visitor` design is hard to reason about: structural nesting is not normalized in one place. Some nesting is flattened during parsing, while named line nesting is expanded during tracking.

There may also be a real behavioral risk: `DefaultVisitor::visitFlaggedElmPtr()` toggles `local_flip` for reflected members, but `ParallelTracker::visitBeamline()` currently does:

```
fbl->iterate(*this, false);
```

It ignores `local_flip`. So a reflected named nested line may not be traversed in reverse by `ParallelTracker`, even though the generic `DefaultVisitor::visitBeamline()` would honor `local_flip`. That is worth checking with a small nested/reflected `LINE` test.

38.1.1.8 Proposal - Layout Schemes

The proposal is useful because it names three user-visible layout policies that old OPAL treated inconsistently:

1. offset longitudinal placement using `ELEMEDGE`,
2. relative longitudinal placement, the Lego-block model,
3. beamline-offset placement using `X`, `Y`, `Z`, `THETA`, `PHI`, and `PSI`.

However, we should not make these three policies the primary architecture. The project direction is the more general Chapter 2 placement model: elements have placements, ports, nominal/actual poses, and possibly a separate field-local or reference-orbit chart. The current branch `391-placement-of-sbend-and-rbend-analytic-fields-including-fringe-fields` already moves in that direction through `PlacedElement`, `PlacementPose`, `Misalignment`, entry/body/exit ports, and explicit field-local transforms for analytic bends.

So the three layout schemes are best understood as **input and compatibility policies** that produce the same internal placement representation. They should not become three

separate internal geometry mechanisms. This is where the proposal is less general than Chapter 2: it is still centered on arranging a `LINE`, while the Chapter 2 model is about geometric placement of elements and ports in space, independent of the particular input syntax that produced the placement.

Please see General Structure, OpalElement as Parser-to-Runtime Bridge, and Element Placing (Posing) for the design route we will follow.

38.1.1.9 Other Aims

Jon's proposal also lists several implementation aims beyond the three layout modes. Several of them are already discussed above and are not repeated here: `Element`/`ElementBase`, `TBeamline<T> : public std::list<T>`, the `Beamline` hierarchy, bend enum checks, nested `LINE` traversal, and the repeated bend/non-bend placement scans.

The remaining cross-cutting aims are valid and useful:

- Support the three layout schemes through one unified internal placement representation, rather than splitting responsibility between parser objects, `OpalBeamline`, and `Ring`.
- Remove the visitor pattern from layout, or at least keep it out of the geometry decision path. This should simplify the tracker inheritance tree and reduce the amount of element-type-specific visitor boilerplate.

The important qualification is that the unified representation should be the Chapter 2 placement/port model, not a simpler line-list layout model. The line syntax is only one way to produce a placed beamline. The internal model must also support curved elements, combined-function elements, misalignments, nominal-versus-actual poses, reference-path threading, and field-local charts.

38.1.1.10 Proposal Input file syntax modifications

1. All comments w.r.t LINE and ORIGIN/ORIENTATION are already realized
2. CLOSED and CLOSED_TOLERANCE is a nessesary feature but maybe can also added to TRACK
3. LAYOUT_MODE again TRACK is maybe an alternative for adding such an attribute
4. RINDEFINITION is already out

Note: line is maybe problematic, suppose you have a injection line to your ring. the line gets flatten so where is now CLOSED etc. to be applied? You could have 3 tracks one for injection, then the ring and then exptraction. So the me this decoration make sense on the TRACK (t.b.f.d).

38.1.1.11 Notes to implementation sketches

–> still working on this one

The implementation sketch in Jon's proposal suggests a single `layout()` flow:

```
ParallelTTracker
  -> FlaggedBeamline::layout()
  -> OpalBeamline::startLayout()
  -> element.layout(...)
  -> element.addField(this) when applicable
  -> geometry determines the next pose
  -> OpalBeamline::endLayout()
```

This is useful as a critique of the current code: it points out that layout, field-list construction, nested-line state, and element-specific behavior are currently entangled through visitor traversal and `OpalBeamline` side effects. The proposal is also right that carrying a small layout-state stack would be better than stashing and merging whole `OpalBeamline` objects.

Compared with branch `391-placement-of-sbend-and-rbend-analytic-fields-including-fringe-fields`:

- The branch does not introduce `layout()`, `startLayout()`, or `endLayout()`. Layout still enters through `ParallelTracker::prepareSections()`, `ParallelTracker::visitBeamline()`, visitor dispatch, and `OpalBeamline::visit<T>()`.
- Elements do not add themselves to the field list. `OpalBeamline::visit<T>()` still clones the visited element, initializes it, and pushes a `BeamlineFieldElement` into `elements_m`.
- The visitor pattern remains part of layout collection. The branch has not moved layout responsibility fully into `OpalBeamline` plus element geometry.
- The branch still sorts and prepares a `FieldList`; it also keeps `declaredOrder_m` and `placementAssembly_m` to preserve both tracking order and placement records.
- `compute3DLattice()` has been renamed in spirit to `compileCompatibilityPlacement()`, with `compute3DLattice()` now just forwarding to it. This is useful: it makes the old routine look like a compatibility conversion from legacy `ELEMEDGE` placement into explicit nominal poses.
- The proposal wanted one layout loop. The branch still uses two placement loops inside `compileCompatibilityPlacement()`: a bend pre-pass and then an all-elements pass. This is less than the old scattered model, but it is not the proposed single polymorphic layout pass.
- The proposal wanted nested `LINE` handling through a stack of position/orientation state. The branch still has the old `OpalBeamline` stash/merge code in `ParallelTracker::visitBeamline()`, but it is commented out; the active code simply calls `fbl->iterate(*this, false)`. So this part is unresolved rather than implemented.

- The proposal wanted explicit enum tests removed. The branch still checks `ElementType::SBEND`, `RBEND`, `RBEND3D`, `SOURCE`, and `MARKER` in several places. Some of those checks are followed by `dynamic_cast<BendBase*>`, which is closer to a capability interface, but the dispatch is still enum-driven.
- The proposal wanted the geometry object to determine the next pose. The branch has moved toward this with `PlacedElement`, entry/body/exit ports, `getPlacementGeometry()`, and `getNominalEntryTransform()` / `getNominalExitTransform()`. However, bend advance logic is still hard-coded in `OpalBeamline::compileCompatibilityPlacement()` instead of being hidden behind a uniform element/geometry interface.
- The proposal wanted beamline-relative placement logic moved out of user-interface objects. The branch partially keeps the bridge: `OpalElement::update()` still interprets `ORIGIN`, `ORIENTATION`, `X`, `Y`, `Z`, `THETA`, `PHI`, `PSI`, stores placement state on the runtime element, and may fix the position.
- The proposal's suggestion that `OpalBeamline` may deserve a different name is still relevant. In this branch it is increasingly a runtime field/layout assembly object, not an `OpalElement`-style parser class. A name like `BeamlineLayout` or `BeamlineAssembly` would better describe the role.

Where the proposal is narrower than the Chapter 2 model:

- It treats the line layout problem as the central abstraction. Chapter 2 treats placement, ports, nominal/actual transforms, and reference charts as the central abstraction. That is more general and is the direction we should keep.
- It does not clearly separate body placement from field-local coordinates. This distinction matters for analytic `SBEND`/`RBEND` and will matter for other curved or combined-function elements.
- It does not explicitly cover nominal versus actual poses and local misalignment corrections. Those are first-class concepts in the Chapter 2 model and in the current placement bridge.
- It suggests element-driven `addField(this)` behavior, which may reduce visitor boilerplate but risks moving beamline assembly policy into every element. We should be careful not to replace one form of scattering with another.
- It is mainly framed around `LINE` traversal. The eventual design also has to serve `OrbitThreader`, reference-path construction, diagnostics, geometry export, and tracking field evaluation.

Where the proposal is valid and useful:

- It correctly identifies that old OPAL has several layout schemes implemented in different places.
- It correctly criticizes public inheritance from `std::list<T>`.
- It correctly questions the `Beamline` / `TBeamline` / `TLine` / `FlaggedBeamline` hierarchy.
- It correctly identifies explicit enum checks as a barrier to adding new elements.
- It correctly points out that nested `LINE` handling should use a small layout state stack rather than stashing entire `OpalBeamline` objects.

- It correctly argues that layout should not require repeated full scans when a single geometric placement pass can be made to work.
- It correctly observes that `OpalBeamline` is poorly named for its runtime assembly role.

Net assessment: Jon's proposal is a useful critique and should influence the cleanup plan, but it should not replace the Chapter 2 placement model. The branch is best understood as a bridge stage toward that model, not as the final implementation. It introduces useful placement vocabulary and explicit pose records, but the architecture is still centered on visitor-based collection, `FieldList`, compatibility placement, and special-case bend logic. The next step is to turn the port/placement vocabulary into the actual layout API so that straight elements, analytic bends, curved multipoles, nested lines, misalignments, and field-local charts all participate through one coherent interface.

38.2 OPALX has 4 Layers of Structure (need better title)

Figure 1 extends the element-only (bend) class structure from the Elements chapter into the placement and tracking path used by the present `SBEND` and `RBEND` implementation in OPALX.

The diagram emphasizes four layers:

- parser-facing `Opal*` element definitions,
- runtime bend and field representations,
- beamline placement and coordinate-system assembly,
- reference-path and tracking-side infrastructure.

The parser-side bend objects define attributes, defaults, and input-language validation. The actual runtime geometry and field ownership lives in the representation classes `SBendRep` and `RBendRep`.

The next structural step is the placement layer. `OpalBeamline` assembles the declared components into a placed beamline and keeps a `PlacedElement` description for each runtime element, including the nominal and actual body, entry, exit, and support transforms.

For tracking, `OrbitThreader` threads a reference path through the placed beamline and builds the indexing and registration information needed by the tracker. `ParallelTracker` then consumes the placed beamline and the threaded reference-path information when updating external fields and advancing the reference particle.

38.2.1 `QUADRUPOLE` and `MULTIPOLE`

The same parser-to-runtime analysis can be carried out for the present quadrupole and generic multipole path. The main structural difference relative to bends is that `OpalQuadrupole` and `OpalMultipole` inherit **directly** from `OpalElement`. There is no intermediate parser-side family class analogous to `OpalBend`.

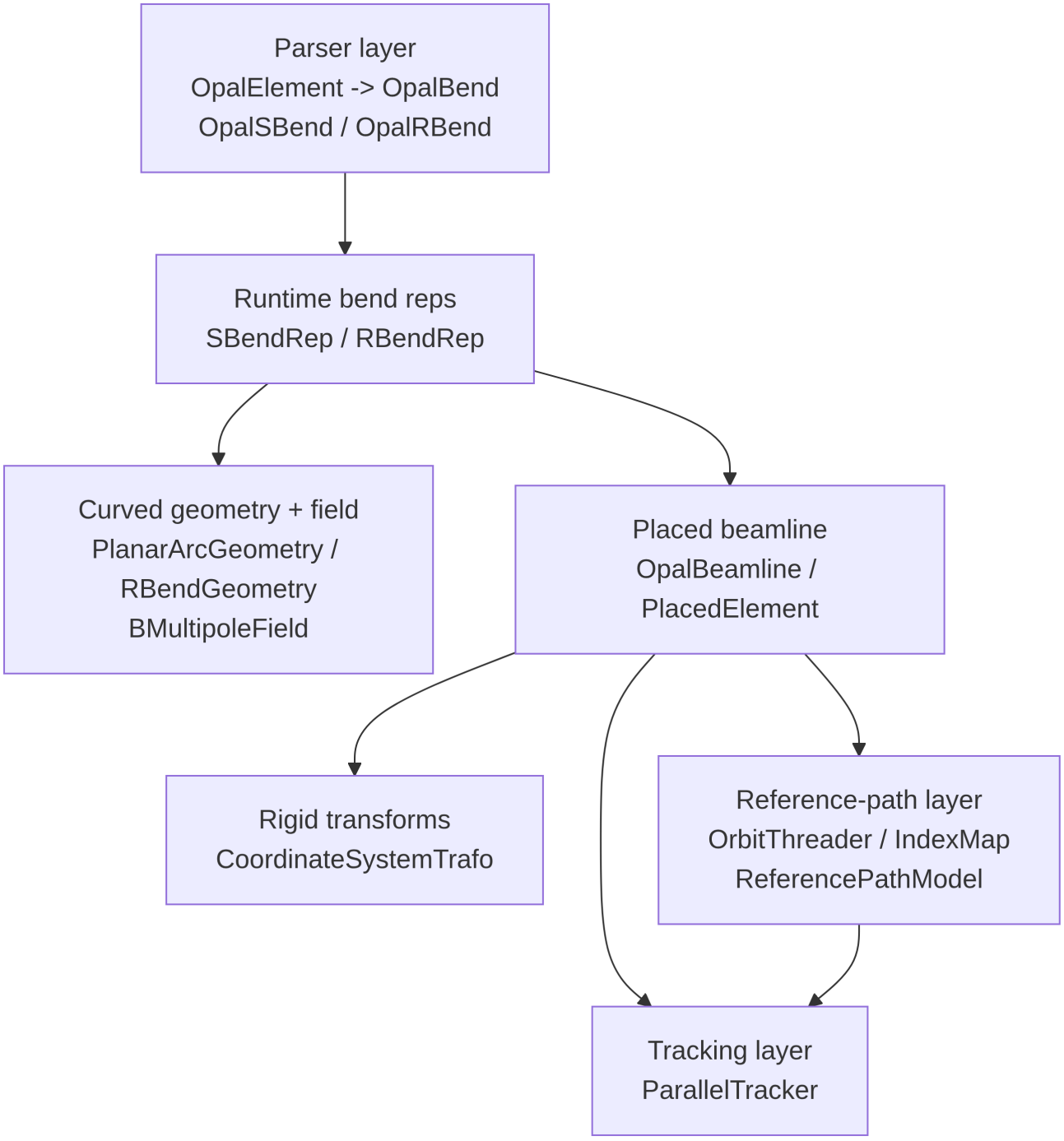


Figure 38.1: Draft class diagram for the current bend, placement, and tracking path in OPALX.

That detail matters because all common element-level attributes are inherited immediately from `OpalElement`, including:

- design type and aperture,
- placement and floor coordinates,
- orientation angles,
- local misalignments,
- output and common element options.

At update time, both parser-side front ends call the common `OpalElement` machinery and then configure the same runtime representation `MultipoleRep`. `QUADRUPOLE` is therefore a restricted front end on top of the generic `MULTIPOLE` runtime path.

The runtime path is straight-element based. `MultipoleRep` owns a `StraightGeometry` object and a `BMultipoleField`, and both `MULTIPOLE` and `QUADRUPOLE` enter the beamline placement pipeline as ordinary `Component` instances. Consequently, once the parser-side update has completed, the placement and tracking infrastructure is the same one used elsewhere in the beamline:

- `OpalBeamline` stores the placed component and its transforms,
- `PlacedElement` records the nominal and actual coordinate systems,
- `OrbitThreader` evaluates the host reference-path field model through the placed beamline,
- `ParallelTracker` uses the same placed component during bunch tracking.

In other words, the present quadrupole family differs from the bend family mainly on the parser side and in the runtime geometry type:

- bends use `OpalBend` and specialized curved runtime representations,
- `QUADRUPOLE` and `MULTIPOLE` use `OpalElement` directly and share one straight multipole runtime representation.

The separate `MULTIPOLET` implementation is not included in this draft diagram. It is a different combined-function magnet path and should be documented separately once the straight `MULTIPOLE` and `QUADRUPOLE` structure is fully settled.

38.3 `OpalElement` as Parser-to-Runtime Bridge

`OpalElement` is primarily a parser-side front-end class. It is **not** the object that is later threaded through `OpalBeamline` or stepped by `ParallelTracker`. Instead, it is the bridge between the OPAL input language and the runtime `ElementBase` hierarchy.

The essential structural point is:

- `OpalElement` inherits from `Element`,
- `Element` owns an embedded runtime `ElementBase`,

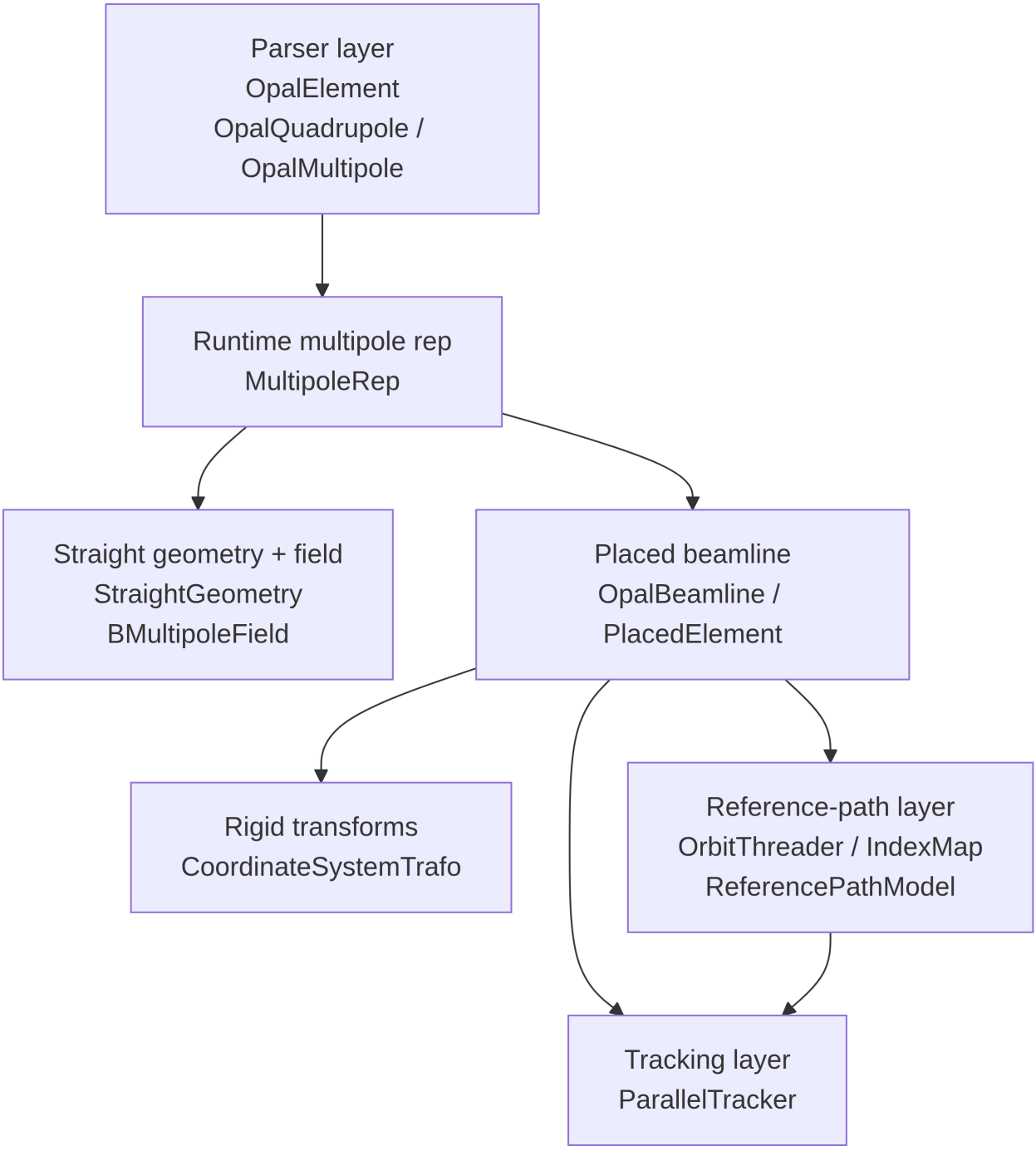


Figure 38.2: Draft class diagram for the current `QUADRUPOLE` and `MULTIPOLE` placement and tracking path in OPALX.

- derived parser-side classes such as `OpalSBend`, `OpalRBend`, `OpalQuadrupole`, and `OpalMultipole` configure that embedded runtime object.

This means that `OpalElement` has two responsibilities:

1. store and parse common OPAL element attributes,
2. transfer common placement, orientation, aperture, and misalignment data into the embedded runtime element before the derived class adds physics-specific configuration.

The common parser-side attributes handled in `OpalElement` include:

- `L`,
- aperture,
- `ORIGIN` and `ORIENTATION`,
- `X`, `Y`, `Z`, `THETA`, `PHI`, `PSI`,
- `DX`, `DY`, `DZ`, `DTHETA`, `DPHI`, `DPSI`,
- `ELEMEDGE`,
- the transverse-exit deletion flag.

The following diagram shows the role of `OpalElement` in the current control flow.

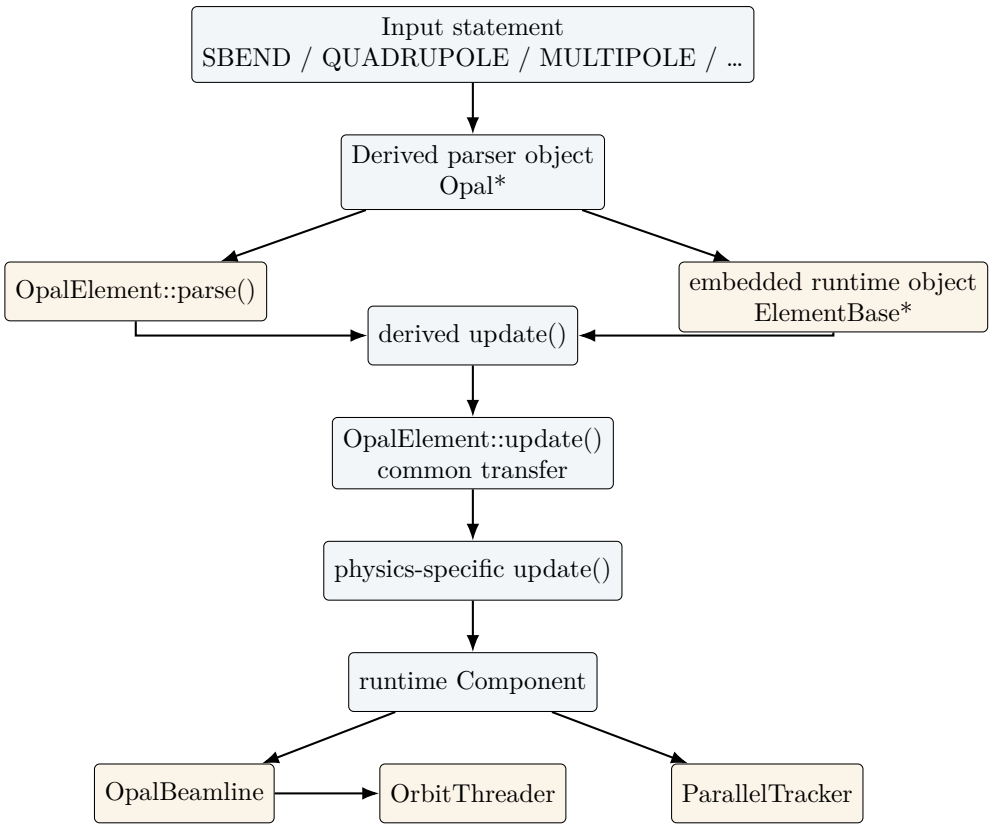


Figure 38.3: Control-flow view of `OpalElement` as the bridge from parsed OPAL element definitions to runtime `ElementBase` objects.

The corresponding class-level ownership relation is:

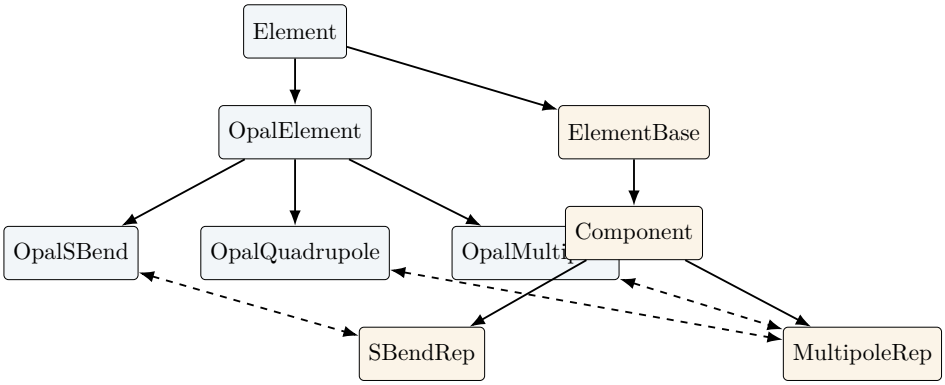


Figure 38.4: Class-level bridge from `OpalElement` to the runtime `ElementBase` hierarchy.

So the answer to the practical question "where is `OpalElement` used?" is:

- yes, first of all on the parser/input side,
- but more precisely as the **shared front-end bridge** from parsed element attributes into the runtime `ElementBase` object,
- after that handoff, placement, reference-path construction, and tracking are performed on the runtime object, not on `OpalElement` itself.

This distinction is especially important for the bend work, since parser-side inheritance and runtime-side inheritance are intentionally different:

- parser side: `OpalElement -> OpalBend -> OpalSBend / OpalRBend`,
- runtime side: `ElementBase -> Component -> BendBase -> SBend / RBend -> Rep`.

38.4 Element Placing (Posing)

This subsection is the working area for documenting how OPALX places, or *poses*, elements in space in a way that is consistent with Section 2: Coordinate Systems of the manual.

The immediate goal is not yet a final polished derivation. Instead, this sandbox section should fix the vocabulary, code anchors, and transformation chain used by the current implementation.

The placement topic will need to connect four layers:

- the parser-side placement attributes in `OpalElement`,
- the runtime pose stored in `ElementBase`,
- the placed beamline view represented by `PlacedElement`,
- the way `OpalBeamline`, `OrbitThreader`, and `ParallelTracker` consume those transforms.

38.4.1 Input placement, the current SE(3) bridge, and mismatch handling

The current OPALX implementation does not have a dedicated SE(3) class. The role of the rigid-motion carrier is instead played by `CoordinateSystemTrafo`. The important convention is that the stored transform maps **parent-frame coordinates into local-frame coordinates**. For a point $\mathbf{r}_{\text{parent}}$,

$$\mathbf{r}_{\text{local}} = Q(\mathbf{r}_{\text{parent}} - \mathbf{o}).$$

This is a sound rigid transform, but it is the parent-to-local form rather than the local-to-parent pose matrix often used in robotics texts. Composition is still an SE(3) composition, with the OPALX convention

$$(A \cdot B)(\mathbf{r}) = A(B(\mathbf{r})),$$

so the right-hand transform acts first.

The notation used in this subsection is:

- bold symbols such as $\mathbf{r}$, $\mathbf{o}$, and $\mathbf{R}(s)$ denote vectors or embedded points in $\mathbb{R}^3$,
- plain symbols such as s, t, and Δt denote scalars,
- frame and space symbols such as K, K_s, SO(3), and SE(3) are written non-bold,
- rigid transforms such as G_i^{nom}, Δ_i, and $B_{i,\alpha}$ are treated as maps in SE(3), acting on vectors by function application,
- $Q \in \text{SO}(3)$ denotes the rotational part of a rigid transform,
- C++ identifiers such as `CoordinateSystemTrafo` or `PlacedElement` are kept in monospace.

To remain consistent with Chapter 2, this sandbox note reserves:

- P_i for the **placement map** from the canonical local chart U_i into the laboratory frame K,
- $p_{i,\alpha}$ for the **abstract port** consisting of a marked boundary chart together with an adapted frame,
- $T_i(t)$ for the Chapter 2 laboratory-frame pose of the canonical element frame.

The present code discussion therefore uses distinct implementation-side symbols:

- G_i^{nom} for the stored parent-to-nominal-body rigid transform,
- G_i^{act} for the stored parent-to-actual-body rigid transform,
- $B_{i,\alpha}$ for the body-relative rigid transform of the port frame materialized by `ElementGeometry`.

The words `nominal` and `actual` need to be fixed before the bridge equations. In this note:

- `nominal` means the rigid placement of the **canonical element body frame** in the parent frame before any local survey or misalignment correction is applied,

- `actual` means the rigid placement of that same body frame after the local correction has been applied,
- these terms refer to **body placement in physical space**, not yet to the field-local charts used later by analytic bends.

For element i, the current bridge therefore distinguishes:

- the nominal body pose $G_i^{\mathrm{nom}} \in \mathrm{SE}(3)$, stored as `PlacementPose(parentToNominal)`,
- the local correction $\Delta_i \in \mathrm{SE}(3)$, stored as `Misalignment(nominalToActual)`,
- the actual body pose G_i^{act}, assembled from the first two quantities.

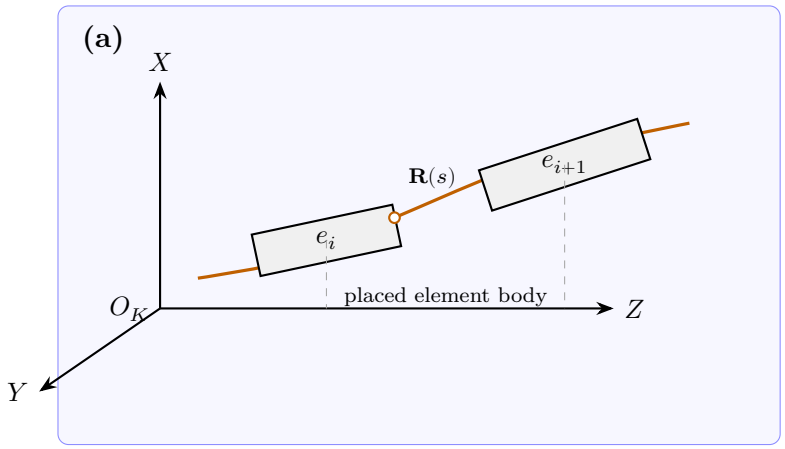


Figure 38.5: Floor Cartesian frame $K = (X, Y, Z)$ with the reference line shown in orange. Two placed element bodies, e_i and e_{i+1}, are shown together with a representative point $\mathbf{R}(s)$ on the reference line. The label `placed element body` is kept inside the figure as the only descriptive in-figure annotation.

The figure above should be read as the **floor-frame interpretation of nominal placement**. The global frame is the floor Cartesian frame $K = (X, Y, Z)$. The orange reference line is the reference path in that frame, and $\mathbf{R}(s)$ is a representative point on that path. The drawn element bodies e_i and e_{i+1} represent placed element bodies in floor space. At this stage of the discussion they should be read as **nominal** body poses. The corresponding actual body poses are obtained by applying the local correction Δ_i or Δ_{i+1} to those nominal poses; they are not drawn as separate boxes in this first figure.

The corresponding Chapter 2 to Sandbox crosswalk is:

Table 38.1: Table 10.3.1: Chapter 2 objects and their Sandbox implementation-side representatives.

Chapter 2 object	Sandbox implementation-side representative	Current code anchor	Comment
K	floor Cartesian parent frame	beamline / lab frame	same object as in Chapter 2
P_i	conceptual local-to-lab placement map	not stored directly	Chapter 2 geometric notion
G_i^{nom}	stored inverse rigid representative of the nominal body pose	`PlacementPose`, `ElementBase::getCSTransformGlobal2Local()`	implementation-side parent-to-local map
Δ_i	nominal-to-actual local correction	`Misalignment`, `ElementBase::getMisalignment()`	same correction idea as in Chapter 2
$G_i^{\mathrm{act}} = \Delta_i \cdot G_i^{\mathrm{nom}}$	stored inverse rigid representative of the actual body pose	assembled in `PlacedElement`	implementation-side parent-to-local map
$p_{i,\alpha}$	abstract port in the Chapter 2 sense	not stored as a single object	marked boundary chart plus adapted frame
$B_{i,\mathrm{entry}}$, $B_{i,\mathrm{body}}$, $B_{i,\mathrm{exit}}$	body-relative rigid transforms of the port frames	`ElementGeometry` via `getEntryPort()`, `getBodyPort()`, `getExitPort()`	implementation-level port-frame representatives
field-local chart	runtime field-evaluation chart, especially for analytic bends	`OpalBeamline::transformToFieldLocalCS()` and related methods	chart beyond Chapter 2 rigid placement

This table makes the key distinction explicit: the bridge first constructs **placement in floor space**, and only afterwards do some runtime paths map positions and vectors into a separate field-local chart. Those two layers should not be conflated.

Under that convention, the current posing bridge can be stated compactly as:

$$G_i^{\mathrm{act}} = \Delta_i \cdot G_i^{\mathrm{nom}},$$

The canonical port-frame representatives $B_{i,\mathrm{entry}}$, $B_{i,\mathrm{body}} = I$, and $B_{i,\mathrm{exit}}$ are body-relative marked frames. This means the same port can be queried either nominally or actually, depending on whether it is composed with G_i^{nom} or with G_i^{act}.

The corresponding nominal and actual port transforms are

$$G^{\mathrm{nom}}_{i,\mathrm{entry}} = B_{i,\mathrm{entry}} \cdot G^{\mathrm{nom}}_{i}, \qquad G^{\mathrm{nom}}_{i,\mathrm{exit}} = B_{i,\mathrm{exit}} \cdot G^{\mathrm{nom}}_{i},$$

and

$$G^{\mathrm{act}}_{i,\mathrm{entry}} = B_{i,\mathrm{entry}} \cdot G^{\mathrm{act}}_{i}, \qquad G^{\mathrm{act}}_{i,\mathrm{exit}} = B_{i,\mathrm{exit}} \cdot G^{\mathrm{act}}_{i}.$$

This is exactly the bridge assembled in `PlacedElement`. `ElementBase` stores the ingredients `PlacementPose` and `Misalignment`, while `PlacedElement` materializes both the nominal and actual body/port transforms.

Table 38.2: Table 10.3.2: Input-file quantities and their placement-bridge interpretation.

Input-file quantity	Geometric meaning	Bridge object	Main C++ entry points
`ORIGIN`, `ORIENTATION`	explicit nominal body pose	`PlacementPose`	`OpalElement::update()`, `ElementBase::setCSTrafoGlobal2Lo` `ElementBase::fixPosition()`
`X`, `Y`, `Z`, `THETA`, `PHI`, `PSI`	explicit nominal body pose in split scalar form	`PlacementPose`	`OpalElement::update()`, `ElementBase::setCSTrafoGlobal2Lo` `ElementBase::fixPosition()`
`DX`, `DY`, `DZ`, `DTHETA`, `DPHI`, `DPSI`	local nominal-to-actual correction	`Misalignment`	`OpalElement::update()`, `ElementBase::setMisalignment()`
`ELEMEDGE`	compatibility reference-path start coordinate	`elementPosition_m`	`ElementBase::setElementPosition(` `OpalBeamline::resolveVisitStartP` `OpalBeamline::compileCompatibili`
legacy edge geometry	body-relative entry/body/exit ports	`ElementGeometry`	`ElementBase::getEntryPort()`, `getBodyPort()`, `getExitPort()`, `getPlacementGeometry()`

The parser-side and runtime-side split is therefore:

1. `OpalElement::update()` interprets the input attributes and pushes them into the embedded runtime element.
2. `ElementBase` stores the nominal rigid transform, the local correction, and the optional `ELEMEDGE` reference coordinate.
3. `ElementBase::getPlacedElement()` assembles these quantities into the bridge view `PlacedElement`.
4. `OpalBeamline` consumes that placed-element view, either by respecting an already fixed pose or by compiling compatibility placement from reference order and `ELEMEDGE`.

Current mismatch detection is partial and distributed, not centralized:

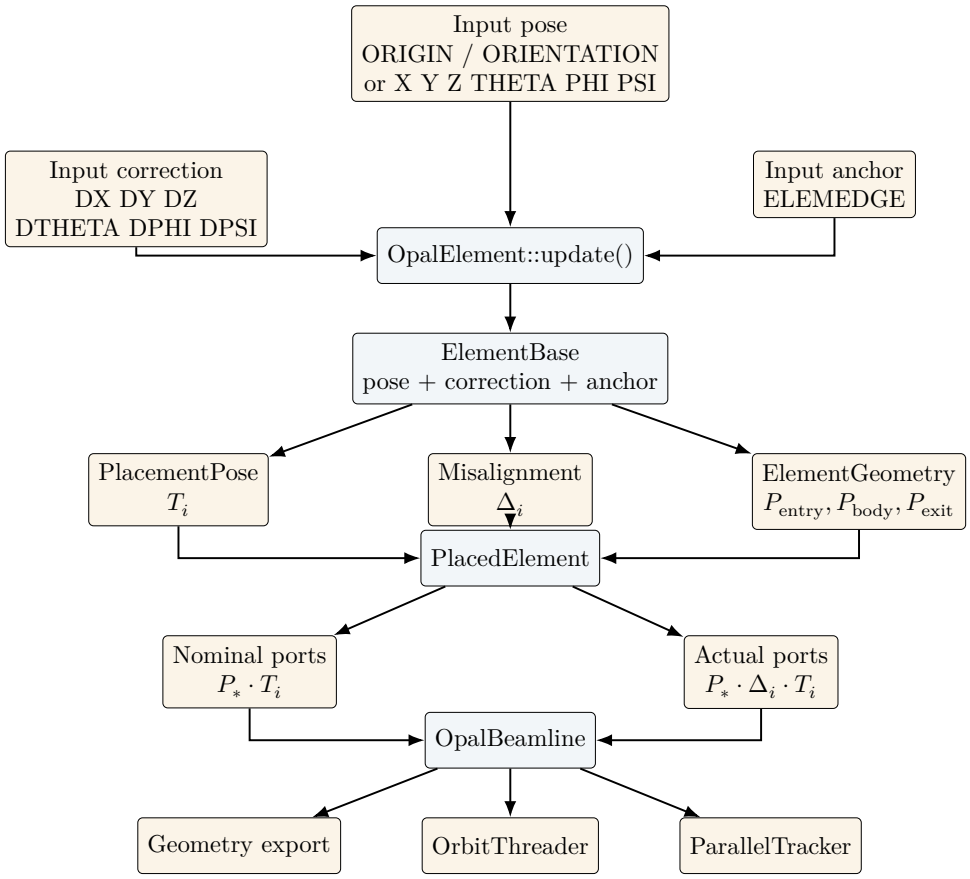


Figure 38.6: Bridge from input-file placement syntax to placed-element transforms and runtime use.

- malformed `ORIGIN` or `ORIENTATION` arrays are rejected in `OpalElement::update()` with an `OpalException`,
- analytic `SBEND` and `RBEND` elements without explicit `Z` placement still require `ELEMEDGE`; this is enforced in `OpalBeamline::resolveVisitStartField()`,
- explicit nominal placement is protected by `fixPosition()`, so the later compatibility placement pass does not overwrite an already posed element,
- port continuity and field-local consistency are currently validated mainly by regression tests such as `TestOpalBeamlinePlacement`, together with exported placement summaries, rather than by a dedicated runtime validator.

This also clarifies where a future mismatch detector should sit. The natural place is not the parser, but the bridge boundary where `PlacedElement` combines the nominal pose, local correction, and body-relative ports into a single explicit geometric record.

38.4.2 Current placement path in the code

At the moment, the placement handoff in OPALX is centered around `OpalElement::update()`. That function interprets the parser-side placement attributes and writes them into the embedded runtime `ElementBase` object.

In particular, the current path includes:

- global pose through `setCSTrafoGlobal2Local(...)`,
- the fixed-position flag through `fixPosition()`,
- local misalignment through `setMisalignment(...)`,
- optional longitudinal placement through `setElementPosition(...)`.

Once this has been done, `ElementBase::getPlacedElement()` provides the bridge to the explicit placed-element representation that is later used by the beamline assembly.

38.4.3 Scope of the placement note

The intended scope of this sandbox subsection is:

1. restate the placement conventions from Section 2 in code-facing language,
2. explain how `ORIGIN` / `ORIENTATION` relate to `X`, `Y`, `Z`, `THETA`, `PHI`, `PSI`,
3. explain how local misalignments `DX`, `DY`, `DZ`, `DTHETA`, `DPHI`, `DPSI` are applied,
4. explain the role of nominal versus actual placement,
5. explain how entry, body, and exit transforms are represented for curved elements,
6. prepare the later `OrbitThreader` documentation.

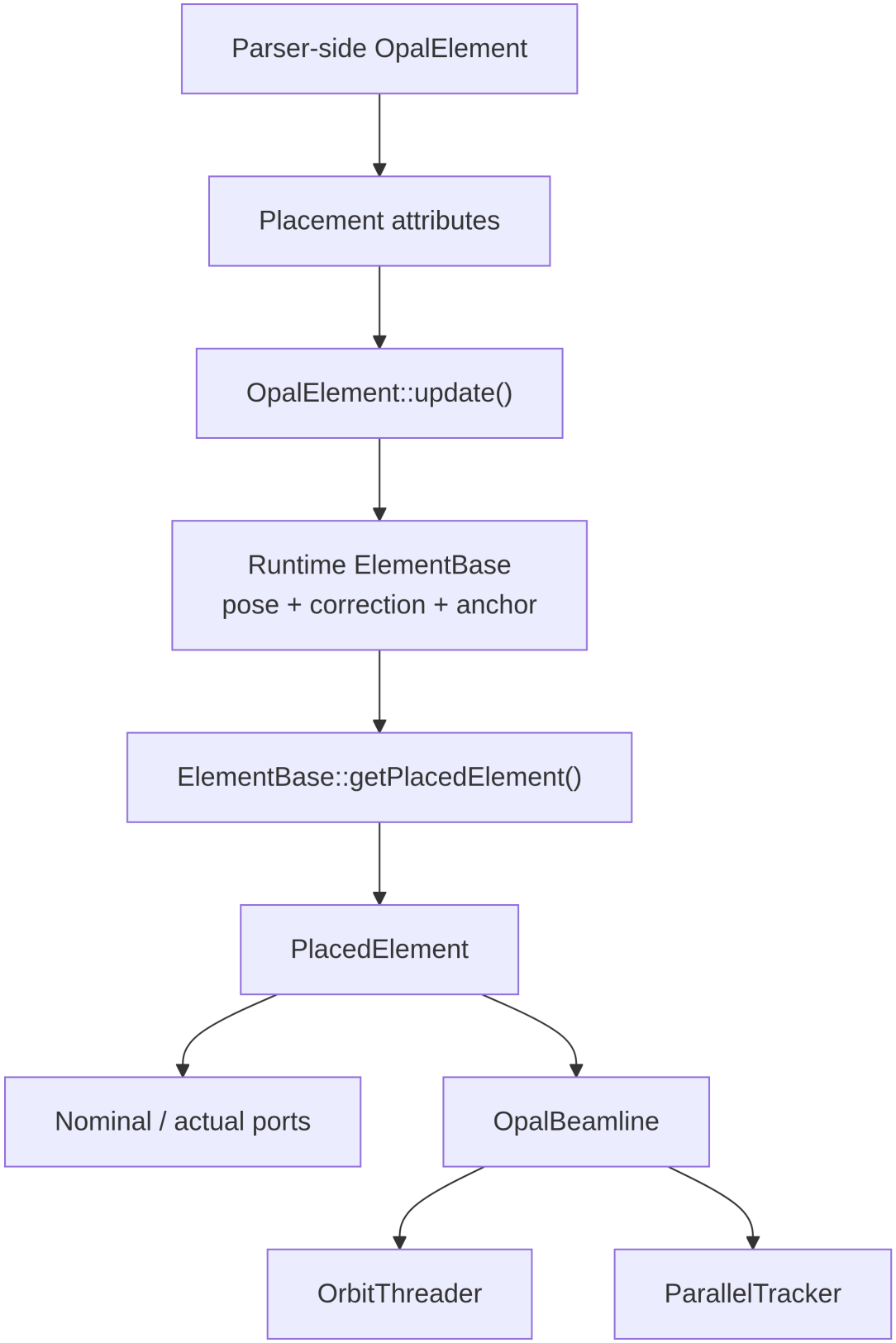


Figure 38.7: Draft flow for element posing in the current OPALX implementation.

38.4.4 Straight elements, curved elements, and runtime charts

For accelerator physicists the practical question is not just how a rigid body is posed in floor space, but how that rigid placement is connected to the reference trajectory and to the coordinates used later for field evaluation and tracking.

The comoving reference-orbit frame is the next layer:

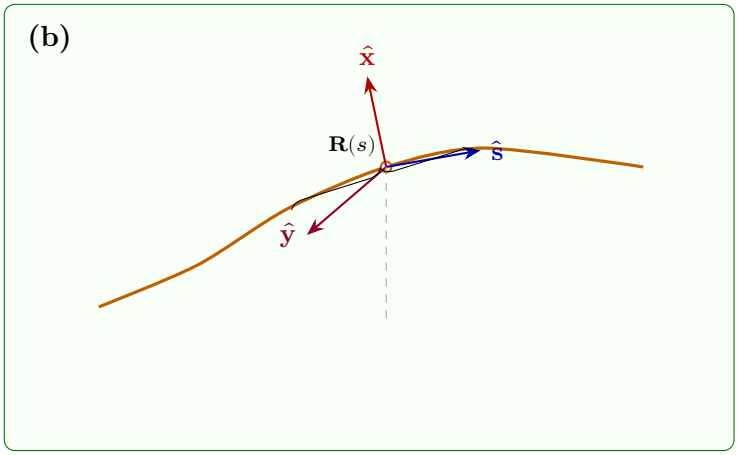


Figure 38.8: Comoving Frenet-Serret frame $K_s = (x, y, s)$ attached to the reference orbit. The tripod at $\mathbf{R}(s)$ defines the local axes $\hat{\mathbf{x}}$, $\hat{\mathbf{y}}$, and $\hat{\mathbf{s}}$, and the brace indicates the local longitudinal coordinate s along the reference curve.

The comoving Frenet-Serret frame $K_s = (x, y, s)$ is attached to the reference orbit rather than to the placed hardware. In the Chapter 2 language, this is the practical split between the **world chart** $(\mathbf{r}, t)$ used for placement in laboratory space and the **reference chart** s used to report where the reference orbit currently sits. For a straight element this distinction is almost invisible: the body z-direction and the reference longitudinal coordinate s coincide, so the body placement and the field chart are effectively the same to first order. For a curved element, however, that is no longer true. The rigid body is still placed in floor space by a single stored pose G_i^{nom}, but the field chart follows the design orbit through the magnet.

This is the key machinery of the current OPALX bridge:

In accelerator language, the chain is:

1. place the element body in the floor frame K,
2. define entry/body/exit ports relative to that body,
3. assemble nominal and actual placed-element transforms,
4. pass to a reference-orbit-based chart when field evaluation requires it.

This leads to the following conventions.

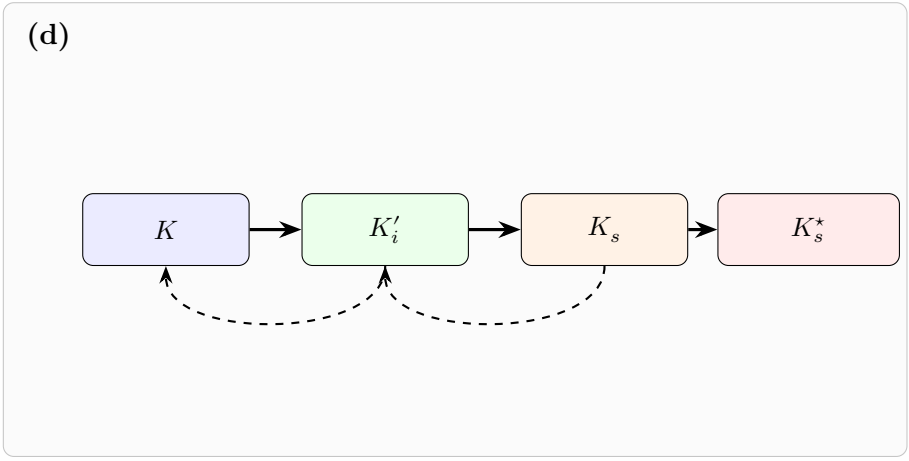


Figure 38.9: Transform chain between the floor frame K, the element-local frame K_i', the comoving Frenet-Serret frame K_s, and the boosted frame $K_s^\star$. The solid arrows represent the forward chain through placement pose (`CoordinateSystemTrafo`), reference-path transform, and Lorentz boost. The dashed arrows indicate inverse or field-query transforms used later in the runtime path.

Table 38.3: Table 10.3.3: Straight-element versus curved-element placement and chart conventions.

Topic	Straight elements	Curved elements (`SBEND`, `RBEND`, `RBEND3D`)	Main code anchors
Nominal body pose	one rigid stored pose G_i^{nom} in floor space	one rigid stored pose G_i^{nom} in floor space	`PlacementPose`, `setCSTrafoGlobal2Local()`, `getPlacedElement()`
Actual body pose	$G_i^{\mathrm{act}} = \Delta_i \cdot G_i^{\mathrm{nom}}$	$G_i^{\mathrm{act}} = \Delta_i \cdot G_i^{\mathrm{nom}}$	`Misalignment`, `PlacedElement::getActualBodyTran`
Entry/body/exit ports	rigid port-frame transforms $B_{i,\alpha}$ aligned with the straight body chart	rigid port-frame transforms $B_{i,\alpha}$ attached to the curved element body; they define geometric entry and exit faces	`getEntryPort()`, `getBodyPort()`, `getExitPort()`, `ElementGeometry`
Longitudinal chart used for geometry export	body z coordinate	nominal body ports plus bend design path / edge geometry	`getNominalEntryTransform()`, `getNominalExitTransform()`, `save3DLattice()`

Topic	Straight elements	Curved elements (SBEND, RBEND, RBEND3D)	Main code anchors
Longitudinal chart used for field evaluation	body-local chart is usually sufficient	separate field-local chart tied to the reference orbit	`transformToFieldLocalCS()`, `transformFromFieldLocalCS()`, `BendBase` field-local mapping
Compatibility placement without explicit pose	`ELEMEDGE` can supply the start coordinate, but explicit `Z` pose also works	analytic bends remain strict: without explicit `Z`, `ELEMEDGE` is required	`resolveVisitStartField()`, `compileCompatibilityPlacement()`

The parser-side orientation and misalignment angles should also be stated in the same code-facing way they are actually built:

- the orientation quaternion is assembled in `OpalElement::update()` as `rotTheta * (rotPhi * rotPsi)`,
- the corresponding axes are `theta -> y`, `phi -> x`, and `psi -> z`,
- the same axis order is used for the local misalignment rotation `rotationY * rotationX * rotationZ`,
- the transform stored in `CoordinateSystemTrafo` is the conjugated, parent-to-local form.

For the current implementation it is therefore safest to document the parser angles by their **construction order in code**, not by introducing additional intrinsic/extrinsic terminology that might obscure the existing convention.

From the accelerator-physics point of view, three additional remarks matter.

First, there is a distinction between **survey placement** and **reference orbit**. The survey or input placement fixes where the hardware sits in the floor frame. The reference orbit is the path that the traced reference particle actually follows through the placed lattice. In a perfectly matched straight section those two viewpoints nearly coincide. In curved or misaligned sections they do not: the survey pose still describes the body in floor space, while the reference orbit defines the comoving longitudinal coordinate used later for tracking and field queries.

Second, the **entry and exit faces** are geometric ports, not yet the same thing as a hard-edge transport model. The ports determine where the canonical body chart begins and ends, and they are used for geometric chaining, export, and diagnostics. A hard-edge or fringe-field model then decides how the field support extends relative to those geometric faces. This distinction is especially important for bends, where body extent and field-support extent need not coincide.

Third, `ELEMEDGE` should be understood physically as a **reference-path anchoring coordinate**. It is not an independent survey pose. In the compatibility placement path

it specifies where the element edge sits on the longitudinal reference coordinate used for beamline assembly. For ordinary straight elements this is mostly a convenience. For analytic bends without an explicit rigid `Z` placement it remains essential, because the code still needs an unambiguous longitudinal anchor from which the bend body and field support are laid out.

The main placement split for later runtime documentation is then:

- `OpalElement` parses and constructs the rigid nominal pose and local correction,
- `ElementBase` stores those ingredients and the optional compatibility longitudinal coordinate,
- `PlacedElement` assembles the explicit nominal/actual body and port transforms,
- `OpalBeamline` compiles these into the beamline placement assembly,
- `OrbitThreader` and `ParallelTracker` consume that assembly, with bends introducing a separate field-local chart on top of the rigid body placement.

This subsection is now the right place to continue with a dedicated `OrbitThreader` note, because the distinction between floor placement, body ports, and reference-orbit charts has been made explicit.

38.5 OrbitThreader

`OrbitThreader` is the first stage where the placed geometry is converted into a **traced reference orbit**. For accelerator physicists, its role is simple to state:

- start from the reference particle state,
- move that particle through the placed lattice in the summed external fields,
- record which elements are active along the traced longitudinal coordinate,
- hand that occupancy and interval information to the main tracker.

In other words, `OrbitThreader` is the bridge between static placement and the dynamic reference-path picture used by time-based tracking.

The symbols T_i, Δ_i, and $P_{i,*}$ retain the meanings fixed in Table Table 38.1; C++ class and function names stay in monospace.

38.5.1 Physical role of the traced reference path

The placed lattice tells us where hardware sits in floor space. The traced reference path tells us which parts of that hardware are actually intersected by the reference particle and in what longitudinal order. Those are not the same question in a general 3D placed lattice.

This is why `OrbitThreader` exists at all. It does not merely sort elements by a nominal `Z` coordinate. It integrates the reference particle through the placed fields and constructs a path-based activation model from that integration.

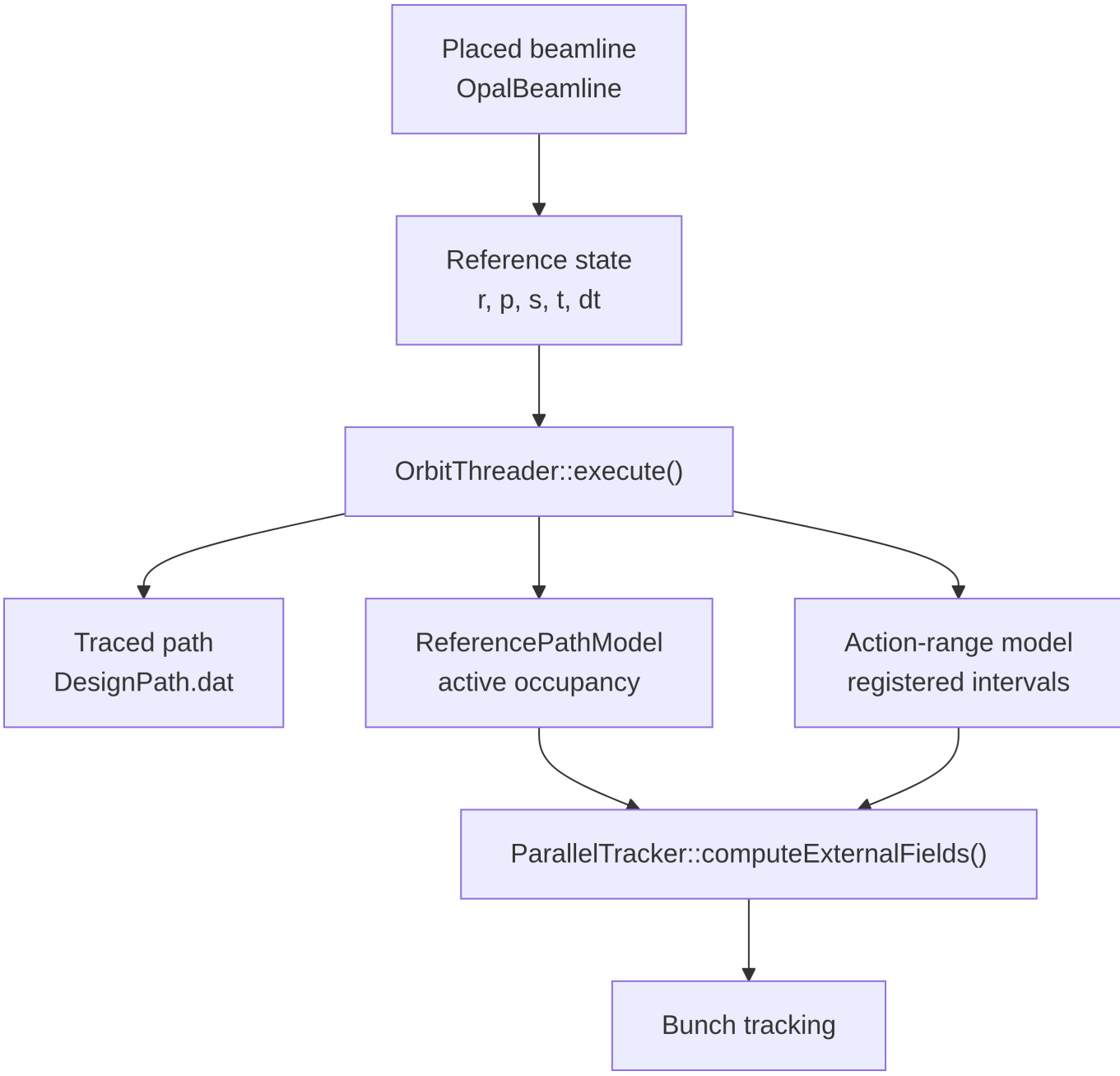


Figure 38.10: Draft OrbitThreader role in the OPALX tracking machinery.

38.5.2 Code-side path construction

The current implementation is centered around:

- constructor: `OrbitThreader(ref, r, p, s, maxDiffZBunch, t, dT, stepSizes, beamline)`
- main driver: `OrbitThreader::execute()`
- local integrator: `OrbitThreader::integrate(...)`

The input state comes directly from the bunch reference particle in `ParallelTracker`:

- reference particle rest data,
- lab-frame reference position and momentum,
- current path-length coordinate `s`,
- bunch time `t`,
- current time step `dt`.

So `OrbitThreader` does not invent a separate reference particle. It reuses the same physical reference state that the main time-based tracker is about to use.

38.5.3 What OrbitThreader produces

The class owns two distinct path models.

1. `ReferencePathModel` from `imap_m`

- this is the traced active-element occupancy model,
- it answers: which elements are active around the current traced longitudinal position?

2. `actionRangeRegistrationModel_m`

- this is the registration model for backward-compatible action ranges,
- it answers: which element intervals should be registered on the elements themselves for later legacy-style use?

This split matters physically. The first model is a **traced occupancy model**. The second is a **registration model** that translates the traced result into legacy element intervals and `ELEMEDGE`-compatible ranges.

38.5.4 Why bends need special treatment

The field evaluation stage later distinguishes straight elements from analytic bends. `ParallelTracker::computeExternalFields()` uses the `OrbitThreader` occupancy query, but then treats bends specially:

- straight and most other elements use one rigid local field transform,

- bends build tracking slices and use an entry-to-slice-local reference-path transform inside each slice.

So the information flow is:

1. `OrbitThreader` says which bend is active along the traced path,
2. `ParallelTracker` asks the bend for tracking slices,
3. the bunch is mapped slice-by-slice into the bend's local reference-path chart,
4. the bend applies the slice field,
5. the bunch is mapped back to the reference frame.

This is the point where the distinction made in `10.3.4` becomes operational: - rigid floor placement tells us where the bend is, - the field-local slice chart tells us how the bunch is advanced through it. - reference-path slices and bend-local tracking charts in more detail.

38.5.5 Disconnected placement and mismatch detection

`OrbitThreader` also contains the first real runtime consistency check for placed lattices. After the traced path has been built, `validateVisitedElements(...)` scans placed elements that should have been reachable and checks whether the reference path ever visited them.

If a nontrivial placed element lies within the traced longitudinal window but is never intersected, the code raises a user-facing exception and reports its nominal entry, body, and exit coordinates.

This is the current runtime answer to the practical question:

- "Did I place the hardware in a way that the reference trajectory can actually thread through it?"

For an accelerator physicist, this is often the most useful placement diagnostic, because it converts a disconnected survey configuration into a clear statement about the traced reference orbit.

38.5.6 Relation to `ELEMEDGE`

`OrbitThreader::registerElement(...)` and `processElementRegister()` are where the traced path is turned back into legacy-compatible element ranges.

For general elements, the code infers an effective element-edge coordinate from the traced local entrance state. For bends, `processElementRegister()` applies the bend field extent explicitly:

- it gets `fieldBegin` and `fieldEnd`,
- if the bend already has an explicit `ELEMEDGE`, that value is kept as the anchor,
- the registered action range is then built from that anchored field interval.

Physically, this means: - the traced path determines what the reference particle actually visited, - ELEMEDGE still acts as the longitudinal anchor used to express the bend field interval in the legacy coordinate language.

ELEMEDGE should be seen as depricayed, in a linear setup instead Z should be used.

38.5.7 Handoff to the main tracker

The handoff is in ParallelTracker:

1. construct OrbitThreader,
2. call execute(),
3. retrieve the global bounding box,
4. use oth.query(...) for active elements around each bunch container,
5. also inspect oth.getActionRangeRegistrationModel() to ensure bend segments remain visible in the queried window.

So OrbitThreader is not only a diagnostic tool or a design-path logger. It is part of the live tracking machinery that decides which elements are considered active for field application.

38.5.8 Worked example: one placed bend

It is useful to make the machinery concrete on the simplest nontrivial case: a single placed analytic bend, for example one SBEND instance B1.

The physical story is:

1. the input deck fixes the bend body pose in floor space, either explicitly by ORIGIN / ORIENTATION or X,Y,Z,THETA,PHI,PSI, or compatibly through a longitudinal anchor such as ELEMEDGE,
2. OpalElement::update() turns that into a stored nominal body pose G_{B1}^{nom} and an optional local correction,
3. OpalBeamline assembles the corresponding PlacedElement,
4. OrbitThreader traces the reference particle and determines the interval along the traced path where B1 is actually active,
5. ParallelTracker queries that interval and, because B1 is a bend, applies the bend field slice by slice in a reference-path-local chart.

For this one-bend case, the key objects and physical meanings are:

Table 38.4: Table 10.4.1: Worked-example dictionary for one placed analytic bend.

Stage	Object or quantity	Physical meaning	Main code anchors
1	G_{B1}^{nom}	nominal bend body pose in floor space	PlacementPose, setCSTrafoGlobal2Local()

Stage	Object or quantity	Physical meaning	Main code anchors
2	Δ_{B1}	local survey / misalignment correction	`Misalignment`, `setMisalignment()`
3	$B_{B1,\mathrm{entry}}$, $B_{B1,\mathrm{exit}}$	body-relative port-frame transforms for the geometric bend entry and exit faces	`getEntryPort()`, `getExitPort()`, `ElementGeometry`
4	traced active interval	part of the reference path where the bend is actually intersected	`OrbitThreader::execute()`, `imap_m.add(...)`
5	registered action range	legacy-compatible bend field interval	`registerElement()`, `processElementRegister()`
6	bend slices	local reference-path pieces used for field application	`buildTrackingSlices()`, `applySlice()`

The important physics point is that the rigid bend pose alone does not yet tell the tracker how to advance the bunch through the bend. It tells the code where the hardware sits. `OrbitThreader` then determines where the **reference orbit** actually threads that hardware, and the bend tracker finally resolves that path into slice-local field applications.

This is also where `ELEMEDGE` enters most clearly. For the worked bend:

- if an explicit bend edge coordinate is already set, that coordinate remains the bend anchor,
- the traced reference path then determines the visited field interval around that anchor,
- the action-range registration model carries that interval forward so that the main tracker continues to see the bend in the correct longitudinal window.

So the one-bend example already contains the full machinery in miniature:

- survey placement,
- nominal/actual body pose,
- geometric ports,
- traced occupancy interval,
- legacy-compatible registration interval,
- slice-based bend tracking.

38.5.9 ToDo

1. OrbitThreader: a short note on `trackBack()` and why the path is traced slightly backward before the forward pass,

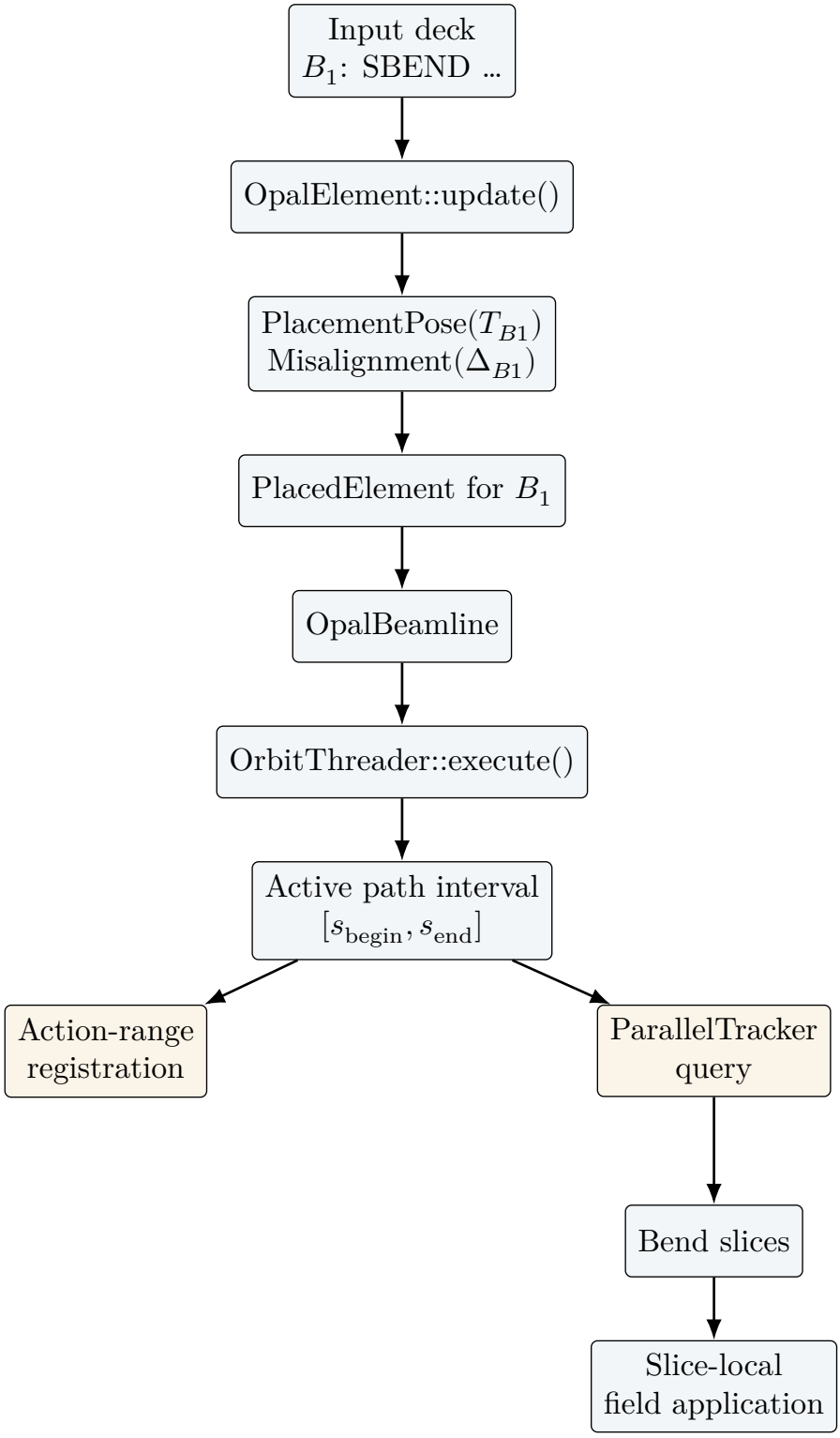


Figure 38.11: Worked-example flow for one placed analytic bend.

2. OrbitThreader: a diagram of the two internal models: `ReferencePathModel` versus action-range registration,

38.6 Autophasing

Autophasing fixes the RF phase, or lag, used by an accelerating element for a given reference particle. In the old OPAL formulation the RF elements are external field maps. The spatial field is sampled on a one-, two-, or three-dimensional grid, interpolated at the particle position, and multiplied by the time-dependent RF factor

$$\cos(\omega t + \varphi),$$

where ω is the angular RF frequency and φ is the lag. The autophasing problem is therefore: choose φ so that the reference particle obtains the desired, usually maximal, energy gain through the element.

This sandbox note uses the old OPAL derivation as the starting point, but keeps the wording OPALX-facing. The code-facing question for OPALX is how this phase condition is tied to the traced reference path, the element-local field map, and the time model used during reference-particle threading.

38.6.1 Standing-Wave Cavity

For a standing-wave cavity with longitudinal field $E_z(z, r)$, the energy gain of a particle with charge q is written

$$\Delta E(\varphi, r) = qV_0 \int_{z_{\text{begin}}}^{z_{\text{end}}} \cos(\omega t(z, \varphi) + \varphi)\, E_z(z, r)\, dz.$$

The auto-phase for maximum energy gain satisfies

$$\frac{d\Delta E}{d\varphi} = 0.$$

Keeping the possible phase-dependence of the arrival-time model gives

$$\begin{aligned} 0 &= \int_{z_{\text{begin}}}^{z_{\text{end}}} \left(1 + \omega \frac{\partial t}{\partial \varphi}\right) \sin(\omega t(z, \varphi) + \varphi)\, E_z(z, r)\, dz \\ &= \cos\varphi\, \Gamma_1 + \sin\varphi\, \Gamma_2. \end{aligned}$$

Thus the lag is determined by

$$\tan\varphi = -\frac{\Gamma_1}{\Gamma_2}.$$

For a sampled field map, OPAL assumes a linear field interpolation and a linear time interpolation between neighboring samples z_{i-1} and z_i. With $\Delta z_i = z_i - z_{i-1}$, $\Delta t_i = t_i - t_{i-1}$, and $\Delta E_{z,i} = E_{z,i} - E_{z,i-1}$,

$$\Gamma_1 = \sum_{i=1}^{N-1} \left(1 + \omega\frac{\partial t}{\partial\varphi}\right) \int_{z_{i-1}}^{z_i} \sin\left(\omega\left(t_{i-1} + \Delta t_i \frac{z - z_{i-1}}{\Delta z_i}\right)\right) \left(E_{z,i-1} + \Delta E_{z,i}\frac{z - z_{i-1}}{\Delta z_i}\right) dz$$

and

$$\Gamma_2 = \sum_{i=1}^{N-1} \left(1 + \omega\frac{\partial t}{\partial\varphi}\right) \int_{z_{i-1}}^{z_i} \cos\left(\omega\left(t_{i-1} + \Delta t_i \frac{z - z_{i-1}}{\Delta z_i}\right)\right) \left(E_{z,i-1} + \Delta E_{z,i}\frac{z - z_{i-1}}{\Delta z_i}\right) dz.$$

The products in these interval integrals can be evaluated analytically. In normalized interval coordinates $\tau \in [0, 1]$, define

$$\begin{aligned}
\Gamma_{11,i} &= \int_0^1 \sin(\omega(t_{i-1} + \tau\Delta t_i))\; d\tau = -\frac{\cos(\omega t_i) - \cos(\omega t_{i-1})}{\omega\Delta t_i},\\
\Gamma_{12,i} &= \int_0^1 \tau\sin(\omega(t_{i-1} + \tau\Delta t_i))\; d\tau = \frac{-\omega\Delta t_i\cos(\omega t_i) + \sin(\omega t_i) - \sin(\omega t_{i-1})}{\omega^2\Delta t_i^2},\\
\Gamma_{21,i} &= \int_0^1 \cos(\omega(t_{i-1} + \tau\Delta t_i))\; d\tau = \frac{\sin(\omega t_i) - \sin(\omega t_{i-1})}{\omega\Delta t_i},\\
\Gamma_{22,i} &= \int_0^1 \tau\cos(\omega(t_{i-1} + \tau\Delta t_i))\; d\tau = \frac{\omega\Delta t_i\sin(\omega t_i) + \cos(\omega t_i) - \cos(\omega t_{i-1})}{\omega^2\Delta t_i^2}.
\end{aligned}$$

Then the sampled sums become

$$\Gamma_1 = \sum_{i=1}^{N-1} \left(1 + \omega\frac{\partial t}{\partial\varphi}\right) \Delta z_i \left[E_{z,i-1}(\Gamma_{11,i} - \Gamma_{12,i}) + E_{z,i}\Gamma_{12,i}\right],$$

and

$$\Gamma_2 = \sum_{i=1}^{N-1} \left(1 + \omega\frac{\partial t}{\partial\varphi}\right) \Delta z_i \left[E_{z,i-1}(\Gamma_{21,i} - \Gamma_{22,i}) + E_{z,i}\Gamma_{22,i}\right].$$

The remaining ingredient is the time model $t(z, \varphi)$. The old OPAL algorithm initializes it from an approximate kinetic-energy ramp,

```
K[i] = K[i-1] + (z[i] - z[0]) * q * V;
b[i] = sqrt(1. - 1. / ((K[i] - K[i-1]) / (2.*m*c^2) + 1)^2);
t[i] = t[0] + (z[i] - z[0]) / (c * b[i]);
```

which assumes that the kinetic energy increases linearly with position and is proportional to the maximum voltage. With that provisional time model one computes φ, updates the kinetic-energy model using the corresponding field integral, recomputes the times, and repeats until the phase converges.

For OPALX this iteration should be interpreted as a reference-particle preprocessing step. It provides the RF lag consistent with the chosen reference trajectory and field-map convention before the main bunch tracking step uses the element.

38.6.2 Traveling-Wave Structure

For a traveling-wave structure, the same idea is applied to a field assembled from several phase-shifted pieces. The old manual describes the field as an entry fringe field, two core standing-wave contributions, and an exit fringe field. A representative energy-gain model is

$$\begin{aligned}\Delta E(\varphi, r) = qV_0 &\int_{z_{\mathrm{begin}}}^{z_{\mathrm{beginCore}}} \cos(\omega t + \varphi) E_z(z, r)\, dz \\ &+ qV_{\mathrm{core}} \int_{z_{\mathrm{beginCore}}}^{z_{\mathrm{endCore}}} \cos(\omega t + \varphi_{\mathrm{c1}} + \varphi) E_z(z, r)\, dz \\ &+ qV_{\mathrm{core}} \int_{z_{\mathrm{beginCore}}}^{z_{\mathrm{endCore}}} \cos(\omega t + \varphi_{\mathrm{c2}} + \varphi) E_z(z + s, r)\, dz \\ &+ qV_0 \int_{z_{\mathrm{endCore}}}^{z_{\mathrm{end}}} \cos(\omega t + \varphi_{\mathrm{ef}} + \varphi) E_z(z, r)\, dz,\end{aligned}$$

where s is the cell length. Compared with the standing-wave case, the stationarity condition contains several Γ_1- and Γ_2-type sums with different phase offsets and integration ranges.

For the old `FINLB02_RAC` example,

```
FINLB02_RAC: TravelingWave, L=2.80, VOLT=14.750*30/31,
             NUMCELLS=40, FMAPFN="FINLB02-RAC.T7",
             ELEMEDGE=2.67066, MODE=1/3,
             FREQ=1498.956, LAG=FINLB02_RAC_lag;
```

the decomposition gives

$$
\begin{aligned}
V_{\mathrm{core}} &= \frac{V_0}{\sin(2\pi/3)} = \frac{2V_0}{\sqrt{3}},\\
\varphi_{\mathrm{c1}} &= \frac{\pi}{6},\\
\varphi_{\mathrm{c2}} &= \frac{\pi}{2},\\
\varphi_{\mathrm{ef}} &= -2\pi(\mathrm{NUMCELLS}-1)\,\mathrm{MODE}.
\end{aligned}
$$

In the ultra-relativistic limit, where β changes only weakly along the structure, the arrival time can be approximated by

$$t(z,\varphi) \simeq \frac{z}{\beta c} + t_0.$$

The phase condition can then be simplified by exploiting the periodicity of the core field and the sinusoidal RF factor. In the old example the core field has period $3s$ and $\omega/(\beta c) \approx 10\pi$, which reduces the traveling-wave autophasing problem to a single periodic convolution-like condition over one three-cell interval. This approximation is useful as a diagnostic and a fast initial guess, but the general OPALX implementation should keep the full piecewise field-map and reference-time model available.

38.6.3 Consistency Notes for OPALX

- The phase φ is an element-local RF lag, not a global beamline coordinate.
- The time samples t_i must be consistent with the reference path used by the element placement and tracking machinery.
- The sign convention in $\cos(\omega t + \varphi)$ must be kept aligned with the input-language definition of `LAG` and with any field-map convention.
- Standing-wave and traveling-wave elements share the same mathematical structure: maximize the reference-particle energy gain with respect to the RF lag, then use the resulting lag during tracking.

38.7 Koordinate Systems

38.7.1 OPALX Coordinate Frames and Transform APIs

This note summarizes the coordinate frames currently used in OPALX and maps the PhysicsManual terminology to the concrete code structures and helper functions.

The PhysicsManual coordinate-system chapter uses the following conceptual language:

- `K = (X,Y,Z)`: laboratory or floor Cartesian frame.
- `B`: reference base used for ordering and reporting, such as a transfer-line interval, ring circle, or graph.

- `s`: reference/reporting coordinate on `B`.
- `U_i`: canonical local chart of element `i`.
- `Omega_i`: local support of the element field, aperture, or material model.
- `p_i`: named port or marked boundary chart, such as entrance or exit.
- `P_i`: rigid placement map embedding the local chart into the lab frame.
- `T_i`: rigid placement transform, represented by a translation and rotation.
- `Gamma_i(ell)`: optional intrinsic reference path of element `i`.
- `C_pq` or `g_ij`: port-to-port or chart-to-chart transition map.

In the current code, `lab`, `global`, and `floor` are often used interchangeably for the same Cartesian world frame. For element placement, treat these as aliases of the lab/floor frame `K`.

38.7.1.1 Core Transform Convention

The common implementation type is `CoordinateSystemTrafo`.

API	Meaning
`transformTo(r)`	Transform a point from the parent/source frame into this transform's target frame.
`transformFrom(r)`	Transform a point from the target frame back to the parent/source frame.
`rotateTo(v)`	Rotate a vector into the target frame, without translation.
`rotateFrom(v)`	Rotate a vector back to the parent/source frame, without translation.
`transformBunchTo(R,n)`	Apply `transformTo` to a particle-position view.
`transformBunchFrom(R,n)`	Apply `transformFrom` to a particle-position view.
`rotateBunchTo(P,n)`	Apply `rotateTo` to a particle/vector view.
`rotateBunchFrom(P,n)`	Apply `rotateFrom` to a particle/vector view.
`inverted()`	Return the inverse transform.

Relevant implementation:

- `src/Algorithms/CoordinateSystemTrafo.h`
- `src/Algorithms/CoordinateSystemTrafo.cpp`

38.7.1.2 Frame Table

Frame	PhysicsManual name	Code storage	Into this frame	Out of this frame
Lab / floor / global Cartesian	`K = (X,Y,Z)`	implicit root frame	identity	identity
Beamline / line frame	line `ORIGIN`, `ORIENTATION`; compatibility reference for `ELEMEDGE`	`OpalBeamline::com`	`OpalBeamline::transformTo`, `rotateTo`, `getCSTrafoLab2Local()`	`OpalBeamline::transformFrom`, `rotateFrom`, `getCSTrafoLab2Local().invert`
Element nominal local/body frame	local chart `U_i`, placed by `P_i`	`ElementBase::csTrafoLab2Local`	`OpalBeamline::transformTo(comp,r)` or `comp->getCSTrafoGlobal2Local()`	`OpalBeamline::transformFromLocalCS(comp,r)`, `rotateFromLocalCS`, `comp->getCSTrafoGlobal2Local().transformTo(r)` or `comp->getCSTrafoGlobal2Local().tr`
Explicitly posed element frame	`X,Y,Z,THETA,PHI,PSI` placement	set in `OpalElement::update()`	same as element local frame after `setCSTrafoGlobal2Local()`	same as element local frame
Misaligned element frame	static error transform `Delta T_i^err`	`ElementBase::misalignment`	`getMisalignment(comp)` * `getCSTrafoLab2Local()` in tracking also * `pc->getToLabTrafo()`	`getMisalignment(comp)` the full chain, or `comp->getRefCSTrafo` = `refToLocalCSTrafo.inverted()`
Element begin / entrance port	`p_i^in`, entrance boundary chart	`ElementBase::getEdgeToBegin()`	`getEdgeToBegin()` * `getCSTrafoGlobal2Local()`	use `getEdgeToBegin().transformFrom()` on the local-based transform
Element end / exit port	`p_i^out`, exit boundary chart	`ElementBase::getEdgeToEnd()`	`getEdgeToEnd()` * `getCSTrafoGlobal2Local()`	use `.transformFrom()` on the local-based transform
Bend intrinsic reference path	`Gamma_i(ell)`	`PlanarArcGeometry`, `RBendGeometry`, `Euclid3D`	`getTransform(s)`, `getEntranceFrame()`, `getExitFrame()`	`Euclid3D` inverse / reverse transform logic; not the same API as `CoordinateSystemTrafo`

Frame	PhysicsManual name	Code storage	Into this frame	Out of this frame
Runtime moving reference/bunch frame	tracker reference frame around reference particle	`ParticleContainer::refToLabTrafo_m`, `refPartR_m`, `refPartP_m`, `sPos_m`	lab/reference is used through `pc->getToLabTrafo()` in transform chains	update via `ParticleContainer::updateRef`
Element-local field-evaluation frame	element local frame plus errors	temporary chain in `ParallelTracker::computeExternalFields()`	`refToLocalCSTrafo = misalignment * (lab2local * pc->getToLabTrafo());` then `pc->transformBunch(refToLocalCSTrafo)`	`localToRefCSTrafo = refToLocalCSTrafo.inverted()`
Beam / self-field frame	beam frame, origin at reference, `z` along mean momentum	local variables in `ParallelTracker::computeSpaceChargeFields()`	`referenceToBeamFrameCSTrafo`	`beamToReferenceCSTrafo` and `applyInBunch(E/B)`
Mesh frame for self fields	follows current particle `R` representation	field solver mesh state	if `R` is beam-frame, mesh is beam-frame	after transform back, `bunchUpdate()` restores reference-frame mesh
Adaptive-bin rest frame	per-bin quasi-rest frame	`BinnedFieldSolver::BinKinematics`	`CoordinateSystemTrafo` uses `gammaBin`, `pmean`, `prepareRhoForBin()`	`accumulateFieldToTemp()` transforms `E'` to lab `E,B`

38.7.1.3 Important Code Paths

Explicit Element Placement

For elements with absolute placement attributes, `OpalElement::update()` builds the placement transform from `X`, `Y`, `Z`, `THETA`, `PHI`, and `PSI`:

```
CoordinateSystemTrafo global2local(origin, rotation.conjugate());
base->setCSTrafoGlobal2Local(global2local);
base->fixPosition();
```

This makes the element-local frame accessible through `ElementBase::getCSTrafoGlobal2Local()`.

Relevant files:

- `src/Elements/OpalElement.cpp`
- `src/AbsBeamline/ElementBase.h`

Beamline to Element Local

`OpalBeamline` wraps the element transform API:

```
transformToLocalCS(comp, r)
transformFromLocalCS(comp, r)
rotateToLocalCS(comp, v)
rotateFromLocalCS(comp, v)
getCSTrafoLab2Local(comp)
```

Internally these use `comp->getCSTrafoGlobal2Local()`.

Relevant file:

- `src/Elements/OpalBeamline.h`

Begin and End Port Frames

The default element begin and end frames are:

```
getEdgeToBegin() = identity
getEdgeToEnd()   = translation by (0, 0, L)
```

The lab-frame begin and end poses are usually formed as:

```
CoordinateSystemTrafo toBegin = element->getEdgeToBegin()
                              * element->getCSTrafoGlobal2Local();

CoordinateSystemTrafo toEnd = element->getEdgeToEnd()
                            * element->getCSTrafoGlobal2Local();
```

Relevant files:

- `src/AbsBeamline/ElementBase.h`
- `src/Elements/OpalBeamline.cpp`

Runtime Field Evaluation Chain

During particle tracking, particle coordinates are stored in the moving reference frame. To apply an element, OPALX composes:

```
CoordinateSystemTrafo refToLocalCSTrafo =
    itsOpalBeamline_m.getMisalignment(element)
    * (itsOpalBeamline_m.getCSTrafoLab2Local(element) * pc->getToLabTrafo());

CoordinateSystemTrafo localToRefCSTrafo = refToLocalCSTrafo.inverted();

pc->transformBunch(refToLocalCSTrafo);
element->apply(pc);
pc->transformBunch(localToRefCSTrafo);
```

This is the active runtime transition:

```
reference frame -> lab frame -> nominal element local -> misaligned element local
```

and then back again.

Relevant file:

- `src/Algorithms/ParallelTracker.cpp`

Moving Reference Frame

Each particle container stores the current moving reference state:

```
refPartR_m
refPartP_m
sPos_m
toLabTrafo_m
```

Useful accessors:

```
pc->getRefPartR()
pc->getRefPartP()
pc->get_sPos()
pc->getToLabTrafo()
pc->updateRefToLabCSTrafo(dt)
```

Relevant file:

- `src/PartBunch/ParticleContainer.hpp`

Beam Frame for Space Charge

`ParallelTracker::computeSpaceChargeFields()` temporarily transforms the bunch into a beam frame whose origin is the reference particle and whose local `z` axis is aligned with the mean momentum.

The code builds:

```
CoordinateSystemTrafo beamToReferenceCSTrafo(
    Vector_t<double, 3>(0, 0, pc->get_sPos()),
    alignment.conjugate());

CoordinateSystemTrafo referenceToBeamCSTrafo =
    beamToReferenceCSTrafo.inverted();
```

Then it transforms particle positions to the beam frame, solves self-fields, and transforms positions and fields back to the reference frame.

Relevant file:

- `src/Algorithms/ParallelTracker.cpp`

Adaptive-Bin Rest Frames

The binned space-charge solver does not use a rigid coordinate transform for the rest frame. Instead it computes per-bin kinematics:

```
BinnedFieldSolver::BinKinematics {
    Vector_t<double, Dim> pmean;
    double gammaBin;
}
```

The solver treats the mesh field as a bin-frame electric field `E'`, applies the Lorentz scaling, and accumulates lab-frame `E` and `B`.

Relevant file:

- `src/PartBunch/BinnedFieldSolver.h`

38.7.1.4 Notes and Terminology Cleanup

The current implementation would be easier to reason about if the transform names were made explicit:

- `Lab2Element`
- `Element2Lab`
- `Reference2Lab`

- `Lab2Reference`
- `Reference2Element`
- `Element2Reference`

In the current code, `getCSTrafoLab2Local()` returns `getCSTrafoGlobal2Local()`. This is mathematically fine because `lab`, `global`, and `floor` are aliases here, but the naming obscures the intended frame chain.

38.7.2 Boosted Frenet-Serret frame

The next refinement step here should be:

1. align the boosted-frame notation with the field-solver sections,
2. decide how `K_s^\star` should be named in the final text,
3. connect the boosted-frame picture to the actual solver data flow.

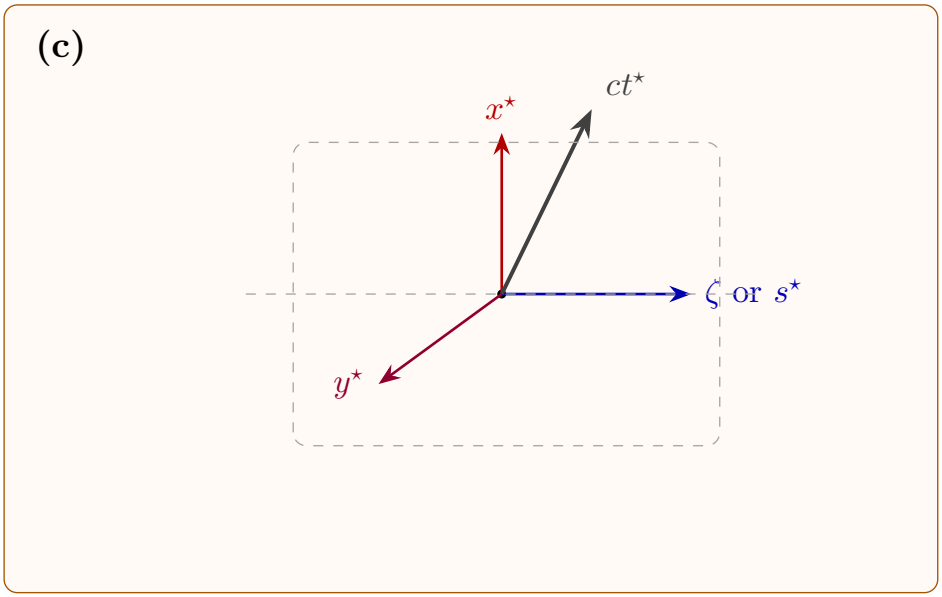


Figure 38.12: Boosted Frenet-Serret-style frame used as a draft field-solver frame. The marked point denotes the bunch centroid, the dashed horizontal line indicates the longitudinal beam direction, the heavy diagonal arrow indicates the boost direction, and the dashed rectangle indicates the solver box in the boosted frame.

38.8 Low Energy e-/e+ in a High Energy Coulomb Field

This sandbox note describes the present OPALX model for low-energy electron/positron witness particles born near an interaction point and moving through the field of a high-

energy primary bunch. The current implementation is useful as a first numerical experiment, but it is not yet a complete laboratory-frame treatment of a relativistic Coulomb field.

38.8.1 Implemented model

The implemented path is driven by a `BEAMBEAM` element. Container `0` is the source bunch. The optional attribute

```
WITNESS_CONTAINERS = "1,2"
```

selects passive witness containers. The default is

```
WITNESS_CONTAINERS = "NONE"
```

so legacy beam-beam tracking is unchanged unless witness kicks are requested.

When the source bunch enters the BeamBeam window, OPALX replaces the normal self-field mesh by a fixed interaction-window mesh. The longitudinal extent is the placed `BEAMBEAM` element length, centered on the interaction point. If an aperture is provided, the transverse half widths define the fixed x-y field domain. The source charge density is deposited on this mesh and the open Poisson problem is solved in the BeamBeam/source frame,

$$\nabla^2\phi_s(\mathbf{r}_s) = -\frac{\rho_s(\mathbf{r}_s)}{\epsilon_0}, \qquad \mathbf{E}_s(\mathbf{r}_s) = -\nabla\phi_s(\mathbf{r}_s).$$

The witness particles do not contribute to this charge density. They only sample the already-solved source field. For a witness particle stored relative to its own container reference path, the current code maps the longitudinal coordinate into the source frame with

$$z_s = z_w + (s_w - s_s),$$

where s_s is the source-container reference path length and s_w is the witness-container reference path length. The transverse coordinates are rotated with the same BeamBeam source-frame rotation used for the source bunch. The field is gathered on the fixed BeamBeam mesh and then rotated back to the reference frame before the normal Boris kick.

This gives a well-defined one-way coupling:

1. source container deposits charge,
2. source field is solved on the BeamBeam window mesh,
3. witness containers gather that field,
4. witness momenta are kicked,
5. witness charge never feeds back into the source solve.

This is the approximation currently used for the gamma-gamma pair-tracking prototype. It is a source-field sampling model, not yet a full transformation of the high-energy bunch field into the physical frame seen by the low-energy pairs.

38.8.2 Missing physics

The primary missing piece is the relativistic field transformation. The source solve is presently electrostatic in the BeamBeam/source frame. A high-energy bunch seen in the laboratory frame has compressed transverse electric fields and a magnetic field. Starting from a source rest-frame Coulomb field $\mathbf{E}'$, $\mathbf{B}' = 0$, the idealized boost along the source velocity $\beta_s c$ gives, schematically,

$$\mathbf{E}_{\parallel} = \mathbf{E}'_{\parallel}, \qquad \mathbf{E}_{\perp} = \gamma_s \mathbf{E}'_{\perp}, \qquad \mathbf{B} = \frac{1}{c}\,\beta_s \times \mathbf{E},$$

with signs fixed by the exact frame convention. The witness kick should then use the Lorentz force in a common physical frame,

$$\frac{d\mathbf{p}_w}{dt} = q_w \left(\mathbf{E}_{\mathrm{lab}} + \mathbf{v}_w \times \mathbf{B}_{\mathrm{lab}}\right).$$

For low-energy pairs this magnetic term and the difference between source, lab, and witness frames are not optional details: the witnesses can have large angles and broad velocities, so a scalar electrostatic kick in the source frame is only a first approximation.

The implementation also still treats the witness field sample as synchronous with the source field solve. A more complete model must define the time at which pairs are created, the source bunch phase at that time, and whether retardation or a quasi-static boosted-field approximation is sufficient for the target accuracy.

38.8.3 Work needed

The next implementation step is to make the frame contract explicit:

1. define the source field frame, the OPALX reference frame, and the witness container frame in one notation;
2. decide whether the source field should be stored as a rest-frame field plus boost metadata, or directly as a laboratory-frame $(\mathbf{E}, \mathbf{B})$;
3. apply the Lorentz-transformed field to witnesses in a common frame;
4. preserve the current one-way coupling switch so `WITNESS_CONTAINERS="NONE"` remains a true no-witness-space-charge mode.

The minimum validation set should include:

1. a boosted point-charge or Gaussian-bunch manufactured solution for $\mathbf{E}$ and $\mathbf{B}$;
2. a regression proving that witness containers receive a nonzero kick only when `WITNESS_CONTAINERS` is enabled;
3. a check that witness particles outside the BeamBeam aperture are counted and reported;
4. one-rank versus two-rank comparisons for the gamma-gamma pair input;

5. conservation and reproducibility diagnostics for the source container when witnesses are enabled, verifying that witness charge is not deposited on the BeamBeam source mesh.

Part V

Reference

39 Command Reference

The executable registers parser exemplars in `opalx/src/OpalConfigure/Configure.cpp`. This inventory is deliberately limited to that current OPALX registration point.

39.1 Actions

Command	Purpose
CALL	Read and execute another input source.
DUMPEMFIELDS	Request electromagnetic-field output.
ECHO	Write input-language text to the log.
HELP	Request parser help.
OPTION	Configure global runtime behavior.
SELECT	Select output or object state.
STOP, QUIT	Stop or leave input processing.
PSYSTEM, SYSTEM	Invoke supported system actions.
TITLE	Attach a title to the calculation.
TRACK	Enter a tracking block.
VALUE	Inspect or emit expression values.

39.2 Definitions

The current registry provides values, constants, variables, and macros, plus BEAM, DISTRIBUTION, EMISSIONSOURCE, and EMISSIONSOURCELIST, FIELDSOLVER, and BINNING. LINE assembles elements into a beamline. The complete statement catalog is in Input Language and Execution.

The following high-use interfaces already have source-audited tables:

- Beam Definitions
- Tracking
- Runtime Options
- Field Output Commands
- Elements
- Structures / Binning

40 Element Reference

The following objects are registered in the current OPALX configurator.

Family	Registered objects
RF and electric fields	`RFCAVITY`, `TRAVELINGWAVE`, `CONSTANTEFIELDCAVITY`, `VARIABLE_RF_CAVITY`
Transport	`DRIFT`, `QUADRUPOLE`, `RBEND`, `SBEND`, `SOLENOID`
Multipoles and specialized magnets	`MULTIPOLE`, `MULTIPOLET`, `VERTICALFFAMAGNET`
Sources and interactions	`LASER`
Diagnostics and structure	`MONITOR`, `MARKER`, `PROBE`, `LINE`
Time dependence	Polynomial, sinusoidal, and spline time-dependence objects

Capitalization above follows the user-facing convention. The Elements User Guide gives source-audited attributes for every registered type and distinguishes maintained regression coverage from source-only support. Its OPAL view also retains non-CYCL legacy interfaces; that does not imply OPALX registration. See Element Physics for detailed bends and multipoles and the API Reference for source-level definitions.

41 File Formats

OPALX consumes statement-oriented `.in` files plus model-specific field maps and particle distributions. It produces statistics, particle, field, loss, and diagnostic data through several writer paths.

The audited formats currently documented are:

- OPALX field-map types and retained OPAL formats
- `FROMFILE` and `EMITTEDFROMFILE` distribution records
- `DUMPEMFIELDS` output

Other output and restart formats remain **planned** until each reader or writer has been traced in the current source. A legacy OPAL variant is not an OPALX compatibility guarantee.

Each completed format page will specify:

- File extension and text or binary encoding.
- Units, coordinate frame, and column ordering.
- Required and optional metadata.
- Parallel write behavior.
- Versioning and backward-compatibility policy.
- A minimal reader and writer example.

42 Build Configuration

Pass configuration values to CMake with `-D<NAME>=<VALUE>`. For example:

```
cmake -S . -B build_cuda \
  -DBUILD_TYPE=Release \
  -DPLATFORMS=CUDA \
  -DARCH=AMPERE80 \
  -DOPALX_ENABLE_UNIT_TESTS=ON
```

Reconfigure from a clean build directory when changing compilers, MPI implementations, execution backends, or GPU architectures. CMake cache values from one toolchain should not be reused with another.

42.1 Build and platform selectors

Flag	Default	Accepted values and effect
`BUILD_TYPE`	`Release`	`Debug`, `Release`, `RelWithDebInfo`, or `MinSizeRel`. This value sets `CMAKE_BUILD_TYPE`.
`PLATFORMS`	`SERIAL`	Semicolon-separated selection from `SERIAL`, `OPENMP`, `CUDA`, `HIP`, and `SYCL`. CUDA and HIP cannot be enabled together.
`ARCH`	empty	Enables the matching `Kokkos_ARCH_*` setting. CUDA builds require an architecture such as `AMPERE80`; SYCL uses an appropriate `INTEL_*` value.

Flag	Default	Accepted values and effect
BUILD_SHARED_LIBS	OFF	Build the OPALX library as shared rather than static.

Supported ARCH values are grouped by target family:

- NVIDIA GPU: HOPPER90, AMPERE80, AMPERE86, AMPERE87, VOLTA70, VOLTA72, PASCAL61, PASCAL60, and ADA102.
- CPU: BDW, SKX, CNL, ZEN, ZEN2, ZEN3, ARM80, and ARM81.
- Intel accelerator: INTEL_GEN9, INTEL_GEN11, INTEL_GEN12LP, INTEL_DG1, INTEL_XEHP, and INTEL_PVC.

For CUDA, OPALX derives CMAKE_CUDA_ARCHITECTURES from the numeric suffix of ARCH. HIP builds instead require an explicit -DCMAKE_HIP_ARCHITECTURES=<arch>. SYCL architecture configuration also depends on the selected compiler and environment.

42.2 OPALX feature flags

Flag	Default	Effect when enabled
OPALX_EMBED_BUILD_METADATA	OFF	Embed the build user, machine, and date in BuildInfo.h. Leave off to avoid rebuilds caused only by changing metadata.
OPALX_ENABLE_UNIT_TESTS	OFF	Build the GoogleTest unit-test targets.
OPALX_ENABLE_EXAMPLES	OFF	Build the example module.
OPALX_ENABLE_TESTS	OFF	Build integration tests from the test directory.
OPALX_ENABLE_COVERAGE	OFF	Add coverage instrumentation with a supported GNU or Clang compiler.
OPALX_ENABLE_SANITIZER	OFF	Enable sanitizer instrumentation for supported debug builds.

Flag	Default	Effect when enabled
OPALX_ENABLE_NSYS_PROFILER	OFF	Enable NVIDIA Nsight Systems integration. This is valid only with PLATFORMS=CUDA.
OPALX_ENABLE_HIP_PROFILER	OFF	Reserved for HIP profiler integration. The current configuration rejects ON because support is not implemented yet.
OPALX_USE_ALTERNATIVE_VARIANT	OFF	Use the modified variant implementation retained for CUDA 12.2 and GCC 12.3 compatibility.
OPALX_USE_STANDARD_FOLDERS	OFF	Place generated executables and libraries under build-tree bin/ and lib/ directories.
OPALX_SKIP_FAILING_TESTS	OFF	Exclude tests currently marked as failing from the build and test run.
OPALX_ENABLE_SCRIPTS	OFF	Generate supported benchmark and test job-script templates.
OPALX_FIELD_DEBUG	OFF	Compile field-solver field-dump diagnostics. This can create additional output during a simulation.
OPALX_USE_KOKKOS_MATH_CONSTANTS	ON	Source constants such as pi, e, and log10e from Kokkos::numbers; set OFF for literal fallback values.

Enable a Boolean option with -D<FLAG>=ON. Do not enable coverage, sanitizers, or profiler integration in a production build unless those tools are intentionally required.

42.3 Installed dependencies

OPALX fetches and builds these dependencies by default. Set a switch to `ON` only when a compatible installation is already available in the active toolchain environment.

Flag	Default	Effect when enabled
`OPALX_USE_INSTALLED_HDF5`	`OFF`	Find a system HDF5 installation instead of fetching HDF5. Version 1.10.0 or newer is required.
`OPALX_USE_INSTALLED_H5HUT`	`OFF`	Find a system H5hut installation instead of fetching H5hut.
`OPALX_USE_INSTALLED_GTEST`	`OFF`	Find system GoogleTest when unit tests are enabled instead of fetching GoogleTest.

The installed HDF5 and H5hut packages must match the MPI implementation and compiler used for OPALX. A package built against a different MPI library can configure successfully and still fail at link or runtime.

42.4 Dependency versions and source revisions

Flag	Default	Effect
`IPPL_GIT_TAG`	`master`	IPPL branch, tag, commit, or release to fetch. A numeric value such as `3.2.0` resolves to `IPPL-3.2.0`.
`Kokkos_VERSION`	`5.1.1`	Kokkos version requested through IPPL. Prefix a tag, branch, or commit with `git.` to request that source revision.
`Heffte_VERSION`	`2.4.0`	heFFTe version requested through IPPL when FFT support is enabled. The same `git.<ref>` form selects a source revision.

Plain Kokkos and heFFTe versions allow a compatible installed package of that version or newer. Pin a source revision only for reproducible testing or when a specific upstream fix is required.

42.5 IPPL settings managed by OPALX

OPALX currently enables IPPL FFT and solver support, forwards `PLATFORMS`, and disables IPPL Alpine support and IPPL's own tests. These values are set by OPALX rather than being independent user options:

README flag	Current CMake variable	Value
`IPPL_ENABLE_ALPINE`	`IPPL_ENABLE_ALPINE`	`OFF`
`IPPL_ENABLE_TEST`	`IPPL_ENABLE_TESTS`	`OFF`

The singular `IPPL_ENABLE_TEST` spelling appears in the OPALX README; the actual variable forwarded to IPPL is `IPPL_ENABLE_TESTS`. Because OPALX sets these cache values with `FORCE`, command-line overrides are not currently supported.

42.6 Inspect a configured build

After configuration, use CMake's cache listing to confirm the effective values:

```
cmake -N -LA build_cuda
```

The configure log should identify the selected build type, execution backend, architecture, dependency sources, output layout, and high-signal feature toggles. Delete and recreate the build directory if those values do not match the intended toolchain.

43 API Reference

The source-level OPALX API reference is generated and published separately from this manual. This keeps the curated documentation small while allowing the API site to follow the source code closely.

Open the OPALX Doxygen documentation

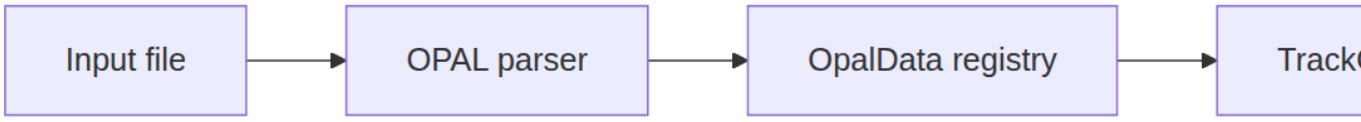
The API site lives at PSI and is not copied into this Quarto build. Use the command and element reference in this manual for user-facing input syntax, and the Doxygen site for C++ classes, functions, and source-level relationships.

Part VI

Developer Guide

44 Architecture

OPALX keeps the statement-driven OPAL object model at its boundary while using IPPL and Kokkos for distributed, performance-portable particle and field work.



The configurator creates the command and element exemplars available to the parser. `ParallelTracker` coordinates the time step, while local kernels such as the Boris pusher operate on particle state. `OrbitThreader` determines active elements and their local transforms. IPPL provides distributed data structures and solvers; Kokkos maps kernels to CPU and GPU backends.

GPU-callable code must avoid host-only state. In particular, enclosing methods used by CUDA device lambdas need public accessibility and all data captured by device kernels must have a valid device representation.

45 Build, Test, and CI

Use a debug build with unit tests for local development:

```
cmake -S . -B build_serial \
  -DBUILD_TYPE=Debug \
  -DPLATFORMS=SERIAL \
  -DOPALX_ENABLE_UNIT_TESTS=ON
cmake --build build_serial -j 8
ctest --test-dir build_serial --output-on-failure
```

Repeat relevant tests with `OPENMP` and the target accelerator backend. A test that passes only on the serial backend is not sufficient for a performance-portable change.

Pull-request CI includes serial, OpenMP, CUDA, and HIP compilation, with SYCL coverage where the runner supports it. Compile workflows run on labeled, non-draft pull requests to `master`; use the `compile-ci` label when the change is ready for those checks. Fast regression cases are executed with the serial artifact and compared to maintained metrics.

46 Adding Features

46.1 Parser-visible commands and elements

1. Implement the object and its attributes in the appropriate OPALX module.
2. Register a prototype in `opalx/src/OpalConfigure/Configure.cpp`.
3. Add parser and behavior tests.
4. Verify serial and accelerator builds.
5. Add the object to the user and reference documentation in the same change.

46.2 Physics features

Document the governing equations, units, coordinate frame, numerical method, step ordering, and known approximations. Add a manufactured solution, convergence study, or benchmark that makes failures observable.

46.3 Performance portability

- Keep kernels valid for CPU and GPU execution.
- Avoid host-only calls and non-device captures inside device lambdas.
- Make required enclosing functions public for CUDA compilation.
- Test MPI decomposition as well as single-rank behavior.
- Document backend-specific constraints instead of silently falling back.

47 Documentation Workflow

Every substantive page declares `title`, `description`, `audience`, `status`, and `last-reviewed`. Mark incomplete work visibly as `planned`; do not fill a gap by copying an unverified historical OPAL chapter.

47.1 Local preview

```
quarto preview --profile opalx
```

The `opalx` profile targets the production `OPALX-project/opalx-manual` and `OPALX-project/opalx-documents` repositories. For a one-off private fork, copy `_quarto.yml.local.example` to `_quarto.yml.local` and set its `documents-base-url`. The local file is ignored by Git.

47.2 Linking binary documents

Place OPALX presentations, reports, examples, and datasets in the category-first `opalx-documents` tree. Link them only through project metadata:

```
[OPALX overview](https://github.com/OPALX-project/opalx-documents/blob/main/presentations/
```

Project metadata supplies `documents-base-url`, so page content does not embed repository locations directly.

Report summaries live in the nested `resources/reports/` Quarto project. Add a detail page beneath its year directory and link it from `resources/presentations-reports.qmd`; do not add individual reports to the numbered book chapters. The nested archive is rendered after the main HTML site and shares the selected deployment profile.

47.3 Asset policy

PDFs, Office documents, archives, and media are rejected from the manual. Raster figures must remain at or below 500 KiB. Source SVG, TikZ, Mermaid, small plots, and reproducible text scripts may remain with the page they support. Generated Quarto output is never committed. The separately hosted Doxygen API reference is linked rather than copied into this manual.

Part VII

Troubleshooting

48 Common Problems

48.1 CMake cannot find a compiler or MPI

Load a consistent compiler and MPI toolchain, remove only the affected build directory, and configure it again. Do not reuse a build directory created with a different backend.

48.2 CUDA or HIP configuration fails

Check that `PLATFORMS` and `ARCH` match the installed toolkit and physical GPU. Confirm that the host compiler is supported by that toolkit.

48.3 The input parser rejects a statement

Start from a passing OPALX regression input. Check semicolons, object names, attribute spelling, and referenced paths. Historical OPAL examples may use features that are not registered in OPALX.

48.4 Several MPI ranks select the same GPU

Launch one rank per GPU and add `--kokkos-map-device-id-by=mpi_rank`.

48.5 A run completes but results change with the backend

Reduce the time step, repeat with a finer mesh, record rank and device counts, and compare a small deterministic case across serial and accelerator builds.

48.6 A bunch splits near a hard-edge field

A discontinuous fringe field can make nearby particles receive different numbers of kicks, especially when their path lengths differ. The numerical artifact can look like a physically split bunch. Prefer a resolved fringe model where one is available and repeat the run with a smaller time step.

48.7 A very short active element gives step-dependent results

If an active element is comparable in length to one integration step, some particles can enter or leave it on different steps and receive different numbers of kicks. Reduce the time step until the observable of interest is stable. Treat that convergence check as part of the simulation setup.

49 Bug Reports and Investigations

A useful bug report contains:

- OPALX commit or release and whether the tree has local changes.
- Compiler, MPI, platform backend, GPU architecture, and CMake options.
- Rank, thread, and device counts.
- The smallest input and supporting files that reproduce the problem.
- Exact launch command, complete error output, and expected behavior.
- Whether serial, OpenMP, and accelerator builds behave differently.
- A convergence comparison when the issue is numerical rather than a crash.

Do not attach private production data when a reduced synthetic input can show the defect. Large OPALX investigation files belong in `opalx-documents/reports/<year>/investigations/`, not in this manual.

Open an OPALX issue

50 Known Limitations

- OPALX is not documented as a drop-in implementation of every historical OPAL flavor. OPAL-CYCL and OPAL-MAP material is not part of this manual.
- Registered but regression-untested interfaces are identified as source-supported; that label does not claim regression coverage.
- Some physics chapters document recently implemented models whose interface may still change; these pages are marked `experimental`.
- Backend support must be checked for each new algorithm; successful serial execution does not establish GPU support.
- Hard-edge fields and active elements shorter than a few integration steps can create numerical splitting. Always check time-step convergence.

These limitations are documentation status, not a substitute for the issue tracker. Report unexpected implemented behavior with a reproducible case.

Part VIII

Resources

51 Publications and Citation

The bibliography below is curated from the integrated OPALX physics chapters. A release-specific software citation and DOI will be added when the redesigned documentation is published as the canonical manual.

When reporting results today, record the exact OPALX revision and cite both the software repository and the publications associated with the physical model being used.

51.1 Student Projects and Contributions

All projects and associated resources in the following table are published in the AMAS completed-projects archive. The list includes projects that used OPAL or OPALX directly, or contributed models, algorithms, and software components incorporated into them.

	Year	Title	Student
	2025	Adaptive Energy Binning in OPALX	Alexander Liemen
	2024	A Dual-Space Multilevel Kernel-Splitting Algorithm for the Open Poisson Equation	Ryan Ammann
	2023	Accelerator Net	Pranas Juknevicius
	2022	On understanding the differences in measured and calculated energy spread in the SwissFEL injector	Garðar Árni Skarphéðinsson
	2021	A Performance Portable Poisson Solver for the Hose Instability	Sonali Mayani
	2020	Accelerating accelerators: Fast surrogate models for beam prediction	Renato Bellotti
	2020	On the Modelling of Collisions in Cold Particle Electron Sources	Tim Wyssling

	Year	Title	Student
	2020	Start-to-End Modelling of the AWA Microbunched Electron Cooling POP-Experiment	Arnau Albà
	2020	Precise Simulations of Multibunches in High Intensity Cyclotrons	Matthias Frey
	2018	Limitations of a linear transfer map method for finding matched distributions in high intensity cyclotrons	Edgar Cristopher Cortés García
	2018	Precise Beam Dynamics Models for Transport Lines in a Cyclotron-based Proton Therapy Facility	Valeria Rizzoglio
	2017	Future Processor Hardware Architectures for the Benefit of Precise Particle Accelerator Modeling	Uldis Locans
	2016	Central Region Design of a Compact High Intensity Cyclotron	Jakob Jonnerby
	2016	The P3M Model on Emerging Computer Architectures with Application to Microbunching	Benjamin Ulmer
	2016	Improving Single Core Performance in OPAL	Lionel Miserez
	2016	Space Charge Aspects of Particle Transport	A. Kolano
	2015	Performance Analysis of MKL Fast Fourier Transform on Intel Xeon Phi	Benjamin Ulmer et al.
	2015	Performance Analysis of MKL Fast Fourier Transform and FFT-based Poisson Solver on Intel Xeon Phi	Benjamin Ulmer et al.

	Year	Title	Student
	2015	Matched Distributions in Cyclotrons with Higher Order Moments of the Charge Distribution	Matthias Frey
	2014	Matched Distributions in Cyclotrons	Matthias Frey
	2014	A Self-Consistent Particle-In-Cell Time-Domain Solver Incorporating Radiative Interaction	Christof Metzger-Kraus
	2013	Double Degrader for Proton Therapy	Helene Stachel
	2013	Modeling of the COMET Cyclotron in OPAL and Switching Improvements of the Beam Intensity	Jeroen Anthonius Veenendaal
	2013	Parallelization of a Differential Algebra Framework	Matthias Frey
	2013	A Homotopy Method for Large-Scale Multi-Objective Optimization	Andrew Foster
	2013	Toward Massively Parallel Multi-Objective Optimization with Application to Particle Accelerators	Yves Ineichen
	2012	Towards Large-Scale Simulation-Based Multi-Objective Optimization	Andrew Foster
	2011	An Adaptive Time Integration Method for more Efficient Simulation of Particle Accelerators	Matthias Toggweiler
	2011	Towards quantitative simulations of high power proton cyclotrons	Yuanjie Bi
	2010	Beam dynamics in high intensity cyclotrons including neighboring bunch effects	Jianjun Yang

	Year	Title	Student
	2008	A Parallel Multigrid Solver for Beam Dynamics	Yves Ineichen
	2008	Short-range Wakefield Model Implementation in OPAL	Stefan Pauli

References

[1] W. Joho, "Representation of beam ellipses for transport calculations," Paul Scherrer Institute, TM-11-14, 1980. Available: https://indico.psi.ch/event/3484/attachments/5948/7502/TM-11-14.pdf

[2] J. G. Power and C. Jing, "Temporal laser pulse shaping for RF photocathode guns: The cheap and easy way using UV birefringent crystals," in *AIP conference proceedings*, 2009, p. 689. doi: 10.1063/1.3080991.

[3] K. Fl"ottmann, "Note on the thermal emittance of electrons emitted by cesium telluride photo cathodes," Deutsches Elektronen-Synchrotron, DESY-TESLA-FEL-97-01, 1997.

[4] J. E. Clendenin *et al.*, "Reduction of thermal emittance of RF guns," *Nuclear Instruments and Methods in Physics Research Section A*, vol. 455, p. 198, 2000, Available: https://www.sciencedirect.com/science/article/pii/S0168900200007312

[5] D. H. Dowell and J. F. Schmerge, "Quantum efficiency and thermal emittance of metal photocathodes," *Physical Review Special Topics - Accelerators and Beams*, vol. 12, p. 074201, 2009, doi: 10.1103/PhysRevSTAB.12.074201.

[6] J. E. Spencer and H. A. Enge, "Split-pole magnetic spectrograph for precision nuclear spectroscopy," *Nuclear Instruments and Methods*, vol. 49, pp. 181–193, 1967, doi: 10.1016/0029-554X(67)90684-2.

[7] J. H. Billen and L. M. Young, "POISSON SUPERFISH," Los Alamos National Laboratory, LA-UR-96-1834, 2004.

[8] J. Qiang, S. M. Lidia, R. D. Ryne, and C. Limborg-Deprey, "A three-dimensional quasi-static model for high brightness beam dynamics simulation," Lawrence Berkeley National Laboratory, LBNL-59098, 2005. Available: http://repositories.cdlib.org/lbnl/LBNL-59098

[9] J. Qiang, S. M. Lidia, R. D. Ryne, and C. Limborg-Deprey, "Three-dimensional quasi-static model for high brightness beam dynamics simulation," *Physical Review Special Topics - Accelerators and Beams*, vol. 9, p. 044204, 2006, doi: 10.1103/PhysRevSTAB.9.044204.

[10] J. Qiang, S. M. Lidia, R. D. Ryne, and C. Limborg-Deprey, "Erratum: Three-dimensional quasi-static model for high brightness beam dynamics simulation," *Physical Review Special Topics - Accelerators and Beams*, vol. 10, p. 129901, 2007, doi: 10.1103/PhysRevSTAB.10.129901.

[11] G. Fubiani, S. M. Lidia, J. Qiang, R. D. Ryne, and C. Limborg-Deprey, "Space charge modeling of dense electron beams with large energy spreads," *Physical Review Special Topics - Accelerators and Beams*, vol. 9, p. 064402, 2006, doi: 10.1103/PhysRevSTAB.9.064402.

[12] C. K. Birdsall and A. B. Langdon, *Plasma physics via computer simulation*. New York: McGraw-Hill, 1985.

[13] "Tait-bryan angles." Wikipedia. Available: https://en.wikipedia.org/wiki/Euler_angles#Tait.E2.80.93Bryan_angles

[14] E. Forest and K. Hirata, "A contemporary guide to beam dynamics," KEK, KEK Report 92-12, 1992.

[15] W. Herr and F. Schmidt, *A MAD-x primer*. CERN, 2004.

[16] P. K. Skowronski, F. Schmidt, and E. Forest, "Advances in MAD-x using PTC," CERN, LHC Project Report 1016, 2007.

[17] M. Borland, "User's manual for elegant." Advanced Photon Source online manual.

[18] A. J. Dragt, E. Forest, L. M. Healy, P. Schütt, and J. van Zeijts, *MARYLIE 3.0 users' manual*. 2003.

[19] J. Qiang, "IMPACT-t user document, version 2.2." 2022.

[20] D. Sagan, "Bmad manual."

[21] D. Nguyen, J. Lewellen, and L. Duffy, "RF linac for high-gain FEL: Bunch compression." US Particle Accelerator School, Jun. 2014. Available: https://uspas.fnal.gov/materials/14UNM/E_Bunch_Compression.pdf

[22] S. Di Mitri, "Bunch-length compressors," in *Proceedings of the CAS-CERN accelerator school on free electron lasers and energy recovery linacs*, vol. 1, in CERN yellow reports: School proceedings, vol. 1., 2018. doi: 10.23730/CYRSP-2018-001.361.

[23] J. D. Jackson, *Classical electrodynamics*, 3rd ed. New York: Wiley, 1999.

[24] Particle Data Group, "Review of particle physics: Muon listings." Particle Data Group, 2024. Available: https://pdg.lbl.gov/2024/listings/rpp2024-list-muon.pdf

[25] Particle Data Group, "Review of particle physics: Charged pion listings." Particle Data Group, 2024. Available: https://pdg.lbl.gov/2024/listings/rpp2024-list-pi-plus-minus.pdf

[26] Particle Data Group, "Review of particle physics: kinematics." Particle Data Group, 2024. Available: https://pdg.lbl.gov/2024/reviews/rpp2024-rev-kinematics.pdf

[27] Particle Data Group, "Review of particle physics: Muon decay parameters." Particle Data Group, 2024. Available: https://pdg.lbl.gov/2024/reviews/rpp2024-rev-muon-decay-params.pdf

[28] V. Bargmann, L. Michel, and V. L. Telegdi, "Precession of the polarization of particles moving in a homogeneous electromagnetic field," *Physical Review Letters*, vol. 2, no. 10, pp. 435–436, 1959, doi: 10.1103/PhysRevLett.2.435.

52 Presentations and Reports

This catalog is the manual's single entry point for OPALX reports and presentations. Entries are grouped by year and sorted newest first. Each link opens an unnumbered detail page with a searchable text summary and a link to the original file in `opalx-documents`.

The detail pages are deliberately outside the numbered book structure. Adding reports therefore does not expand the manual sidebar, renumber its chapters, or add an unbounded archive to the downloadable PDF.

2026

Date	Entry	Kind	Status
2026-07-14	CUDA extended-lambda linkage bug	Bug investigation	Resolved with workaround
2026-07-06	Implicit `this` capture in IPPL field assignment	Bug investigation	Resolved in IPPL #561
2026-04-30	OpalElement layout proposal	Design report	Experimental
2026-04-06	OPALX overview	Architecture presentation	Current

52.1 Add an entry

Create the detail page under `resources/reports/YYYY/` and add it to the matching year above. The report archive renders those pages separately, so no per-report entry belongs in the book chapters in `_quarto.yml`. Put the original PDF or presentation in the matching category and year in `opalx-documents`, then link it through `documents-base-url`.

The Documentation Workflow contains the link template and asset rules. Supporting datasets and examples should remain linked from the manual page that explains how they are used.

53 Regression Tests

The maintained regression suite is the best source of complete input files that exercise the current parser and implementation.

Open the live OPALX regression-test dashboard

When documenting a feature, link its regression case and explain what metric the test protects. A test that only checks process completion should be strengthened with a physically meaningful observable where practical.