Physics Department,
Technical University of Denmark
2800 Kongens Lyngby, Denmark

# Component Manual for the Xray-Tracing Package McXtrace, version 3.5

E. B. Knudsen, A. Prodi, J. Baltser, P. Willendrup,
A. Vickery, E. Farhi, K. Lefmann, S. Schmidt

September, 2024

The software package McXtrace is a tool for carrying out highly complex Monte Carlo ray-tracing simulations of X-ray beamlines to high precision. The simulations can compute all aspects of the performance of instruments and can thus be used to optimize the use of existing equipment, design new instrumentation, and carry out virtual experiments for e.g. training, experimental planning or data analysis. McXtrace is based is based on a unique design, inhereted from its sister McStas, where an automatic compilation process translates high-level textual instrument descriptions into efficient ANSI-C code. This design makes it simple to set up typical simulations and also gives essentially unlimited freedom to handle more unusual cases. This report constitutes the component manual for McXtrace, and, together with the manual for the McXtrace system, it contains full documentation of all aspects of the program. It covers a description of all official components of the McXtrace package with some theoretical background. Selected test instruments and representative McXtrace simulations performed with these instruments are described in the User Manual.

This report documents the components for McXtrace version 3.5, released September, 2024.

The authors are:

Erik Bergbäck Knudsen
Physics Department, Techical Univerisity of Denmark, Kgs. Lyngby, Denmark
email: erkn@fysik.dtu.dk

Andrea Prodi
Niels Bohr Institute, University of Copenhagen, Denmark
European Synchrotron Radiation Facility, Grenoble, France
email: aprodi@nbi.ku.dk

Peter Kjær Willendrup
Physics Department, Techical Univerisity of Denmark, Kgs. Lyngby, Denmark
email: pkwi@fysik.dtu.dk

Kim Lefmann
Niels Bohr Institute, University of Copenhagen, Denmark
email: lefmann@fys.ku.dk

Emmanuel Farhi
Synchrotron SOLEIL, Saint-Aubin, France
email: emmanuel.farhi@synchrotron-soleil.fr

# Contents

# Preface and acknowledgements

This document contains information on the x-ray scattering components which are the building blocks for defining instruments in the Monte Carlo Xray-tracing program McXtrace version 3.5. The initial release in June 2011 of version 1.0 was presented in Ref. [LN99]. The reader of this document is not supposed to have specific knowledge of xray scattering, but some basic understanding of physics is helpful in understanding the theoretical background for the component functionality. For details about setting up and running simulations, we refer to the McXtrace system manual [Knu+14]. We assume familiarity with the use of the C programming language.

We would like to explicitly thank all the partners in this project:

- The European Synchrotron Radiation Facility (ESRF), Grenoble, France
- SAXSLAB Aps., Lundtofte, Denmark
- Risø DTU, Roskilde, Denmark
- Niels Bohr Institute, Univeristy of Copenhagen, Copenhagen, Denmark.
- Faculty for Life Sciences, University of Copenhagen, Copenhagen, Denmark.

As McXtrace has inherited much of its functionality from its sister McStas we take the oppurtunity to thank Dir. Kurt N. Clausen, PSI, for his continuous support to McStas and for having initiated the project. Continuous support to McStas has also come from Prof. Robert McGreevy, ISIS.

We have further benefited from discussions with many other people in the scattering community, too numerous to mention here.

The users who contributed components to this manual are acknowledged as authors of the individual components. We encourage other users to contribute components with manual entries for inclusion in future versions of McXtrace.

In case of errors, questions, or suggestions, do not hesitate to contact the user/developer community by writing to the user mailiing list `mcxtrace-users@mcxtrace.órg` or consult the McXtrace home page [Mcx]. A special developement website (shared with the sister project McStas) complete with bug/request reporting service is available [Tra].

We would like to kindly thank all McXtrace component contributors. This is the way we improve the software alltogether.

If you **appreciate** this software, please subscribe to the `mcxtrace-users@mcxtrace.org` email list, send us a smiley message, and contribute to the package.

We also encourage you to refer to this software when publishing results, with the following citation:

- E. B. Knudsen, et.al, J. Applied Cryst. **46**, pp 679, 2013.

# 1. About the component library

This McXtrace Component Manual consists of the following major parts:

- An introduction to the use of Monte Carlo methods in McXtrace.
- A thorough description of system components, with one chapter per major category: Sources, optics, monochromators, samples, monitors, and other components.
- The McXtrace library functions and definitions that aid in the writing of simulations and components in Appendix A.
- An explanation of the McXtrace terminology in Appendix B.

Additionally, you may refer to the list of example instruments from the library in the McXtrace User Manual.

## 1.1. Authorship

The component library is maintained by the McXtrace system group. A number of basic components "belongs" the McXtrace system, and are supported and tested by the McXtrace team.

Other components are contributed by specific authors, who are listed in the code for each component they contribute as well as in this manual. McXtrace users are encouraged to send their contributions to us for inclusion in future releases.

## 1.2. Symbols for x-ray scattering and simulation

In the description of the theory behind the component functionality we will use the usual symbols $\mathbf{r}$ for the position $(x, y, z)$ of the particle (unit m), and $\mathbf{k}$ for the particle wave vector $(k_x, k_y, k_z)$ (unit $\text{Å}^{-1}$). The wavelength is the reciprocal wave vector, $\lambda = 2\pi/k$. By convention energy is usually given i keV and may be calculated from the wavelength by: $\lambda = \frac{12.398}{E}$

In general, vectors are denoted by boldface symbols.

Subscripts "i" and "f" denotes "initial" and "final", respectively, and are used in connection with the photon state before and after an interaction with the component in question.

| MCXTRACE/data | Description |
|---|---|
| *.lau | Laue pattern file, as issued from Crystallographica. For use with Single_crystal and PowderN. Data: [ h k l Mult. d-space 2Theta F-squared ] |
| *.laz | Powder pattern file, as obtained from Lazy/ICSD. For use with PowderN. |
| *.txt | General text file, containing ascii data. Currently used for elemental data extracted from NIST[Nis]. |

Table 1.1.: Data files of the McXtrace library.

## 1.3. Component coordinate system

All mentioning of component geometry refer to the local coordinate system of the individual component. The axis convention is so that the $z$ axis is along the photon propagation axis, the $y$ axis is vertical up, and the $x$ axis points left when looking along the $z$-axis, completing a right-handed coordinate system. Most components 'position' (as specified in the instrument description with the `AT` keyword) corresponds to their volume centre. The mirrors are an important counterexmaple. In this case the 'position' is the centre of the mirror surface

Components are not necessarily designed to overlap. This may lead to loss of rays. Warnings will be issued during simulation if sections of the instrument are not reached by any xrays, or if a significant number of xrays are removed. This is usually the sign of either overlapping components or low intensity.

## 1.4. About data files

Some components require external data files, e.g. lattice crystallographic definitions for Laue and powder pattern diffraction, absorption and reflectivity files, etc.

Such files distributed with McXtrace are located in the `data` sub-directory of the `MCXTRACE` library. Components that make use of the McXtrace file system, including the `read-table` library (see section A.2) may access all McXtrace data files without making local copies. Of course, you are welcome to define your own data files, and eventually contribute to McXtrace if you find them useful.

File extensions are not interpreted by McXtrace but help in identifying relevant files per application. Table 1.1 lists the current default file extensions.

McXtrace itself generates both simulation and monitor data files, the structure of which is explained in the User Manual (see end of chapter 'Running McXtrace').

## 1.5. Component source code

Source code for all components may be found in the `MCXTRACE` library subdirectory of the McXtrace installation; the default is `/usr/local/lib/mcxtrace/` on Unix-like systems and `C:\mcxtrace\lib` on Windows systems, but it may be changed using the `MCXTRACE` environment variable.

In case users only require to add new features, preserving the existing features of a component, using the `EXTEND` keyword in the instrument description file is recommended. For larger modification of a component, it is advised to make a copy of the component file into the working directory. A component file in the local directory will in McXtrace takes precedence over a library component of the same name.

## 1.6. Documentation

As a complement to this Component Manual, we encourage users to use the `mxdoc` front-end which enables to display both the catalog of the McXtrace library, e.g using:

```
mxdoc
```

as well as the documentation of specific components, e.g with:

```
mxdoc --text name
mxdoc file.comp
```

The first line will search for all components matching the *name*, and display their help section as text. For instance, `mxdoc .laz` will list all available Lazy data files, whereas `mxdoc --text Monitor` will list most Monitors. The second example will display the help corresponding to the *file.comp* component, using your BROWSER setting, or as text if unset. The `--help` option will display the command help, as usual.

An overview of the component library is also given at the McXtrace home page [Mcx] and in the User Manual [Knu+14].

## 1.7. Disclaimer, bugs

We would like to emphasize that the usage of both the McXtrace software, as well as its components are the responsability of the users. Indeed, obtaining accurate and reliable results requires a substantial work when writing instrument descriptions. This also means that users should read carefully both the documentation from the manuals [Knu+14] and from the component itself (using `mcdoc` *comp*) before reporting errors. Most anomalous results often originate from a wrong usage of some part of the package.

Anyway, if you find that either the documentation is not clear, or the behavior of the simulation is undoubtedly anomalous, you should report this to us at `mcxtrace-users@mcxtrace.org` and/or refer to our special bug/request reporting service [Mcz].

# 2. Monte Carlo Techniques and simulation strategy

This chapter explains the simulation strategy and the Monte Carlo techniques used in McXtrace. We first explain the concept of the x-ray weight factor, and discuss the statistical errors in dealing with sums of x-ray weights. Secondly, we give an expression for how the weight factor transforms under a Monte Carlo choice and specialize this to the concept of direction focusing. Finally, we present a way of generating random numbers with arbitrary distributions. More details are available in the Appendix concerning random numbers in the User manual.

## 2.1. X-ray simulations

X-ray scattering beamlines are built as a series of optical elements. Each of these elements modifies the beam characteristics (e.g. divergence, wavelength spread, spatial and temporal distributions) in a way which, for simple x-ray beam configurations, may be modelled with analytical methods.

However, real x-ray beamlines consist of a large number of optical elements, and this brings additional complexity by introducing strong correlations between x-ray beam parameters like divergence and position - which is the basis of the acceptance diagram method - but also wavelength and time. The usual analytical methods, such as phase-space theory, then reach their limit of validity in the description of the resulting effects.

In order to cope with this difficulty, Monte Carlo (MC) methods (for a general review, see Ref. [Jam80]) may be applied to the simulation of x-ray beamlines. The use of probability is commonplace in the description of microscopic physical processes. Integrating these events (absorption, scattering, reflection, ...) over the x-ray trajectories results in an estimation of measurable quantities characterizing the beamline. Moreover, using variance reduction (importance sampling) where possible, reduces the computation time and gives better accuracy.

Implementations of the MC method for X-ray beamlines already exist, most notable is probably *SHADOW* [WCC94], originally developed by the late Franco Cerrina and coworkers, now developed further by M. Sanchez Del Rio at the ESRF[Rio+11][Sha]. Other implementations of the same concept are *RAY* [Sch08] from BESSY and *Xtrace*[Bau+07]. hosted at ANKA

### 2.1.1. Monte Carlo ray tracing simulations

Mathematically, the Monte-Carlo method is an application of the law of large numbers [Jam80; GRR92]. Let $f(u)$ be a finite continuous integrable function of parameter $u$ for which an integral estimate is desirable. The discrete statistical mean value of $f$ (computed as a series) in the uniformly sampled interval $a < u < b$ converges to the mathematical mean value of $f$ over the same interval.

$$\lim_{n\to\infty} \frac{1}{n} \sum_{i=1, a\leq u_i\leq b}^{n} f(u_i) = \frac{1}{b-a} \int_a^b f(u)du \tag{2.1}$$

In the case were the $u_i$ values are regularly sampled, we come to the well known midpoint integration rule. In the case were the $u_i$ values are randomly (but uniformly) sampled, this is the Monte-Carlo integration technique. As random generators are not perfect, we rather talk about *quasi*-Monte-Carlo technique. We encourage the reader to refer to James [Jam80] for a detailed review on the Monte-Carlo method.

## 2.2. The x-ray weight

A totally realistic semi-classical simulation will require that each x-ray is at any time either present or lost. On many beamlines the sheer abundance of x-ray photons makes it impractical to trace each and every photon from the source. This is particularly the case at XFELs. Additionally, only a very small fraction of the initial x-rays will ever be detected, and simulations of this kind will therefore waste much time in dealing with x-rays that never hit the detector.

A way of dealing with these issues and speed up calculations is to introduce a x-ray "weight factor" for each simulated ray and to adjust this weight according to the path of the ray. If *e.g.* the reflectivity of a certain optical component is 10%, and only reflected x-rays ray are considered later in the simulations, the x-ray weight will be multiplied by 0.10 when passing this component, but every x-ray is allowed to reflect in the component. In contrast, the totally realistic simulation of the component would require on average ten incoming x-rays for each reflected one.

Let the initial x-ray weight be $p_0$ and let us denote the weight multiplication factor in the $j$'th component by $\pi_j$. The resulting weight factor for the x-ray ray after passage of the whole beamline becomes the product of all contributions

$$p = p_n = p_0 \prod_{j=1}^{n} \pi_j. \tag{2.2}$$

Each adjustement factor should be $0 < \pi_j < 1$, except in special circumstances, so that total flux can only decrease through the simulation. For convenience, the value of $p$ is updated (within each component) during the simulation.

Simulation by weight adjustment is performed whenever possible. This includes

- Transmission through filters and windows.

- Reflection from monochromator (and analyser) crystals with finite reflectivity and mosaicity.
- Reflections from mirrors.
- Passage of a continuous beam through a chopper.
- Scattering from all types of samples.

### 2.2.1. Statistical errors of non-integer counts

In a typical simulation, the result will consist of a count of x-ray histories ("rays") with different weights. The sum of these weights is an estimate of the mean number of x-rays hitting the monitor (or detector) per second in a "real" experiment. One may write the counting result as

$$I = \sum_i p_i = N\overline{p}, \tag{2.3}$$

where $N$ is the number of rays hitting the detector and the horizontal bar denotes averaging. By performing the weight transformations, the (statistical) mean value of $I$ is unchanged. However, $N$ will in general be enhanced, and this will improve the accuracy of the simulation.

To give an estimate of the statistical error, we proceed as follows: Let us first for simplicity assume that all the counted x-ray weights are almost equal, $p_i \approx \overline{p}$, and that we observe a large number of x-rays, $N \geq 10$. Then $N$ almost follows a normal distribution with the uncertainty $\sigma(N) = \sqrt{N}$ [1]. Hence, the statistical uncertainty of the observed intensity becomes

$$\sigma(I) = \sqrt{N}\overline{p} = I/\sqrt{N}, \tag{2.4}$$

as is used in real x-ray experiments (where $\overline{p} \equiv 1$). For a better approximation we return to Eq. (2.3). Allowing variations in both $N$ and $\overline{p}$, we calculate the variance of the resulting intensity, assuming that the two variables are independent:

$$\sigma^2(I) = \sigma^2(N)\overline{p}^2 + N^2\sigma^2(\overline{p}). \tag{2.5}$$

Assuming as before that $N$ follows a normal distribution, we reach $\sigma^2(N)\overline{p}^2 = N\overline{p}^2$. Further, assuming that the individual weights, $p_i$, follow a Gaussian distribution (which in some cases is far from the truth) we have $N^2\sigma^2(\overline{p}) = \sigma^2(\sum_i p_i) = N\sigma^2(p_i)$ and reach

$$\sigma^2(I) = N\left(\overline{p}^2 + \sigma^2(p_i)\right). \tag{2.6}$$

The statistical variance of the $p_i$'s is estimated by $\sigma^2(p_i) \approx (\sum_i p_i^2 - N\overline{p}^2)/(N-1)$. The resulting variance then reads

$$\sigma^2(I) = \frac{N}{N-1}\left(\sum_i p_i^2 - \overline{p}^2\right). \tag{2.7}$$

[1] This is not correct in a situation where the detector counts a large fraction of the x-rays in the simulation, but we will neglect that for now.

For almost any positive value of $N$, this is very well approximated by the simple expression

$$\sigma^2(I) \approx \sum_i p_i^2. \tag{2.8}$$

As a consistency check, we note that for all $p_i$ equal, this reduces to eq. (2.4)

In order to compute the intensities and uncertainties, the detector components in McXtrace will keep track of $N = \sum_i p_i^0, I = \sum_i p_i^1$, and $M_2 = \sum_i p_i^2$.

## 2.3. Weight factor transformations during a Monte Carlo choice

When a Monte Carlo choice must be performed, *e.g.* when the initial energy and direction of the x-ray ray is decided at the source, it is important to adjust the x-ray weight so that the combined effect of x-ray weight change and Monte Carlo probability of making this particular choice equals the actual physical properties we like to model.

Let us follow up on the simple example of transmission. The probability of transmitting the real x-ray is $P$, but we make the Monte Carlo choice of transmitting the x-ray every time: $f_{\mathrm{MC}} = 1$. This must be reflected on the choice of weight multiplier $\pi_j$ given by the master equation In the "real" semi-classical world, there is a distribution (probability density) for the x-rays in the six dimensional (energy, direction, position) space of $\Pi(E, \mathbf{\Omega}, \mathbf{r}) = dP/(dEd\mathbf{\Omega}d^3\mathbf{r})$ depending upon the source type and its parameters (such as gap, period, field strength etc. for an undulator). In the Monte Carlo simulations, the six coordinates are for efficiency reasons in general picked from another distribution: $f_{\mathrm{MC}}(E, \mathbf{\Omega}, \mathbf{r}) \neq \Pi(E, \mathbf{\Omega}, \mathbf{r})$, since one would *e.g.* often generate only x-rays within a certain parameter interval. However, we must then require that the weights are adjusted by a factor $\pi_j$ (in this case: $j = 1$) so that

$$f_{\mathrm{MC}}\pi_j = P. \tag{2.9}$$

This probability rule is general, and holds also if, e.g., it is decided to transmit only half of the rays ($f_{\mathrm{MC}} = 0.5$). An important different example is elastic scattering from a powder sample, where the Monte-Carlo choices are the particular powder line to scatter from, the scattering position within the sample and the final x-ray direction within the Debye-Scherrer cone.

### 2.3.1. Direction focusing

An important application of weight transformation is direction focusing. Assume that the sample scatters the x-rays in many directions. In general, only x-rays in some of these directions will stand any chance of being detected. These directions we call the *interesting directions*. The idea in focusing is to avoid wasting computation time on x-rays scattered in the other directions. This trick is an instance of what in Monte Carlo terminology is known as *importance sampling*.

If *e.g.* a sample scatters isotropically over the whole $4\pi$ solid angle, and all interesting directions are known to be contained within a certain solid angle interval $\Delta\mathbf{\Omega}$, only these solid angles are used for the Monte Carlo choice of scattering direction. According to Eq. (2.9), the weight factor will then have to be changed by the amount $\pi_j = |\Delta\mathbf{\Omega}|/(4\pi)$. One thus ensures that the mean simulated intensity is unchanged during a "correct" direction focusing, while a too narrow focusing will result in a lower (*i.e.* wrong) intensity, since we cut x-rays rays that should have reached the final detector.

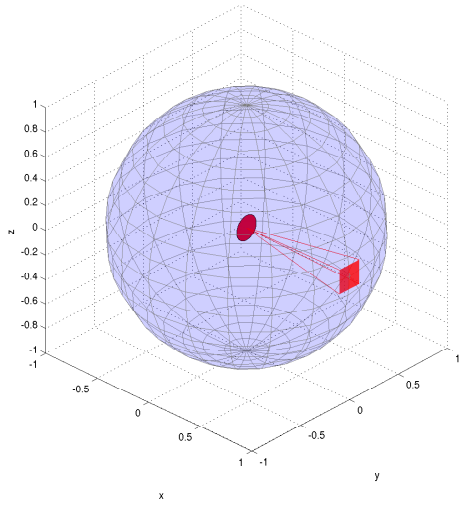


Figure 2.1.: Illustration of the effect of direction focusing in McXtrace. Weights of x-rays emitted into a certain solid angle are scaled down by the full unit sphere area.

## 2.4. Stratified sampling

One particular efficiency improvement technique is the so-called *stratified sampling*. It consists in partitioning the event distributions in representative sub-spaces, which are then all sampled individualy. The advantage is that we are then sure that each sub-space is well represented in the final integrals. This means that instead of shooting $N$ events, we define $D$ partitions and shoot $r = N/D$ events in each partition. We may define partitions so that they represent 'interesting' distributions, e.g. from events scattered on a monochromator or a sample. The sum of partitions should equal the total space integrated by the Monte Carlo method, and each partition must be sampled randomly.

In the case of McXtrace, the stratified sampling is used when repeating events, i.e. when using the SPLIT keyword in the TRACE section on beamline descriptions. We emphasize here that the number of repetitions $r$ should not exceed the dimensionality of the Monte Carlo integration space (which is $d = 10$ for x-ray events) and the dimen-

| Records | Accurarcy |
|---|---|
| $10^3$ | 10 % |
| $10^4$ | 2.5 % |
| $10^5$ | 1 % |
| $10^6$ | 0.25 % |
| $10^7$ | 0.05 % |

Table 2.1.: Accuracy estimate as a function of the number of statistical events used to estimate an integral with McXtrace.

sionality of the partition spaces, i.e. the number of random generators following the stratified sampling location in the beamline.

## 2.5. Accuracy of Monte Carlo simulations

When running a Monte Carlo, the meaningful quantities are obtained by integrating random events into a single value (e.g. flux), or onto an histogram grid. The theory [Jam80] shows that the accuracy of these estimates is a function of the space dimension $d$ and the number of events $N$. For large numbers $N$, the central limit theorem provides an estimate of the relative error as $1/\sqrt{N}$. However, the exact expression depends on the random distributions.

McXtrace uses a space with $d = 12$ parameters to describe x-rays (position, wavevector, weight, polarisation, phase, time). We show in Table 2.1 a rough estimate of the accuracy on integrals as a function of the number of records reaching the integration point. This stands both for integrated flux, as well as for histogram bins - for which the number of events per bin should be used for $N$.

# 3. Source components

McXtrace contains a number of different source components, and any simulation will usually contain exactly one of these sources. The main function of a source is to determine a set of initial parameters $(\mathbf{r}, \mathbf{k}, t)$ for each photon ray. This is done by Monte Carlo choices from suitable distributions. For example, in most present sources the initial position is found from a uniform distribution over the source surface. The initial photon wavenumber is selected within an interval of either the corresponding energy or the corresponding wavelength.

For pulsed sources, the choice of the emission time, $t$, is being made on basis of detailed analytical expressions. For other sources, $t$ is set to zero. In the case one would like to use a steady state source with time-of-flight settings, the emission time of each photon ray should be determined using a Monte Carlo choice. This may be achieved by the `EXTEND` keyword in the instrument description source as in the example below:

```
TRACE

COMPONENT MySource=Source_pt(...) AT (...)
EXTEND
%{
  t = 1e-3*randpm1(); /* set time to +/- 1 ms */
%}
```

Also take a look at the **Chopper_simple** component.

### 3.0.1. Photon flux and Brilliance

The flux of the sources deserves special attention. The total intensity is defined as the sum of weights of all emitted x-rays during one simulation (the unit of total photon weight is thus photons per second). The flux, $\psi$, at an instrument is defined as intensity per area perpendicular to the beam direction.

The source Brilliance, $\Phi$, is defined in different units (See e.g. [ANM11]): the number of photon rays emitted per second from a 1 $m^2$m area on the source surface, with direction within a 1 $m^2$rad angle window, and with wavelength within a 1% interval. The total intensity of real photons emitted towards a given diaphragm (units: ph./s) is therefore (for constant $\Phi$):

$$I_{\text{total}} = \Phi A \Delta\Omega \Delta\lambda, \tag{3.1}$$

where $A$ is the source area, $\Delta\Omega$ is the solid angle of the diaphragm as seen from the source surface, and $\Delta\lambda$ is the width of the wavelength interval in which photons are emitted (assuming a uniform wavelength spectrum).

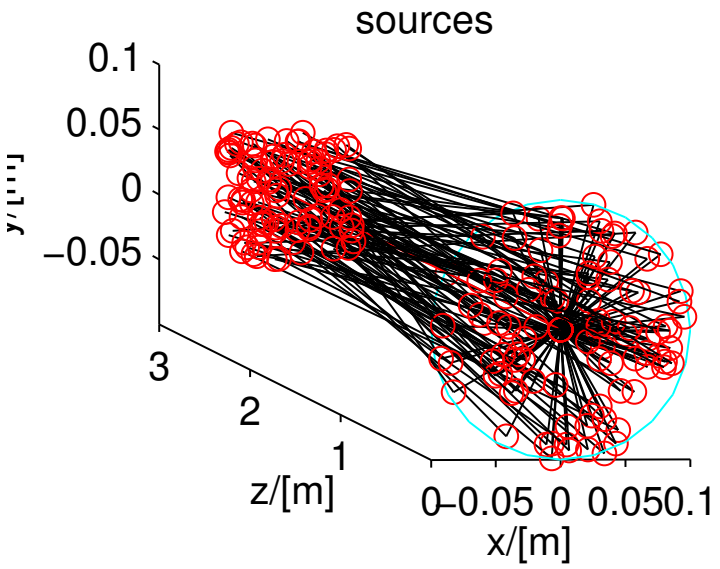


Figure 3.1.: A circular source component (at z=0) emitting photon rays randomly, either from a model, or from a data file.

The simulations are performed so that detector intensities are independent of the number of photon histories simulated (although more photon histories will give better statistics). If $N_{\text{sim}}$ denotes the number of x-ray histories to simulate, the initial photon weight $p_0$ must be set to

$$p_0 = \frac{N_{\text{total}}}{N_{\text{sim}}} = \frac{\Phi(\lambda)}{N_{\text{sim}}} A\Omega\Delta\lambda, \tag{3.2}$$

where the source brilliance is now given a $\lambda$-dependence.

As a start, we recommend new McXtrace users to use the **Source_flat** component. For a slightly more realistic sources are supply **Source_flat** with a spectrum file (for instance generated by SPECTRA [TK01]) or **Source_gaussian**.

Optimizers can dramatically improve the statistics, but may occasionally give wrong results, due to misleaded optimization. You should always check such simulations with (shorter) non-optimized ones.

Other ways to speed-up simulations are to read events from a file. See section 3.6 for details.

## 3.1. Source_pt: A mathematical point emitting photons with a spectrum either uniform, gaussian or generated from a datafile

**Input Parameters for component `Source_pt` from sources**

```
1 <Parameter = value>, [Unit], Description
```

The simplest source model, where a mathematical point source at $(0, 0, 0)$ emits photons. The wavevector of the emitted photons is picked randomly in a defining aperture *focus_xw* by *focus_yh* m at $(0, 0, dist)$. Please note that this aperture is merely a virtual aperture used to reduce the sampling space. This has a few implications: Other components may be placed without reference to the aperture, but if the aperture does not fill the full acceptance window of subsequent components your simulations will be biased. The aperture is simply there to provide efficient sampling.

If a *spectrum_file* is not supplied, the xray is given a weight which is the total wavelength-integrated intensity downscaled by the solid angle subtended by the defining aperture. The Energy/wavelength spectrum of the emitted photons is centered around *E0* or $\lambda 0$ with a width of *dE* or $d\lambda$ respectively. *E0* takes precedence over $\lambda 0$. Thus if $E0 \neq 0$ the the combination *E0, dE* is used, $\lambda 0, d\lambda$ otherwise. If *gauss* (the default) is set to be non-zero the spectrum is Gaussian with mean *E0* ($\lambda 0$) and standard deviation *dE* ($d\lambda$), otherwise the spectrum is uniform in $E0 \pm dE$ ($\lambda 0 \pm d\lambda$).

If a *spectrum_file is* supplied, a slightly different strategy is adopted. In this case the wavelength/energy range implied by the datafile is sampled unformly and each ray is assigned a weight corresponding to the intensity indicated by linear interpolation between datapoints at that wavelength. This implies an oversampling of weak parts of the intensity spectrum.

Currently only completely coherent or fully incoherent beams are supported. If *randomphase* is specified emitted photons will be assigned a random phase, otherwise it is set to the value of *phase*.

## 3.2. Source_flat: A flat surface emitting photons with a spectrum either uniform, gaussian or generated from a datafile

**Input Parameters for component `Source_flat` from sources**

```
1 <Parameter = value>, [Unit], Description
```

A simple source model, with a flat surface emitting photons. The surface in the $xy$-plane is specified as a rectangle with dimensions *xwidth* $\times$ *yheight* $m^2$, or as a circle with *radius* m. The initial x-ray position is chosen randomly in the source surface — its wavevector is chosen randomly (exactly as in the case of **Source_pt**) (section 3.1) in the defining aperture with height *focus_yh* and width *focus_xw* placed at $(0, 0, dist)$.

Just as for **Source_pt** the aperture is for efficiency purposes and, if misused, may cause biasing.

With the exception of source size related parameters, all other parameters are identical to **Source_pt** Note that this also applies to coherence and photon phase. If *random-phase*= 0 then a photon emitted from $(x_0, y_0, 0)$ will in phase with a photon emitted from $(x_1, y_1, 0)$. I.e. full transversal coherence.

## 3.3. Source_div: A continuous source with specified divergence

**Input Parameters for component `Source_div` from sources**

```
1 <Parameter = value>, [Unit], Description
```

**Source_div** is a rectangular source, $w \times h$, which emits a beam of a specified divergence around the direction of the $z$-axis.

Just as for **Source_pt**, if a *spectrum_file* is not supplied, the xray is given a weight which is the total wavelength-integrated intensity downscaled by the solid angle subtended by the defining aperture and the energy/wavlength spectrum is centered around *E0* or $\lambda 0$ with width *dE* or $d\lambda$ respectively. The profile is Gaussian if *gauss*$\neq$ 0, uniform if *gauss*= 0, If a *spectrum_file is* supplied, a slightly different strategy is adopted. In this case the wavelength/energy range implied by the datafile is sampled unformly and each ray is assigned a weight corresponding to the intensity indicated by linear interpolation between datapoints at that wavelength. This implies an oversampling of weak parts of the intensity spectrum.

The beam intensity is uniform over the whole of the source and the source divergences are *focus_ah* and *focus_aw* in degrees. Unless *gauss_a*$\neq$= 0 the individual photons are sampled from a uniform divergence distribution, otherwise a Gaussian is used.

## 3.4. Source_gaussian: the model has a gaussian distribution of intensity

**Input Parameters for component `Source_gaussian` from sources**

```
1 <Parameter = value>, [Unit], Description
```

A simplified version of a completely incoherent source of horizontal and vertical sizes *sig_x* and *sig_y* respectively with angular divergence *sigPr_x* and *sigPr_y*. Can be used to model an undulator source emitting a photon beam that has gaussian distribution. This naturally generates a larger Gaussian profile at a distance. A sampling window at $(0, 0, dist)$ may be specified by the parameters *focus_xw* and *focus_yh*, which restricts the sampling phases space of the emitted photons (See section 3.1 for details).

The energy spectrum emitted by **Source_gaussian** source is completely analogous to **Source_pt**, as is its photon phase functionality.

## 3.5. Source_lab: X-ray tube laboratory source

**Input Parameters for component `Source_lab` from sources**

```
1 <Parameter = value>, [Unit], Description
```

**Source_lab** is a model of a laboratory X-ray tube. An electron ray hits a target of specified material. Currently, only single material targets are allowed. To model multiple material targets one could construct a model with two or more sources simultaneously. This has consequences for intensity of the source which should be downscaled accordingly.

An electron beam of rectangular transverse crossection (*width,height*) and energy *E0* impinges on the target of material. Wrt. the electron beam, the target is considereded infinitely thick. The beam is considered to have uniform intenisty. Thus, the spatial distribution of x-ray generation will be exponential in the depth of the material.

Further, an exit aperture is defined with dimensions (*focus_xw,focus_yh*). The centre of the aperture is situated at a distance *dist* m from where the electron beam hits the target slab at an elevation of *take_off* (see Figure 3.5).

The **Source_lab** coordinate system has its origin in the center of the elctron beam at the surface of the anode material and is oriented such that the z-axis points at the center of the exit window, and the x-axis is parallel to the *width* of the electron beam.

Note that the exit aperture is merely an opening. If the material absorption of a window, e.g. Be, is to be taken into account a **Filter** (section 4.5) should be inserted after the exit aperture.

For each photon ray to be generated, a Monte Carlo choice is made to generate either a Bremsstrahlung photon or one from one of the x-ray emission lines of the material. $(1 - frac)$ of the rays are generated from characteristic emission, and $(frac)$ from Bremsstrahlung. In most cases Bremsstrahlung is unwanted background, which is why the default is 0.1. Note that this *only* governs how much of the available statistics is diverted into simulating Bremsstrahlung. It does not have an impact on what intensity is detected in subsequent monitors — only on the errorbars of the detected numbers.

The spectral characteristics of the generated Bremsstrahlung is goverened by the model suggested by Kramer [Kra23]. Although disputed in several subsequent papers, the model is simple, and sufficiently accurate for many background estimation purposes.

Characteristic emission on the other hand is sampled from a set of Lorentzian functions with central wavelengths found in the work by [Bea67] with spectral widths taken from [KO79].

An example of beam spectral characteristics emitted from a Cu-anode target detected 1 mm from an exit aperture of $1 \times 1$ cm$^2$ 10 cm downstream from the target at a *take_off* angle of 6° is seen in figure 3.5.

**Source_lab** includes a set of common anode materials: {Cu, Ga, Mo, Ag, W}. More materials can be added by the following procedure:

1. Find the atomic number, $Z$, for the material you want to add. So far only single materials are supported.

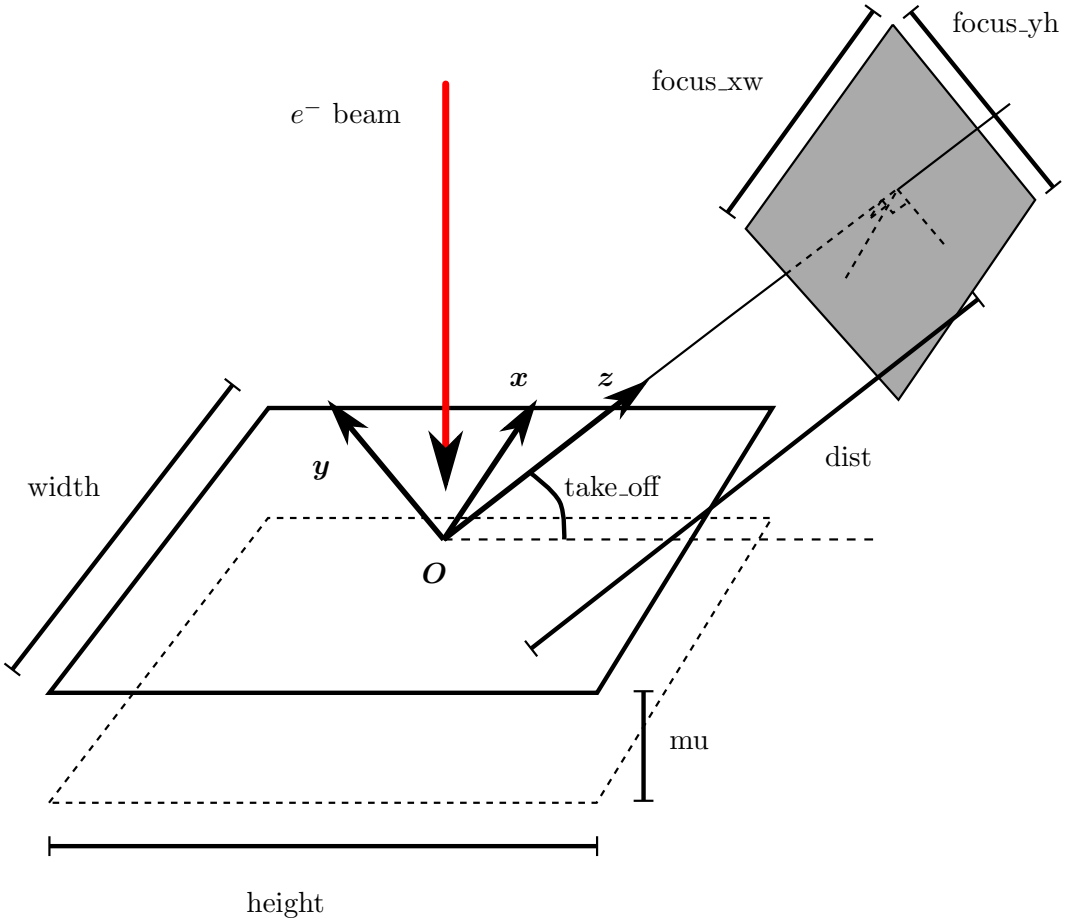


Figure 3.2.: Geometry of the **Source_lab** component. An electron beam impinges at a right angle on an anode material, where X-rays are generated. The Origin is defined to be a the centre of the electron beam on the anode surface, and the coordinate system is oriented such that the $z$-axis point towards the exit aperture.

Figure 3.3.: Intensity vs. wavelenghth for a Cu-anode laboratory source.

2. Look up (and note down) the central energies of (up to) 6 characteristic lines for the material in e.g. [Bea67]. Also note down the number lines you have recorded.

3. Look up the natural spectral widths of those lines in [KO79].

4. Find the relative intensity of the the set of lines. For instance from the x-ray data booklet.

5. Note down the ionization energy $E_I(Z)$, and flourescence yield, $Y(Z)$, of the anode material.

With this information assemble a code line as:
$\{Z, E_I(Z), Y(Z), n, \{[E_C]\}, \{[W]\}, \{[I]\}\},$
and put it in the source file `Source_lab.comp`, just above the line that reads
`{0,0.0,0.0,0,{0,0,0,0,0,0},{0,0,0,0,0,0},{0,0,0,0,0,0}}`
in the `SHARE` section of the component source code. Here $[E_C]$ refers to a comma separated list of characteristic energies in keV, $[W]$ a list of characteristic widths in keV, and $[I]$ a list of relative line intensities. Lastly, $n$ denotes the number of x-ray lines.

## 3.6. Other sources components: virtual sources (event files)

# 4. Beam optical components: Arms, slits, filters etc.

This chapter contains a number of optical components used to modify the x-ray beam in various ways, as well as the "generic" component **Arm**.

## 4.1. Arm: The generic component

**Input Parameters for component `Arm` from optics**

```
1 <Parameter = value>, [Unit], Description
```

The component **Arm** is empty; is resembles an optical bench and has no effect on the x-ray. The purpose of this component is only to provide a standard means of defining a local coordinate system within the instrument definition. Other components may then be positioned relative to the **Arm** component using the McXtrace meta-language. The use of **Arm** components in beamline definitions is not required but is recommended for clarity. **Arm** has no input parameters.

The first Arm instance in an instrument definition may be changed into a **Progress_bar** (sec. 9.1) component in order to display simulation progress on the fly, and possibly save intermediate results.

## 4.2. Slit: A beam defining diaphragm

**Input Parameters for component `Slit` from optics**

```
1 <Parameter = value>, [Unit], Description
```

The component **Slit** is a very simple construction. It sets up an opening at $z = 0$, and propagates the photon rays onto this plane (by the kernel call PROP_Z0). Photons within the slit opening are unaffected, while all other photons are discarded by the kernel call ABSORB.

By using **Slit**, some photons contributing to the background in a real experiment will be neglected. These are, for instance, the ones that scatter off the inner side of the slit, penetrate the slit material, or clear the outer edges of the slit.

The opening of the slit is determined by specifying either a *radius*, a width *(xwidth)* and a height *(yheight)*, or absolute limits in $x$ and $y$ *(xmin, xmax, ymin, ymax)*, in order of precedence. If *radius* is set, the opening is considered circular.

The slit component can also be used to discard insignificant (*i.e.* very low weight) rays, that in some simulations may be very abundant and therefore time consuming. If the optional parameter *cut* is set, all x-rays with $p < cut$ are ABSORB'ed. This use is recommended in connection with **Virtual_output**.

**Slit** may be used to model slit diffraction, thourgh judicious use of the resmapling parameters: *focus_xw,focus_yh, focus_x0, focus_y0*. The similarity in parameter naming for sources (chapter 3) is intentional as computationally the processes are the same. If present the focus parameters cause the ray to be regarded as a Huygens wave and a new direction going towards the resampling window is computed. The resulting difference in phase gives rise to maxima and minima as expected. This feature should be used with care: the further away from the optical axis, the more statistics and finer binning is needed to get meaningful results. Please refer to [BK+13] for more deatils.

## 4.3. Slit_N: multiple slits

**Input Parameters for component `Slit_N` from optics**

```
1 <Parameter = value>, [Unit], Description
```

Documentation pending.

## 4.4. Beamstop: A photon absorbing area

**Input Parameters for component `Beamstop` from optics**

```
1 <Parameter = value>, [Unit], Description
```

The component **Beamstop** can be seen as the reverse of the **Slit** component. It sets up an area at the $z = 0$ plane. Photons that hit the plane within this area are ABSORB'ed, while all others are unaffected.

By using this component, some photons contributing to the background in a real experiment will be neglected. These are the ones that scatter off the side of the (real) beamstop, or penetrate the absorbing material. Further, the holder of the beamstop is not simulated.

**Beamstop** can be either circular or rectangular. The input parameters of **Beamstop** are either height and width *(xwidth, yheight)* or the four coordinates,*(xmin, xmax, ymin, ymax)* defining the opening of a rectangle, or the *radius* of a circle, depending on which parameters are specified.

If the "direct beam" (e.g. after a monochromator or sample) should not be simulated, it is possible to emulate an ideal beamstop so that only the scattered beam is left; without the use of **Beamstop**: This method is useful for instance in the case where only photons scattered from a sample are of interest. The example below removes the direct beam and any background signal from other parts of the beamline

```
COMPONENT MySample=PowderN(...) AT (...)
EXTEND
%{
  if (!SCATTERED) ABSORB;
%}
```

## 4.5. Filter: A general absorption filter model

**Input Parameters for component `Filter` from optics**

```
1 <Parameter = value>, [Unit], Description
```

This component is a filter in the shape of a rectangular block or a general shape defined by a set of polygons. Given an input file containing material parameters. Neccessary parameters are nominal density and a parameterization of the linear attenuation coefficient, $\mu$ as a function of wavelength (or energy).

The model is very simple: Firstly the X-ray is traced to find intersection points between ray and filter (0 or 2). If no intersection is found the x-ray is left untouched and nothing further happens. Assuming the ray intersects the filter: Secondly, the path length d$l$ within the filter is computed. Thirdly a $\mu = f(\lambda, \text{material})$ is computed by interpolating in a datafile, and the x-ray weight is adjusted according to $p = p \exp(-\mathrm{d}l * \mu)$. The x-ray is left at the point where it exits the filter block (the 2nd intersection).

Example data files corresponding to all elements up to $Z = 92$ are distributed with McXtrace in the `MCXTRACE/data` directory as `*.txt` files. These tables have been extracted from the NIST FFAST [Nis] x-ray database. To generate other datafiles from the same source a simple shell script: `MCXTRACE/data/get_xray_db_data` is also distributed with McXtrace Running this script will connect to the NIST webiste and download a `.html` file. This output must now be modified such that `html`-tags are removed and all header lines begin with #

### 4.5.1. Example

This is an example of how to download and generate datafiles for the `Filter.comp` and others.

The distributed tables have been extracted from the NIST x-ray database. To ease generation of more dtafiles from the same source a simple shell script:
`MCXTRACE/data/get_xray_db_data`
is also distributed with McXtrace

Running this script will connect to the NIST webiste and download a `.html` file. This output must now be modified wuch that `html`-tags are removed and all header lines begin with #.

```
/usr/local/lib/mcxtrace/data/get_xray_db_data 3 output.dat
```

where the second parameter (3) is the atom number of the material, for which we want to generate a datafile. Now open the generated datafile (`output.dat`) with your favourite text editor and make sure the file ends up looking like this

```
#Li (Z 3)
#Atomic weight: A[r]  6.941000
#Nominal density: rho 5.3300E-01
#    rho[a](barns/atom) = [mu/rho](cm^2 g^-1)  x  1.15258E+01
#    E(eV) [mu/rho](cm^2 g^-1) = f[2](e atom^-1)  x  6.06257E+06
#    2 edges. Edge energies (keV):
#
#
#    K      5.47500E-02  L I    5.34000E-03
#
#Relativistic correction estimate f[rel] (H82,3/5CL) = -9.8613E-04,
#    -6.0000E-04 e atom^-1
#    Nuclear Thomson correction f[NT] = -7.1131E-04 e atom^-1
#
#-------------------------------------------------------------------------------
#Form Factors, Attenuation and Scattering Cross-sections
#Z=3, E = 0.001 - 433 keV
#
#    E        f[1]         f[2]        [mu/rho]    [sigma/rho]  [mu/rho]   [mu/rho][K] lambda
#                                     Photoelectric Coh+inc      Total
#   keV        e atom^-1    e atom^-1   cm^2 g^-1   cm^2 g^-1   cm^2 g^-1   cm^2 g^-1  nm
5.233200E-03  9.08733E-01  0.0000E+00  0.0000E+00  2.3914E-07  2.3914E-07  0.000E+00  2.369E+02
5.313300E-03  8.59283E-01  0.0000E+00  0.0000E+00  2.5404E-07  2.5404E-07  0.000E+00  2.333E+02
5.334660E-03  8.03599E-01  0.0000E+00  0.0000E+00  2.5813E-07  2.5813E-07  0.000E+00  2.324E+02
5.366700E-03  8.56971E-01  1.0769E-01  1.2165E+05  2.6435E-07  1.2165E+05  0.000E+00  2.310E+02
.
.
.
3.788588E+02  3.00000E+00  3.9121E-08  6.2602E-07  8.4389E-02  8.4390E-02  6.123E-07  3.273E-03
4.050001E+02  3.00000E+00  3.3438E-08  5.0054E-07  8.2127E-02  8.2128E-02  4.895E-07  3.061E-03
4.329451E+02  3.00000E+00  2.8581E-08  4.0022E-07  7.9892E-02  7.9892E-02  3.913E-07  2.864E-03
```

Please make sure you don't forget to remove the html-tags in the bottom of the file as well. In the future we will set up a more streamlined way of doing this.

## 4.6. Chopper_simple: An ideal chopper

**Input Parameters for component `Chopper_simple` from optics**

```
1 <Parameter = value>, [Unit], Description
```

`Chopper_simple` is an idelized version of a chopper with a rectangular chopper opening which may open instantly and has no side-scattering etc.

It models the chopper with a blocking infinitely thin aperture which becomes transparent in the time interval $t \in [t0, t0 + \tau]$. This is an idelized version of a chopper where the chopper opening is rectangular which may open instantly (if *t_rise*= 0, the default). For nonzero rise time the aperture simply becomes gradually less opaque for $t \in [t0 - t\_rise, t0]$.

For correct normalization of intesity a chopper period, $T$, must also be set.

*is_first* is useful when using **Chopper_simple** with continous sources who inherently have no time-dependence. Thus the emission time of the photon ray is arbitrary, and the chopper defines the temporal signature of the beam, i.e. it simply sets the time-parameter of the photon ray randomly in the opening window of the chopper. Naturally this should only be used for the *first* chopper element in a simulation.

# 5. Refractive optical components: lenses

An X-ray refractive lens, often referred to as a Compund Refractive Lens (CRL), is a fairly new type of device, which has gained popularity in the last few years. An early study showing the feasiblity of such devices may be found in [SKS96]. As the refractive index of X-rays is $n \approx 1$ a number of lenses, stacked together is usually necessary to bring the focal length to practical values. McXtrace includes a few lens components, which all have slightly different charactertics.

## 5.1. Lens_simple: Thin lens approximation

**Input Parameters for component `Lens_simple` from optics**

```
1 <Parameter = value>, [Unit], Description
```

This models a thin-lens approximation of a stack of parabolic refractive x-ray lenses

## 5.2. Lens_parab: Thick parabolic CRL

**Input Parameters for component `Lens_parab` from optics**

```
1 <Parameter = value>, [Unit], Description
```

Component model of a stack of compund refractive lenses. Each lens in the stack is modelled by two $2D$ parabolic surfaces, and rays are traced through all the complete stack taking the displacement of the surfaces into account. This is naturally less efficient than a thin lens approximation.

The logic behind the model is the following: a photon interacts with the surface of the lens at a certain angle $\Theta$, which alters in accordance with Snell's law upon photon's entering the lens's material. The combination of the refractive process inside material (that is characterized by a material datafile) and the interaction with a geometrical surface results in a photon's new trajectory, i.e. in focusing.

The functionality of `Lens_parab_Cyl`, `Lens_parab_rough`, and `Lens_parab_Cyl_rough` will be merged into this component.

## 5.3. Lens_parab_Cyl: Thick 1D-parabolic CRL

**Input Parameters for component `Lens_parab_Cyl` from optics**

```
<Parameter = value>, [Unit], Description
```

This component is based on the same logical approach as the **Lens_parab** with one significant difference - geometrical surface. Parabolic cylinder (parabolic curvature along the vertical axes only, invariant along the horizontal) and thus ofocuses the beam in only in the vertical direction.

This component and its functionality is scheduled to be merged into **Lens_parab**

## 5.4. Lens_parab_rough: Thick parabolic CRL including roughness-model

**Input Parameters for component `Lens_parab_rough` from optics**

```
<Parameter = value>, [Unit], Description
```

Identical to **Lens_parab** except it has the option of a *roughness* parameter. Roughness is simply modelled by a stochatstic, normally distributed, displacement of the normal vector of the lens surfaces.

This component and its functionality is scheduled to be merged into **Lens_parab**

## 5.5. Lens_parab_Cyl_rough: Thick 1D-parabolic CRL including roughness-model

**Input Parameters for component `Lens_parab_Cyl_rough` from optics**

```
<Parameter = value>, [Unit], Description
```

Identical to **Lens_parab_Cyl** except it has the option of a *roughness* parameter. Roughness is simply modelled by a stochatstic, normally distributed, displacement of the normal vector of the lens surfaces.

This component and its functionality is scheduled to be merged into `Lens_parab`

## 5.6. Lens_Kinoform: refractice kinoform lens

**Input Parameters for component `Lens_Kinoform` from optics**

```
<Parameter = value>, [Unit], Description
```

Doc. Pend.

## 5.7. Lens_elliptical:

**Input Parameters for component `Lens_elliptical` from optics**

```
1 <Parameter = value>, [Unit], Description
```

doc. pend.

# 6. Reflective optical components: mirrors

This section describes advanced reflective X-ray optics components such as mirrors.

## 6.1. Mirror_curved: Cylindrically curved mirror

**Input Parameters for component `Mirror_curved` from optics**

```
1 <Parameter = value>, [Unit], Description
```

Models a cylindrical mirror, positioned in the XZ-plane curving towards positive X. The input parameter *radius* defines the radius of curvature and the mirror size is given by *length* and *width* where length and width is along Z and Y respectively. *coating* and *R0* are mutually exclusive. If *R0* is nonzero, it is taken as the reflectivity value, irrespective of wavelength, whereas coating nominates a file from which to read values for $f_1$ and $f_2$. See [Nis] for definitions. For elements $Z \in [1, 92]$ datafiles are distributed with the McXtrace system that may be used as: `coating="Rh.txt"`. or `coating="Al.txt"`.

This component is scheduled to be merged with `Mirror_parabolic` and `Mirror_elliptic`

## 6.2. Mirror_parabolic: Mirror with a parabolic curvature profile.

**Input Parameters for component `Mirror_parabolic` from optics**

```
1 <Parameter = value>, [Unit], Description
```

This component is scheduled to be merged with `Mirror_elliptic`

## 6.3. Mirror_elliptic: Mirror with a elliptic curvature profile.

**Input Parameters for component `Mirror_elliptic` from optics**

```
1 <Parameter = value>, [Unit], Description
```

This component is scheduled to be merged with `Mirror_parabolic`

## 6.4. Multilayer_elliptic: Elliptically curved mirror coated with a multilayer

**Input Parameters for component `Multilayer_elliptic` from optics**

```
1 <Parameter = value>, [Unit], Description
```

The component **Multilayer_elliptic** models a single rectangular reflecting multilayer mirror plate with elliptical curvature. It can be used as a sample component, to *e.g.* assemble a Kirkpatrick-Baez focusing system or in combination with a double-crystal monochromator.

Figure 6.1*Left* shows a side view of a mirror (the blue section of the ellipse) in the McXtrace coordinate system. At the mirror center, the mirror tangent is parallel to the $z$ axis and the mirror normal is parallel to the $y$ axis. The width of the mirror is $w$ and in $y-z$ plane the mirror has the curvature of an ellipse with major axis $a$ and minor axis $b$,

$$\frac{z^2}{a^2}+\frac{y^2}{b^2}=1\,,\ |x|<\frac{w}{2}\,. \tag{6.1}$$

The length of the mirror is $L$. The coordinates of the mirror center $(0, Y_0, Z_0)$ and the ellipse parameters $a$, $b$ are determined uniquely by the central glancing angle, the source-mirror distance and the mirror-image distance. The position of the mirror is chosen to be at the positive side of the $y$ axis.

The input parameters of this component are: *theta* [°], the incident angle; *s1* [m], the distance from the source to the multilayer; *s2* [m], the focusing distance of the multilayer; *length* [m], the length of the mirrors; *width* [m], the width of the mirror along the $x$-axis; $R$, the reflectivity.

### 6.4.1. Definition of the reference frames

The direction and position of the incoming photon is defined relative to the coordinate system illustrated in Fig. 6.1*Left* (in the code referred to as *McXtrace coordinate system*):

- the y-axis is parallel to the central mirror normal
- the z-axis is parallel to the central mirror tangent
- the origin is at the mirror center

However, all the calculations are conducted in another reference frame which is illustrated in Fig. 6.1 *Right*(in the following referred to as the *Ellipse coordinate system*):

- the z-axis is parallel to major axis of ellipse
- the y-axis is parallel to minor axis of ellipse
- the origin is at the center of the ellipse
- the mirror center at $(0, Y_0, Z_0)$, uniquely determined by the glancing angle at the mirror center, the source-mirror distance and mirror-image distance.

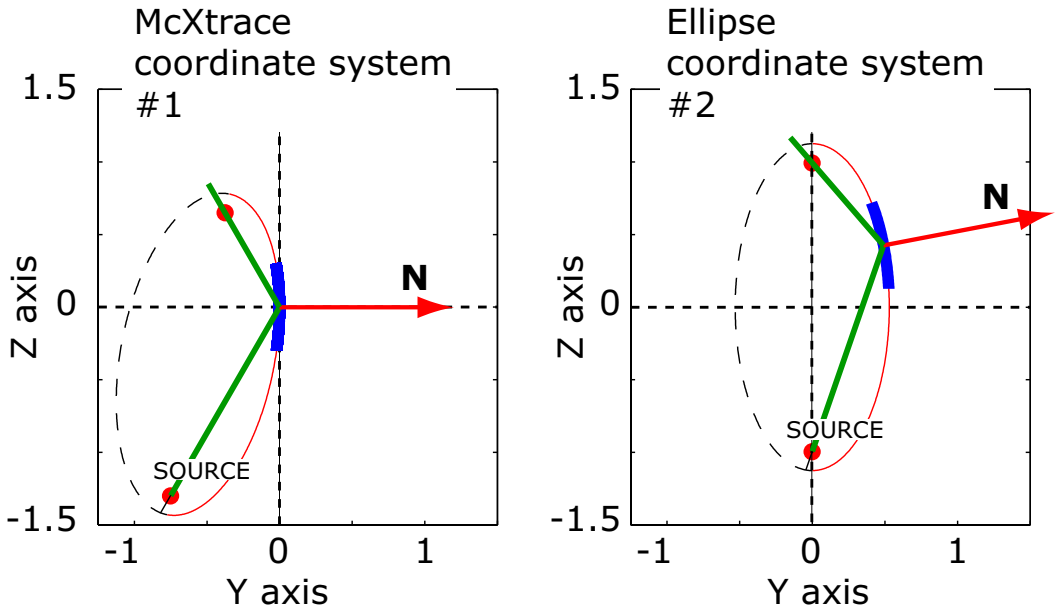


Figure 6.1.: The same image in different coordinate systems.
*Left*: *McXtrace System* with the y-axis is parallel to the central mirror normal, the z-axis is parallel to the central mirror tangent and the origin is at the mirror center.
*Right*: *Ellipse System* with the z-axis parallel to major axis of ellipse, the y-axis is parallel to minor axis of ellipse and the origin is at the center of the ellipse.

### 6.4.2. Algorithm

1. The photon is generated with a starting point $\mathbf{S}$ and a direction $\mathbf{V}_{\text{in}}$ defined in the *McXtrace* coordinate system.

2. All calculations are performed in the *Ellipse* coordinate system, so to proceed the basis is changed to that reference frame.

3. The 2 intersections of the ray with the ellipse are determined.

4. It is checked if any of the intersections are within the area defined by the mirror.

5. If one of the solutions is valid, the reflection of that ray is determined.

6. The coordinates of the starting point and direction of the reflected ray are calculated using the basis of the *McXtrace* coordinate system.

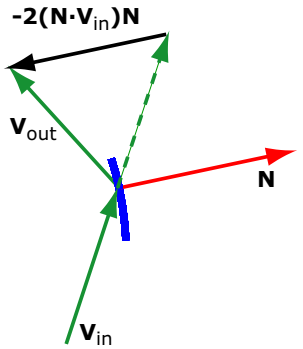


Figure 6.2.: The reflection of the unit vector $\mathbf{V}_{\rm in}$ in the mirror with the normal unit vector $\boldsymbol{N}$ is $\boldsymbol{V}_{\rm out} = \boldsymbol{V}_{\rm in} - 2(\boldsymbol{N} \cdot \boldsymbol{V}_{\rm in})\boldsymbol{N}$

## 6.5. Reflection of the ray in the mirror

The tangent and normal to the ellipse $z^2/a^2 + y^2/b^2 = 1$ at the point $(Y, Z)$ are found by implicit differentiation:

$$\frac{2z}{a^2} + \frac{2y}{b^2}\frac{dy}{dz} = 0\,, \tag{6.2}$$

so at the point $(Y, Z)$ the slope of the tangent is $\frac{dy}{dz} = -\frac{Z\,b^2}{Y\,a^2}$. The slope of the normal is minus the inverse of the tangent slope, so the coordinates of the mirror normal are

$$N_x = 0 \quad N_y = \frac{a^2\,Y}{b^2\,Z} \quad N_z = 1\,. \tag{6.3}$$

With $\mathbf{V}_{\rm in}$ and $\mathbf{N}$ denoting unit vectors (direction and normal respectively), the direction of the reflected ray is calculated as

$$\boldsymbol{V}_{\rm out} = \boldsymbol{V}_{\rm in} - 2(\boldsymbol{N} \cdot \boldsymbol{V}_{\rm in})\boldsymbol{N} = \begin{pmatrix} V_{{\rm in}x} - 2(\boldsymbol{N} \cdot \boldsymbol{V}_{\rm in})N_x \\ V_{{\rm in}y} - 2(\boldsymbol{N} \cdot \boldsymbol{V}_{\rm in})N_y \\ V_{{\rm in}z} - 2(\boldsymbol{N} \cdot \boldsymbol{V}_{\rm in})N_z \end{pmatrix} \tag{6.4}$$

### 6.5.1. Mirror reflectivity

At present, the Multilayer_elliptic Mirror component uses a reflectivity table *reflect*, which 1st column is q [$\mathring{A}^{-1}$] and from the 2nd column on as the reflectivity $R$ in [0-1] as function of tabulated energy [keV]. An example file, calculated for a particular $Si/W$ multilayer, is provided (`reflectivity.txt`). User provided reflectivity data files can be parsed by the component.

## 6.6. TwinKB_ML: Side-by-side Kirkpatrick-Baez mirror pair

**Input Parameters for component `TwinKB_ML` from optics**

```
<Parameter = value>, [Unit], Description
```

Models a pair of perpendicular, elliptically curved mirrors, known as a Montel-mirror or Side-by-side Kirkpatrick-Baez mirror.

# 7. Samples

This class of components models the sample of the experiment. This is by far the most challenging part of an xray scattering instrument to model. However, for purpose of simulating instrument performance, details of the samples are rather unimportant, allowing for simple approximations. On the contrary, for full virtual experiments it is of importance to have realistic and detailed sample descriptions. McXtrace contains both simple and detailed samples.

An important component class is elastic Bragg scattering from an ideal powder. The component **PowderN** models a powder scatterer with reflections given in an input file. The component includes absorption, incoherent scattering, direct beam transmission and can assume *concentric* shape, i.e. can be used for modelling sample enviroments.

Next type is Bragg scattering from single crystals. Two types of single crystal exist in McXtrace at present: The **Perfect_crystal** and **Single_crystal** components. **Perfect_crystal** is in fact most often used as a monchromator crystal. It models a crystal where peak broadening is dominated by the Darwin width. Currently it only handles a single defined reflection. If more than one is wanted this could be accomplished by using two instances in a `GROUP` and dynamically choose between them with a `WHEN`-statement. For details on the `GROUP` and `WHEN` constructs see the main McXtrace user manual [Knu+14]

Much more general, the component **Single_crystal** is a single crystal sample (with multiple scattering) that allows the input of an arbitrary unit cell and a list of structure factors, read from a LAZY / Crystallographica file. This component also allows anisotropic mosaicity and $\Delta d/d$ lattice space variation.

Isotropic small-angle scattering is simulated in **Saxs_Spheres**, which models scattering from a collection of hard spheres (dilute colloids). Furthermore, a whole series of sample components modelling various SAXS standard sample types are available in the `contrib` component library section.

### 7.0.1. Scattering notation

In sample components, we use a notation common for scattering experiments, where the wave vector transfer is denoted the *scattering vector*

$$\boldsymbol{q} \equiv \boldsymbol{k}_\mathrm{i} - \boldsymbol{k}_\mathrm{f}. \tag{7.1}$$

In analygo, the *energy transfer* is given by

$$\hbar\omega \equiv E_\mathrm{i} - E_\mathrm{f} = \frac{\hbar^2}{2m_\mathrm{n}}\left(k_\mathrm{i}^2 - k_\mathrm{f}^2\right). \tag{7.2}$$

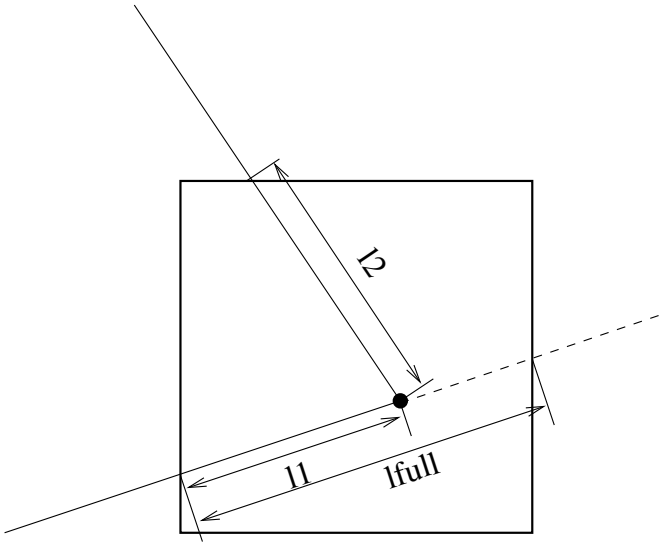


Figure 7.1.: The geometry of a scattering event within a powder sample.

### 7.0.2. Weight transformation in samples; focusing

Within many samples, the incident beam is attenuated by scattering and absorption, so that the illumination varies considerably throughout the sample. For single crystals, this phenomenon is known as *secondary extinction* [Bac75], but the effect is important for all samples. In analytical treatments, attenuation is difficult to deal with, and is thus often ignored, making a *thin sample approximation*. In Monte Carlo simulations, the beam attenuation is easily taken care of, as will be shown below. In the description, we ignore multiple scattering, which is however implemented in some sample components.

The sample has an absorption cross section per unit cell of $\sigma_c^a$ and a scattering cross section per unit cell of $\sigma_c^s$. The x-ray path length in the sample before the scattering event is denoted by $l_1$, and the path length within the sample after the scattering is denoted by $l_2$, see figure 7.1. We then define the inverse penetration lengths as $\mu^s = \sigma_c^s/V_c$ and $\mu^a = \sigma_c^a/V_c$, where $V_c$ is the volume of a unit cell. Physically, the attenuation along this path follows

$$f_{\text{att}}(l) = \exp(-l(\mu^s + \mu^a)), \tag{7.3}$$

where the normalization $f_{\text{att}}(0) = 1$.

The probability for a given x-ray to be scattered from within the interval $[l_1; l_1 + dl]$ will be

$$P(l_1)dl = \mu^s f_{\text{att}}(l_1)dl, \tag{7.4}$$

while the probability for a x-ray to be scattered from within this interval into the solid angle $\Omega$ *and* not being scattered further or absorbed on the way out of the sample is

$$P(l_1, \Omega)dld\Omega = \mu^s f_{\text{att}}(l_1) f_{\text{att}}(l_2)\gamma(\Omega)d\Omega dl, \tag{7.5}$$

where $\gamma(\Omega)$ is the directional distribution of the scattered x-rays, and $l_2$ is determined by Monte Carlo chocies of $l_1$, $\Omega$, and from the sample geometry, see e.g. figure 7.1.

In our Monte-Carlo simulations, we may choose the scattering parameters by making a Monte-Carlo choice of $l_1$ and $\Omega$ from a distribution different from $P(l_1, \Omega)$. By doing this, we must adjust $\pi_i$ according to the probability transformation rule (2.9). If we *e.g.* choose the scattering depth, $l_1$, from a flat distribution in $[0; l_{\text{full}}]$, and choose the directional dependence from $g(\Omega)$, we have a Monte Carlo probability

$$f(l_1, \Omega) = g(\Omega)/l_{\text{full}}, \tag{7.6}$$

$l_{\text{full}}$ is here the path length through the sample as taken by a non-scattered x-ray (although we here assume that all simulated x-rays are being scattered). According to (2.9), the x-ray weight factor is now adjusted by the amount

$$\pi_i(l_1, \Omega) = \mu^s l_{\text{full}} \exp\left[-(l_1 + l_2)(\mu^a + \mu^s)\right] \frac{\gamma(\Omega)}{g(\Omega)}. \tag{7.7}$$

In analogy with the source components, it is possible to define "interesting" directions for the scattering. One will then try to focus the scattered x-rays, choosing a $g(\Omega)$, which peaks around these directions. To do this, one uses (7.7), where the fraction $\gamma(\Omega)/g(\Omega)$ corrects for the focusing. One must choose a proper distribution so that $g(\Omega) > 0$ in every interesting direction. If this is not the case, the Monte Carlo simulation gives incorrect results. All samples have been constructed with a focusing and a non-focusing option.

### 7.0.3. Future development of sample components

There is still room for much more development of functionality in McXtrace samples.

## 7.1. Absorption_sample: An absorption phantom

**Input Parameters for component `Absorption_sample` from samples**

```
1 <Parameter = value>, [Unit], Description
```

This component models an absorption phantom with a single inclusion in surrounding material. It is intended use is for tomographic imaging simulations.

## 7.2. Saxs_spheres: A model of dilute hard spheres in solution for SAXS-use

**Input Parameters for component `Saxs_spheres` from samples**

```
1 <Parameter = value>, [Unit], Description
```

## 7.3. PowderN: A general powder sample

**Input Parameters for component `PowderN` from samples**

```
1 <Parameter = value>, [Unit], Description
```

The powder diffraction component **PowderN** models a powder sample with background coming only from incoherent scattering and no multiple scattering. At the users choice, a given percentage of the incoming events may be transmitted (attenuated) to model the direct beam. The component can also assume *concentric* shape, i.e. be used for describing sample environment (cryostat, sample container etc.).

The description of the powder comes from a file in one of the standard output formats LAZY, FULLPROF, or CRYSTALLOGRAPHICA.

### 7.3.1. Files formats: powder structures

Data files of type `lau` and `laz` in the McXtrace distribution data directory are self-documented in their header.

```
PowderN(<geometry parameters>, filename="Al.laz")
```

Other column-based file formats may also be imported e.g. with parameters such as:

```
format=Crystallographica
format=Fullprof
format={1,2,3,4,0,0,0,0}
```

In the latter case, the indices define order of columns parameters multiplicity, lattice spacing, $F^2$, Debye-Waller factor and intrinsic line width.

The column signification may as well explicitely be set in the data file header using any of the lines:

```
#column_j     <index of the multiplicity 'j' column>
#column_d     <index of the d-spacing 'd' column>
#column_F2    <index of the squared str. factor '|F|^2' column [b]>
#column_F     <index of the structure factor norm '|F|' column>
#column_DW    <index of the Debye-Waller factor 'DW' column>
#column_Dd    <index of the relative line width Delta_d/d 'Dd' column>
#column_inv2d <index of the 1/2d=sin(theta)/lambda 'inv2d' column>
#column_q     <index of the scattering wavevector 'q' column>
```

Other component parameters may as well be specified in the data file header with lines e.g.:

```
#V_rho        <value of atom number density [at/Angs^3]>
#Vc           <value of unit cell volume Vc [Angs^3]>
#sigma_abs    <value of Absorption cross section [barns]>
```

```
#sigma_inc    <value of Incoherent cross section [barns]>
#Debye_Waller <value of Debye-Waller factor DW>
#Delta_d/d    <value of Delta_d/d width for all lines>
#density      <value of material density [g/cm^3]>
#weight       <value of material molar weight [g/mol]>
#nb_atoms     <value of number of atoms per unit cell>
```

Further details on file formats are available in the `mxdoc` page of the component.

### 7.3.2. Geometry, physical properties, concentricity

The sample has the shape of a solid cylinder, radius $r$ and height $h$ or a box-shaped sample of size *xwidth* x *yheight* x *zdepth*. At the users choice, an inner 'hollow' can be specified using the parameter *thickness*.

PowderN can assume a *concentric* shape, i.e. can contain other components inside the inner hollow. To allow this, two almost identical copies of the PowderN components must be set up *around* the internal component(s), for example:

```
COMPONENT Cryo = PowderN(reflections="Al.laz", radius = 0.01, thickness = 0.001,
                         concentric = 1)
AT (0,0,0) RELATIVE Somewhere

COMPONENT Sample = some_other_component(with geometry FULLY enclosed in the hollow)
AT (0,0,0) RELATIVE Somewhere

COMPONENT Cryo2 = COPY(Cryo)(concentric = 0)
AT (0,0,0) RELATIVE Somewhere
```

As outlined, the first instance of PowderN *must* have `concentric = 1` and the second instance *must* have `concentric = 0`. Furthermore, the component(s) inside the hollow *must* have a geometry which can be fully contained inside the hollow.

In addition to the coherent scattering specified in the `reflections` file, absorption- and incoherent cross sections can be given using the input parameters $\sigma_c^a$ and $\sigma_i^s$.

The Bragg scattering from the powder, $\sigma_c^s$ is calculated from the input file, with the parameters $Q$, $|F(Q)|^2$, and $j$ for the scattering vector, structure factor, and multiplicity, respectively. The volume of the unit cell is denoted $Vc$, while the sample packing factor is $f_{\text{pack}}$.

Focusing is performed by only scattering into one angular interval, $d\phi$ of the Debye-Scherrer circle. The center of this interval is located at the point where the Debye-Scherrer circle intersects the half-plane defined by the initial velocity, $\boldsymbol{v}_{\text{i}}$, and a user-specified vector, $\boldsymbol{f}$.

### 7.3.3. Powder scattering

An ideal powder sample consists of many small crystallites, although each crystallite is sufficiently large not to cause measurable size broadening. The orientation of the

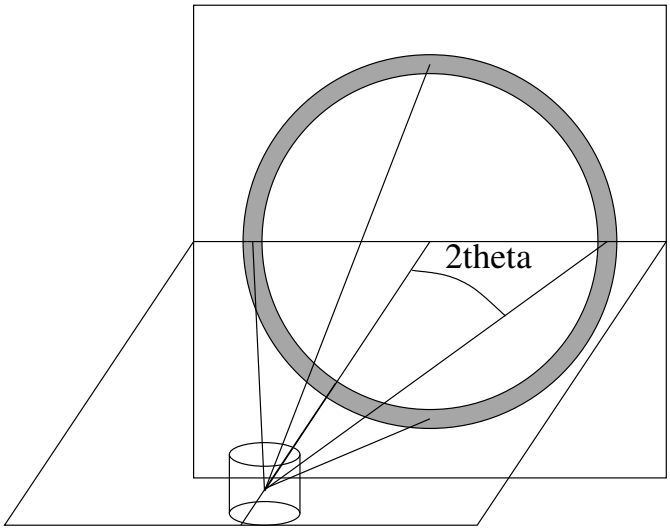


Figure 7.2.: The scattering geometry of a powder sample showing part of the Debye-Scherrer cone (solid lines) and the Debye-Scherrer circle (grey).

crystallites is evenly distributed, and there is thus always a large number of crystallites oriented to fulfill the Bragg condition

$$n\lambda = 2d\sin\theta, \tag{7.8}$$

where $n$ is the order of the scattering (an integer), $\lambda$ is the x-ray wavelength, $d$ is the lattice spacing of the sample, and $2\theta$ is the scattering angle, see figure 7.2. As all crystal orientations are realised in a powder sample, the x-rays are scattered within a *Debye-Scherrer cone* of opening angle $4\theta$ [Bac75].

Equation (7.8) may be cast into the form

$$|\boldsymbol{Q}| = 2|\boldsymbol{k}|\sin\theta, \tag{7.9}$$

where $\mathbf{Q}$ is a vector of the reciprocal lattice, and $\mathbf{k}$ is the wave vector of the x-ray. It is seen that only reciprocal vectors fulfilling $|\boldsymbol{Q}| < 2|\boldsymbol{k}|$ contribute to the scattering. For a complete treatment of the powder sample, one needs to take into account all these $\mathbf{Q}$-values, since each of them contribute to the attenuation.

The strength of the Bragg reflections is given by their structure factors

$$\left|\sum_j b_j \exp(\boldsymbol{R}_j \cdot \boldsymbol{Q})\right|^2, \tag{7.10}$$

where the sum runs over all atoms in one unit cell. This structure factor is non-zero only when $Q$ equals a reciprocal lattice vector.

The textbook expression for the scattering cross section corresponding to one Debye-Scherrer cone reads [Squ78, ch.3.6], with $V = NV_0$ being the total sample volume:

$$\sigma_{\text{cone}} = \frac{V}{V_0^2} \frac{\lambda^3}{4 \sin\theta} \sum_Q |F(Q)|^2. \tag{7.11}$$

For our purpose, this expression should be changed slightly. Firstly, the sum over structure factors for a particular $Q$ is replaced by the sum over essentially different reflections multiplied by their multiplicity, $j$. Then, a finite packing factor, $f$, is defined for the powder, and finally, the Debye-Waller factor is multiplied on the elastic cross section to take lattice vibrations into account (no inelastic background is simulated, however). We then reach

$$\begin{aligned} \sigma_{\text{cone,Q}} &= j_Q f \exp(-2W) \frac{V}{V_0^2} \frac{\lambda^3}{4 \sin\theta} |F(Q)|^2 & (7.12) \\ &= f \exp(-2W) \frac{N}{V_0} \frac{4\pi^3}{k^2} \frac{j_Q |F(Q)|^2}{Q} & (7.13) \end{aligned}$$

in the thin sample approximation. For samples of finite thickness, the beam is being attenuated by the attenuation coefficient

$$\mu_{\text{Q}} = \sigma_{\text{cone,Q}} / V. \tag{7.14}$$

For calibration it may be useful to consider the total intensity scattered into a detector of effective height $h$, covering only one reflection [Squ78, ch.3.6]. A cut though the Debye-Scherrer cone perpendicular to its axis is a circle. At the distance $r$ from the sample, the radius of this circle is $r \sin(2\theta)$. Thus, the detector (in a small angle approximation) counts a fraction $h/(2\pi r \sin(2\theta))$ of the scattered x-rays, giving a resulting count intensity:

$$I = \Psi \sigma_{\text{cone,Q}} \frac{h}{2\pi r \sin(2\theta)}, \tag{7.15}$$

where $\Psi$ is the flux at the sample position.

For clarity we repeat the meaning and unit of the symbols:

| | | |
|---|---|---|
| $\Psi$ | $\text{s}^{-1}\text{m}^{-2}$ | Incoming intensity of x-rays |
| $I$ | $\text{s}^{-1}$ | Detected intensity of x-rays |
| $h$ | m | Height of detector |
| $r$ | m | Distance from sample to detector |
| $f$ | 1 | Packing factor of the powder |
| $j$ | 1 | Multiplicity of the reflection |
| $V_0$ | $\text{m}^3$ | Volume of unit cell |
| $\lvert F(\boldsymbol{Q})\rvert^2$ | $\text{m}^2$ | Structure factor |
| $\exp(-2W)$ | 1 | Debye-Waller factor |
| $\mu_{\text{Q}}$ | $\text{m}^{-1}$ | Linear attenuation factor due to scattering from one powder line. |

A powder sample will in general have several allowed reflections $\boldsymbol{Q}_j$, which will all contribute to the attenuation. These reflections will have different values of $|F(\boldsymbol{Q}_j)|^2$ (and hence of $Q_j$), $j_j$, $\exp(-2W_j)$, and $\theta_j$. The total attenuation through the sample due to scattering is given by $\mu^s = \mu^s_{\text{inc}} + \sum_j \mu^s_j$, where $\mu^s_{\text{inc}}$ represents the incoherent scattering.

### 7.3.4. Algorithm

The algorithm of **PowderN** can be summarized as

- Check if the x-ray intersects the sample (otherwise ignore the following).
- Calculate the attenuation coefficients for scattering and absorption.
- Perform Monte Carlo choices to determine the scattering position, scattering type (coherent/incoherent), and the outgoing direction.
- Perform the necessary weight factor transformation.

## 7.4. Perfect_crystal: A Darwin-width domniated single crystal model

**Input Parameters for component `Perfect_crystal` from samples**

```
1 <Parameter = value>, [Unit], Description
```

Pedning Further Documetation

## 7.5. Single_crystal: The single crystal component

**Input Parameters for component `Single_crystal` from samples**

```
1 <Parameter = value>, [Unit], Description
```

Documentation Pending

## 7.6. Molecule_2state: Excitable time-dependent sample model

**Input Parameters for component `Molecule_2state` from samples**

```
1 <Parameter = value>, [Unit], Description
```

Further Documentaion pending

# 8. Monitors and detectors

In real scattering experiments, detectors and monitors play quite different roles. One wants the detectors to be as efficient as possible, counting all photons (absorbing them in the process), while the monitors measure the intensity of the incoming beam, and must as such be almost transparent, interacting only with (roughly) 0.1-1% of the photons passing by. In computer simulations, it is of course possible to detect every xray without absorbing it or disturbing any of its parameters. Hence, the two components have very similar functions in the simulations, and we do not distinguish between them. For simplicity, they are from here on just called **monitors**.

Another important difference between computer simulations and real experiments is that one may allow the monitor to be sensitive to any xray property, as *e.g.* direction, energy, and divergence, in addition to what is found in real-world detectors (space and time). One may, in fact, let the monitor record correlations between these properties.

When a monitor detects a xray, a number counting variable is incremented: $n_i = n_{i-1} + 1$. In addition, the photon weight $p_i$ is added to the weight counting variable: $I_i = I_{i-1} + p_i$, and the second moment of the weight is updated: $M_{2,i} = M_{2,i-1} + p_i^2$. As also discussed chapter 2, after a simulation of $N$ rays the detected intensity (in units of photonts/sec.) is $I_N$, while the estimated errorbar is $\sqrt{M_{2,N}^2}$.

Several different monitor components have been developed for McXtrace, but we have decided to support only the most important ones. One example of the monitors we have omitted is the single monitor, **Monitor**, that measures just one number (with errorbars) per simulation. This effect is mirrored by any of the 1- or 2-dimensional components we support, e.g. the `PSD_monitor`. In case additional functionality of monitors is required, a few lines of code in existing monitors can easily be modified.

Another solution is the "Swiss army knife" of monitors, **Monitor_nD**, that can handle almost any simulation requirement, but may prove challenging for inexperienced users or users who like to make their own modifications.

## 8.1. Monitor: Simple intensity monitor

**Input Parameters for component `Monitor` from monitors**

```
<Parameter = value>, [Unit], Description
```

The **Monitor** component is a simple photon counter that merely detects the integrated intensity in an aperture *xwidth* by *yheight* $m^2$. It does *not* write a separate datafile, but reports the $I, I_err, N$-signals to the console. The signals represent intensity, error estimate, and number of statistical events (photon rays). If a scan is performed, the intensity recorded by **Monitor** will be written to the scan datafile.

## 8.2. E_monitor: The energy-sensitive monitor

**Input Parameters for component `E_monitor` from monitors**

```
<Parameter = value>, [Unit], Description
```

The component **E_monitor** is sensitive to the xray energy, which in binned in $nE$ bins between $E_{\min}$ and $E_{\max}$ (in keV).

The output parameters from `E_monitor` are the total counts, and a file with 1-dimensional data vs. $E$, similar to `TOF_monitor`.

## 8.3. L_monitor: The wavelength sensitive monitor

**Input Parameters for component `L_monitor` from monitors**

```
<Parameter = value>, [Unit], Description
```

The component **L_monitor** is very similar to **E_monitor**. This component is just sensitive to the xray wavelength. The wavelength spectrum is output in a one-dimensional histogram. between $\lambda_{\min}$ and $\lambda_{\max}$ (measured in Å).

As for the two other 1-dimensional monitors, this component outputs the total counts and a file with the histogram.

## 8.4. PSD_monitor: The PSD monitor

**Input Parameters for component `PSD_monitor` from monitors**

```
<Parameter = value>, [Unit], Description
```

The component `PSD_monitor` resembles other monitors, e.g. `Monitor`, and also propagates the photon ray to the detector surface in the $(x, y)$-plane, where the detector window is set by the $x$ and $y$ input coordinates. The PSD monitor is sensitive to the

arrival position of the of the photon ray. The rectangular monitor window, given by the $x$ and $y$ limits is divided into $n_x \times n_y$ pixels.

The output from `PSD_monitor` is the integrated counts, $n, I, M_2$, as well as three two-dimensional arrays of counts: $n(x,y), I(x,y), M_2(x,y)$. The arrays are written to a file, `filename`, and can be read e.g. by the tool **mxplot**, see the system manual.

## 8.5. PSD_monitor_coh: The coherent PSD monitor

**Input Parameters for component `PSD_monitor_coh` from monitors**

```
1 <Parameter = value>, [Unit], Description
```

The component `PSD_monitor_coh` closely resembles its parent the **PSD_monitor**, and also propagates the photon ray to the detector surface in the $(x,y)$-plane, where the detector window is set by the $x$ and $y$ input coordinates. The coherent PSD monitor, though, is also sensitive considers the phase of the photon ray and emits not one but two datafiles, where one (*filename*.abs) represents the coherent intensity and the other (*filename*.arg) the phase of the collected beam.

The first output file (.abs) from `PSD_monitor_coh` is the integrated counts, $n, I, M_2$, as well as three two-dimensional arrays of counts: $n(x,y), I(x,y), M_2(x,y)$. These arrays are written to a file, `filename.abs`, and can be read e.g. by the tool **mxplot**, see the system manual. Likewise, the second output file (.arg) contains the integrated phases, and are written to the file `filename.arg`

## 8.6. PSD_monitor_4PI: A 4 PI steradian spherical monitor.

**Input Parameters for component `PSD_monitor_4PI` from monitors**

```
1 <Parameter = value>, [Unit], Description
```

**PSD_monitor_4PI** is an unphysical idealized component. It takes on the shape of a complete sphere with a given *radius*, records the longitude and latitude of photon rays as the pass through the sphere. The sphere is binned in constant spaced bins in longitude and latitude. This has the consequence that the image will be distorted when plotted onto a plane, much like projections of a world map.

The output from `PSD_monitor` is the integrated counts, $n, I, M_2$, as well as three two-dimensional arrays of counts: $n(x,y), I(x,y), M_2(x,y)$. The arrays are written to a file, `filename`, and can be read e.g. by the tool **mxplot**, see the system manual.

## 8.7. EPSD_monitor: Energy-selective PSD monitor

**Input Parameters for component `EPSD_monitor` from monitors**

```
1 <Parameter = value>, [Unit], Description
```

The **EPSD_monitor** closely resembles **PSD_monitor** with the difference that this component may be given an energy interval: $[E_{min}, E_{max}]$, in which it is sensitive. Photon rays falling within this interval are detected, those outside are ignored.

The output from **EPSD_monitor** is the integrated counts, $n, I, M_2$, as well as three two-dimensional arrays of counts: $n(x,y), I(x,y), M_2(x,y)$. The arrays are written to a file, `filename`, and can be read e.g. by the tool **mxplot**, see the system manual.

## 8.8. W_psd_monitor: A power vs. position monitor

**Input Parameters for component `W_psd_monitor` from monitors**

```
1 <Parameter = value>, [Unit], Description
```

doc. pend.

## 8.9. Monitor_nD: A general Monitor for 0D/1D/2D records

**Input Parameters for component `Monitor_nD` from monitors**

```
1 <Parameter = value>, [Unit], Description
```

The component **Monitor_nD** is a general Monitor that may output any set of physical parameters regarding the passing photons. The generated files are either a set of 1D signals ([Intensity] *vs.* [Variable]), or a single 2D signal ([Intensity] *vs.* [Variable 1] *vs.* [Variable 2]), and possibly a simple long list of selected physical parameters for each photon ray.

The input parameters for **Monitor_nD** are its dimensions *xmin, xmax, ymin, ymax* (in metres) and an *options* string describing what to detect, and what to do with the signals, in clear language. The *xwidth, yheight, zdepth* may also be used to enter dimensions.

Eventhough the possibilities of Monitor_nD are numerous, its usage remains as simple as possible, specially in the *options* parameter, which 'understands' normal language. The formatting of the *options* parameter is free, as long as it contains some specific keywords, that can be sometimes followed by values. The *no* or *not* option modifier will revert next option. The *all* option can also affect a set of monitor configuration parameters (see below).

As the usage of this component enables to monitor virtually anything, and thus the combinations of options and parameters is infinite, we shall only present the most basic configuration. The reader should refer to the on-line component help, using e.g. `mcdoc Monitor_nD.comp`.

### 8.9.1. The Monitor_nD geometry

The monitor shape can be selected among seven geometries:

1. (*square*) The default geometry is flat rectangular in $(xy)$ plane with dimensions $x_{\min}, x_{\max}, y_{\min}$, $y_{\max}$, or $x_{width}, y_{height}$.
2. (*box*) A rectangular box with dimensions $x_{width}, y_{height}, z_{depth}$.
3. (*disk*) When choosing this geometry, the detector is a flat disk in $(xy)$ plane. The radius is then
$$\text{radius} = \max(\text{abs}\ [x_{\min}, x_{\max}, y_{\min}, y_{\max}, x_{width}/2, y_{height}/2]). \tag{8.1}$$
4. (*sphere*) The detector is a sphere with the same radius as for the *disk* geometry.
5. (*cylinder*) The detector is a cylinder with revolution axis along $y$ (vertical). The radius in $(xz)$ plane is
$$\text{radius} = \max(\text{abs}\ [x_{\min}, x_{\max}, x_{width}/2]), \tag{8.2}$$

and the height along $y$ is

$$\text{height} = |y_{\max} - y_{\max}| \text{or} y_{height}. \tag{8.3}$$

6. (*banana*) The same as the cylinder, but without the top/bottom caps, and on a restricted angular range. The angular range is specified using a `theta` variable limit specification in the `options`.

7. (*previous*) The detector has the shape of the previous component. This may be a surface or a volume. In this case, the photon is detected on the previous component, and there is no photon propagation.

By default, the monitor is flat, rectangular. Of course, you can choose the orientation of the **Monitor_nD** in the instrument description file with the usual `ROTATED` modifier.

For the *box*, *sphere* and *cylinder*, the outgoing photons are monitored by default, but you can choose to monitor incoming photons with the *incoming* option.

At last, the *slit* or *absorb* option will ask the component to absorb the photons that do not intersect the monitor. The *exclusive* option word removes photons which are similarly outside the monitor limits (that may be other than geometrical).

The *parallel* option keyword is of common use in the case where the **Monitor_nD** is superposed with other components. It ensures that photons are detected independently of other geometrical constrains. This is generally the case when you need e.g. to place more than one monitor at the same place.

### 8.9.2. The photon parameters that can be monitored

There are many different variables that can be monitored at the same time and position. Some can have more than one name (e.g. `energy` or `omega`).

```
1     kx ky kz k wavevector [Angs-1] (     usually axis are
2     vx vy vz v            [m/s]          x=horz., y=vert., z=on axis)
3     x y z                 [m]       Distance, Position
4     kxy vxy xy radius     [m]       Radial wavevector, velocity and position
5     t time                [s]       Time of Flight
6     energy omega          [meV]
7     lambda wavelength     [Angs]
8     p intensity flux      [n/s] or [n/cm^2/s]
9     ncounts               [1]
10    sx sy sz              [1]       Spin
11    vdiv ydiv dy          [deg]     vertical divergence (y)
12    hdiv divergence xdiv  [deg]     horizontal divergence (x)
13    angle                 [deg]     divergence from  direction
14    theta longitude       [deg]     longitude (x/z) [for sphere and cylinder
          ]
15    phi   lattitude       [deg]     lattitude (y/z) [for sphere and cylinder
          ]
```

as well as four other special variables

```
1    user user1              will monitor the [Mon_Name]_Vars.UserVariable
         {1|2}
2    user2 user3             to be assigned in an other component (see below)
```

To tell the component what you want to monitor, just add the variable names in the *options* parameter. The data will be sorted into *bins* cells (default is 20), between some default *limits*, that can also be set by user. The *auto* option will automatically determine what limits should be used to have a good sampling of signals.

### 8.9.3. Important options

Each monitoring records the flux (sum of weights $p$) versus the given variables, except if the `signal=<variable>` word is used in the `options`.

The `auto` option is probably the most useful one: it asks the monitor to determine automatically the best limits for each variable, in order to obtain the most significant monitored histogram. This option should preceed each variable, or be located after all variables in which case they are all affected. On the other hand, one may manually set the limits with the `limits=[min max]` option. If no limits are set monitor_nd uses predefined limits that usually make sense for most x-ray scattering simulations. Example: the default upper energy limit is 100 meV, but may be changed with an options string like `options="energy limits 0 200"`. Note that the limits also apply in list mode (see below).

The `log` and `abs` options should be positioned before each variable to specify logarithmic binning and absolute value respectively.

The *borders* option will monitor variables that are outside the limits. These values are then accumulated on the 'borders' of the signal.

### 8.9.4. The output files

By default, the file names will be the component name, followed by a time stamp and automatic extensions showing what was monitored (such as `MyMonitor.x`). You can also set the filename in *options* with the *file* keyword followed by the file name that you want. The extension will then be added if the name does not contain a dot (.). Finally, the *filename* parameter may also be used.

The output files format are standard 1D or 2D McXtrace detector files. The *no file* option will *inactivate* monitor, and make it a single 0D monitor detecting integrated flux and counts. The *verbose* option will display the nature of the monitor, and the names of the generated files.

#### The 2D output

When you ask the **Monitor_nD** to monitor only two variables (e.g. *options* = "x y"), a single 2D file of intensity versus these two correlated variables will be created.

| McXtrace monitor | Monitor_nD equivalent |
|---|---|
| E_monitor | *options*="energy bins=*nchan* limits=[$E_{min}E_{max}$]" |
| EPSD_monitor | *options*="energy bins=$n_E$ limits=[$E_{min}E_{max}$], x bins=$nx$" $xmin$=$x_{min}$ $xmax$=$x_{max}$ |
| L_monitor | *options*="lambda bins=$nh$ limits=[$-\lambda_{max}/2\lambda_{max}/2$]" *filename*=$file$ |
| Monitor | *options*="inactivate" |
| PSD_monitor_4PI | *options*="theta y, sphere" |
| PSD_monitor | *options*="x bins=$nx$, y bins=$ny$" $xmin$=$x_{min}$ $xmax$=$x_{max}$ $ymin$=$y_{min}$ $ymax$=$y_{max}$ *filename*=$file$ |

Table 8.1.: Using Monitor_nD in place of other components. All limits specifications may be advantageously replaced by an *auto* word preceeding each monitored variable. Not all file and dimension specifications are indicated (e.g. filename, xmin, xmax, ymin, ymax).

#### The 1D output

The **Monitor_nD** can produce a set of 1D files, one for each monitored variable, when using 1 or more than 2 variables, or when specifying the *multiple* keyword option.

#### The List output

The **Monitor_nD** can additionally produce a *list* of variable values for photons that pass into the monitor. This feature is additive to the 1D or 2D output. By default only 1000 events will be recorded in the file, but you can specify for instance "*list* 3000 photons" or "*list all* photons". This last option may require a lot of memory and generate huge files. Note that the limits to the measured parameters also apply in this mode. To exemplify, a monitor_nd instance with the option string `"list all k"` will *only* record those photons which have a below 2000 AA$^{-1}$, whereas an instance with the option string `"list all kx ky kz 0 2000"` will record all photons with $|k_x, k_y| < 2000$ AA$^{-1}$ and $0 < v_z < 2000$ AA$^{-1}$. Thus, in this latter case, any neutron travelling in the negative z-direction will be disregarded.

### 8.9.5. Monitor equivalences

In the following table 8.1, we show how the Monitor_nD may substitute any other McXtrace monitor.

### 8.9.6. Usage examples

```
COMPONENT MyMonitor = Monitor\_nD(
    xmin = -0.1, xmax = 0.1,
    ymin = -0.1, ymax = 0.1,
```

```
4     options = "energy auto limits")
```

will monitor the photon energy in a single 1D file (a kind of E_monitor)

- `options = "banana, theta limits=[10,130], bins=120, y bins=30"`
  is a theta/height banana detector.
- `options = "banana, theta limits=[10,130], auto time"`
  is a theta/time-of-flight banana detector.
- `options="x bins=30 limits=[-0.05 0.05] ; y"`
  will set the monitor to look at $x$ and $y$. For $y$, default bins (20) and limits values (monitor dimensions) are used.
- `options="x y, auto, all bins=30"`
  will determine itself the required limits for $x$ and $y$.
- `options="multiple x bins=30, y limits=[-0.05 0.05], all auto"`
  will monitor the photon $x$ and $y$ in two 1D files.
- `options="x y z kx ky kz, all auto"`
  will monitor each of these variables in six 1D files.
- `options="x y z kx ky kz, list all, all auto"`
  will monitor all these photons' variables in one long list, one row per photon event.
- `options="multiple x y z kx ky kz, and list 2000, all auto"`
  will monitor all the photons' variables in one list of 2000 events and in six 1D files.
- `options="signal=energy, x y"`
  is a PSD monitor recording the mean energy of the beam as a function of $x$ and $y$.

### 8.9.7. Monitoring user variables

There are two ways to monitor any quantity with Monitor_nD. This may be e.g. the number of reflections in a mirror system, or the wavevector and energy transfer at a sample. The only requirement is to define the `user1` (and optionally `user2`,`user3`) variables of a given Monitor_nD instance.

#### Directly setting the user variables (simple)

The first method uses the `user1` and `username1` component parameters to directly transfer the value and label, such as in the following example:

```
TRACE
(...)
COMPONENT UserMonitor = Monitor_nD(
  user1    = log(t), username1="Log(time)",
  options  ="auto user1")
```

The values to assign to `user2` and `user3` must be global instrument variables, or a component output variable as in `user1=MC_GETPAR(some_comp, outpar)`. Similarly, the `user2,user3` and `username2,username3` parameters may be used to control the second and third user variable, to produce eventually 2D/3D user variable correlation data and custom event lists.

### Setting indirectly the user variables (only for professionals)

It is possible to control the user variables of a given Monitor_nD instance anywhere in the instrument description. This method requires more coding, but has the advantage that a variable may be defined to store the result of a computation locally, and then transfer it into the UserMonitor, all fitting in an EXTEND block.

This is performed in a 4-step process:

1. Declare that you intend to monitor user variables in a Monitor_nD instance (defined in TRACE):

```
DECLARE
%{ (...)
  %include "monitor_nd-lib"
  MONND\_DECLARE(UserMonitor); // will monitor custom things in
      UserMonitor
%}
```

2. Initialize the label of the user variable (optional):

```
INITIALIZE
%{
   (...)
  MONND\_USER\_TITLE(UserMonitor, 1, "Log(time)");
%}
```

   The value '1' could be '2' or '3' for the `user2,user3` variable.

3. Set the user variable value in a TRACE component EXTEND block:

```
TRACE
(...)
COMPONENT blah = blah\_comp(...)
EXTEND
%{ // attach a value to user1 in UserMonitor, could be much more
    complex here.
  MONND\_USER\_VALUE(UserMonitor, 1, log(t));
%}
(...)
```

4. Tell the Monitor_nD instance to record user variables:

```
TRACE
(...)
COMPONENT UserMonitor = Monitor\_nD(options="auto user1")
(...)
```

Setting the user variable values may either make use of the photon parameters (x,y,z, vx,vy,vz, phi, t, Ex,Ey,Ez, p), access the internal variables of the component that sets the user variables (in this example, those from the `blah` instance), access any component OUTPUT parameter using the `MC_GETPAR` C macro(see chapter A), or simply use a global instrument variable. Instrument parameters can not be used directly.

## 8.10. PreMonitor_nD: Store photon rays for possible later detection.

**Input Parameters for component `PreMonitor_nD` from monitors**

```
<Parameter = value>, [Unit], Description
```

The first immediate usage of the **Monitor_nD** component is when one requires to identify cross-correlations between some photon parameters, e.g. position and divergence (*aka* phase-space diagram). This latter monitor would be merely obtained with:

```
options="x dx, auto", bins=30
```

This example records the correlation between position and divergence of photons at a given instrument location.

It is also possible to search for cross-correlation between two part of the instrument simulation. One example is the acceptance phase-diagram, which shows the photon caracteristics at the input required to reach the end of the simulation. This *spatial* correlation may be revealed using the **PreMonitor_nD** component. This latter stores the photon parameters at a given instrument location, to be used at an other **Monitor_nD** location for monitoring.

The only parameter of **PreMonitor_nD** is the name of the associated Monitor_nD instance, which should use the `premonitor` option, as in the following example:

```
COMPONENT CorrelationLocation = PreMonitor_nD(comp = CorrelationMonitor)
AT (...)

  (... e.g. a bunch of optics )

COMPONENT CorrelationMonitor  = Monitor_nD(
   options="x dx, auto, all bins=30, premonitor")
AT (...)
```

which performs the same monitoring as the previous example, but with a spatial correlation constrain. Indeed, it records the position *vs* the divergence of photons at the correlation location, but only if they reach the monitoring position. All usual **Monitor_nD** variables may be used, except the user variables. The latter may be defined as described in section 8.9.7 in an EXTEND block.

# 9. Special-purpose components

The chapter deals with components that are not easily included in any of the other chapters because of their special nature, but which are still part of the McXtrace system.

One part of these components deals with splitting simulations into two (or more) stages. For example, the front end of a beamline is often not changed much, and a long simulation of photon rays "surviving" through initial optics could be reused for several simulations of the beamline back-end, speeding up the simulations by (typically) one or two orders of magnitude. The components for doing this trick is **Virtual_input** and **Virtual_output**, which stores and reads photon rays, respectively.

For integration with SHADOW [Rio+11] the components **Shadow_inpt** and **Shadow_output** have been written.

**Progress_bar** is a simulation utility that displays the simulation status, but assumes the form of a component.

## 9.1. Progress_bar: Dynamic information output

**Input Parameters for component `Progress_bar` from misc**

```
<Parameter = value>, [Unit], Description
```

The component **Progress_bar** is a special version of an Arm and has no effect on the x-ray. Only one instance of progress bar is allowed - usually it is the very first component of a beamline simulation.

If *percent*$> 0$ (default is 10%) the porgress bar prints an update when approximately *percent* % of the total number of rays have been simulated. If *minutes*$> 0$ (overrides *percen*) this update happens approx. every *minutes* min. If *flag_save*$\neq 0$, the update process also includes saving the status of all monitors to disk. This is useful if something is likely to interrupt the progress of long simulations ot avoid loss of data. *profile* may be set to save the simulation profile to a filename other than the name of the instrument (the default).

## 9.2. Virtual_output: Saving the first part of a split simulation

**Input Parameters for component `Virtual_output` from misc**

```
<Parameter = value>, [Unit], Description
```

The component **Virtual_output** stores the photon ray parameters at the end of the first part of a split simulation. The idea is to let the next part of the split simulation be performed by another instrument file, which reads the stored photon ray parameters by the component **Virtual_input**.

All photon ray parameters are saved to the output file, which is by default of "text" *type*, but can also assume the binary formats "float" or "double". The storing of photon rays continues until the specified number of simulations have been performed.

*buffer-size* may be used to limit the size of the output file, but absolute intentities are then likely to be wrong. Exept when using MPI, we recommend to use the default value of zero, saving all photon rays. The size of the file is then controlled indirectly with the general *ncounts* parameter.

## 9.3. Virtual_input: Starting the second part of a split simulation

**Input Parameters for component `Virtual_input` from misc**

```
<Parameter = value>, [Unit], Description
```

The component **Virtual_input** resumes a split simulation where the first part has been performed by another instrument and the photon ray parameters have been stored by the component **Virtual_output**.

All photon ray parameters are read from the input file, which is by default of "text" *type*, but can also assume the binary formats "float" and "double". Reading of photon rays continues until the specified number of rays have been simulated or the file has been exhausted. If desirable, the input file can be reused a number of times, determined by the optional parameter *repeat-count*. This is only useful if the present simulation makes use of MC choices, otherwise the same outcome will result for each repetition of the simulation (see chapter 2).

Care should be taken when dealing with absolute intensities, which will be correct only when the input file has been exhausted at least once.

The simulation ends with either the end of the repeated file counts, or with the normal end with *ncount* McXtrace simulation events. We recommend controlling the simulation on *repeat-count* by using a larger ncount value.

## 9.4. Shadow_input: Reading input from Shadow

**Input Parameters for component `Shadow_input` from misc**

```
1 <Parameter = value>, [Unit], Description
```

Documentation pending

## 9.5. Shadow_output: Saving the photon rays for use with SHADOW

**Input Parameters for component `Shadow_output` from misc**

```
1 <Parameter = value>, [Unit], Description
```

Documentation pending.

# A. Libraries and conversion constants

The McXtrace Library contains a number of built-in functions and conversion constants which are useful when constructing components. These are stored in the `share` directory of the `MCXTRACE` library.

Within these functions, the 'Run-time' part is available for all component/instrument descriptions. The other parts are dynamic, that is they are not pre-loaded, but only imported once when a component requests it using the `%include` McXtrace keyword. For instance, within a component C code block, (usually SHARE or DECLARE):

```
1  %include "read_table-lib"
```

will include the 'read_table-lib.h' file, and the 'read_table-lib.c' (unless the `--no-runtime` option is used with `mcxtrace`). Similarly,

```
1  %include "read_table-lib.h"
```

will *only* include the 'read_table-lib.h'. The library embedding is done only once for all components (like the SHARE section). For an example of implementation, see **Res_monitor**.

In this Appendix, we present a short list of both each of the library contents and the run-time features.

## A.1. Run-time calls and functions (`mcxtrace-r`)

Here we list a number of preprogrammed macros and functions which may ease the task of writing component and instrument definitions. By convention macros are in upper case whereas functions are in lower case.

### A.1.1. Photon propagation

Propagation routines perform all necessary operations to transport x-rays from one point to an other. Except when using the special `ALLOW_BACKPROP;` call prior to executing any `PROP_*` propagation, the x-rays which have negative propagation lengths are removed automatically.

- **ABSORB**. This macro issues an order to the overall McXtrace simulator to interrupt the simulation of the current x-ray history and to start a new one.
- **PROP_Z0**. Propagates the x-ray to the $z = 0$ plane, by adjusting $(x, y, z)$, $\phi$, and $t$ accordingly from knowledge of the x-ray wavevector $(kx, ky, kz)$. If the propagation length is negative, the x-ray is absorbed, except if a `ALLOW_BACKPROP;` preceeds it.

For components that are centered along the $z$-axis, use the `_intersect` functions to determine intersection time(s), and then a `PROP_DL` call.

- **PROP_X0, PROP_Y0**. These macros are analogous to `PROP_Z0` except they propagate to the $x = 0$ and $y = 0$ planes respectively.

- **PROP_DL**($dl$). Propagates the x-ray by the length $dl$, adjusting $(x, y, z)$, $\phi$, $t$ accordingly, from knowledge of the x-ray wavevector.

- **ALLOW_BACKPROP**. Indicates that the *next* propagation routine will not remove the x-ray, even if negative propagation lengths are found. Subsequent propagations are not affected.

- **SCATTER**. This macro is used to denote a scattering event inside a component. It should be used to indicate that a component has interacted with the x-ray (e.g. scattered or detected). This does not affect the x-ray state (see, however, **Beamstop**), and it is mainly used by the `MCDISPLAY` section and the `GROUP` modifier. See also the SCATTERED variable (below).

### A.1.2. Coordinate and component variable retrieval

- **MC_GETPAR**($comp, outpar$). This may be used in e.g. the FINALLY section of an instrument definition to reference the parameters of a component.

- **NAME_CURRENT_COMP** gives the name of the current component as a string.

- **POS_A_CURRENT_COMP** gives the absolute position of the current component. A component of the vector is referred to as POS_A_CURRENT_COMP.$i$ where $i$ is $x$, $y$ or $z$.

- **ROT_A_CURRENT_COMP** and **ROT_R_CURRENT_COMP** give the orientation of the current component as rotation matrices (absolute orientation and the orientation relative to the previous component, respectively). A component of a rotation matrix is referred to as ROT_A_CURRENT_COMP$[m][n]$, where $m$ and $n$ are 0, 1, or 2 standing for $x, y$ and $z$ coordinates respectively.

- **POS_A_COMP**($comp$) gives the absolute position of the component with the name *comp*. Note that *comp* is not given as a string. A component of the vector is referred to as POS_A_COMP($comp$).$i$ where $i$ is $x$, $y$ or $z$.

- **ROT_A_COMP**($comp$) and **ROT_R_COMP**($comp$) give the orientation of the component *comp* as rotation matrices (absolute orientation and the orientation relative to its previous component, respectively). Note that *comp* is not given as a string. A component of a rotation matrice is referred to as ROT_A_COMP($comp$)$[m][n]$, where $m$ and $n$ are 0, 1, or 2.

- **INDEX_CURRENT_COMP** is the number (index) of the current component (starting from 1).
- **POS_A_COMP_INDEX**(*index*) is the absolute position of component *index*. POS_A_COMP_INDEX (INDEX_CURRENT_COMP) is the same as POS_A_CURRENT_COMP. You may use POS_A_COMP_INDEX (INDEX_CURRENT_COMP+1) to make, for instance, your component access the position of the next component (this is usefull for automatic targeting). A component of the vector is referred to as POS_A_COMP_INDEX(*index*).*i* where *i* is $x$, $y$ or $z$.
- **POS_R_COMP_INDEX** works the same as above, but with relative coordinates.
- **STORE_XRAY**($index, x, y, z, kx, ky, kz, phi, t, Ex, Ey, Ez, p$) stores the current x-ray state in the trace-history table, in local coordinate system. *index* is usually INDEX_CURRENT_COMP. This is automatically done when entering each component of an instrument.
- **RESTORE_XRAY**($index, x, y, z, kx, ky, kz, phi, t, Ex, Ey, Ez, p$) restores the x-ray state to the one at the input of the component *index*. To ignore a component effect, use RESTORE_XRAY (INDEX_CURRENT_COMP, $x, y, z, kx, ky, kz, phi, Ex, Ey, Ez, p$) at the end of its TRACE section, or in its EXTEND section. These x-ray states are in the local component coordinate systems.
- **SCATTERED** is a variable set to 0 when entering a component, which is incremented each time a SCATTER event occurs. This may be used in the `EXTEND` sections to determine whether the component interacted with the current x-ray.
- **extend_list**($n$, &*arr*, &*len*, *elemsize*). Given an array *arr* with *len* elements each of size *elemsize*, make sure that the array is big enough to hold at least $n$ elements, by extending *arr* and *len* if necessary. Typically used when reading a list of numbers from a data file when the length of the file is not known in advance.
- **mcset_ncount**($n$). Sets the number of x-ray histories to simulate to $n$.
- **mcget_ncount**(). Returns the number of x-ray histories to simulate (usually set by option `-n`).
- **mcget_run_num**(). Returns the number of x-ray histories that have been simulated until now.

### A.1.3. Coordinate transformations

- **coords_set**($x, y, z$) returns a Coord structure (like POS_A_CURRENT_COMP) with $x$, $y$ and $z$ members.
- **coords_get**($P$, &$x$, &$y$, &$z$) copies the $x$, $y$ and $z$ members of the Coord structure $P$ into $x, y, z$ variables.

- **coords_add**$(a, b)$, **coords_sub**$(a, b)$, **coords_neg**$(a)$ enable to operate on coordinates, and return the resulting Coord structure.
- **rot_set_rotation**(*Rotation t,* $\phi_x, \phi_y, \phi_z$) Get transformation matrix for rotation first $\phi_x$ around x axis, then $\phi_y$ around y, and last $\phi_z$ around z. $t$ should be a 'Rotation' ([3][3] 'double' matrix).
- **rot_mul***(Rotation t1, Rotation t2, Rotation t3)* performs $t3 = t1.t2$.
- **rot_copy***(Rotation dest, Rotation src)* performs $dest = src$ for Rotation arrays.
- **rot_transpose***(Rotation src, Rotation dest)* performs $dest = src^t$.
- **rot_apply***(Rotation t, Coords a)* returns a Coord structure which is $t.a$

### A.1.4. Mathematical routines

- **NORM**$(x, y, z)$. Normalizes the vector $(x, y, z)$ to have length 1.
- **scalar_prod**$(a_x, a_y, a_z, b_x, b_y, b_z)$. Returns the scalar product of the two vectors $(a_x, a_y, a_z)$ and $(b_x, b_y, b_z)$.
- **vec_prod**$(\&a_x,\&a_y,\&a_z,\ b_x,b_y,b_z,\ c_x,c_y,c_z)$. Sets $(a_x, a_y, a_z)$ equal to the vector product $(b_x, b_y, b_z) \times (c_x, c_y, c_z)$.
- **rotate**$(\&x,\&y,\&z,v_x,v_y,v_z,\varphi,a_x,a_y,a_z)$. Set $(x, y, z)$ to the result of rotating the vector $(v_x, v_y, v_z)$ the angle $\varphi$ (in radians) around the vector $(a_x, a_y, a_z)$.
- **normal_vec**$(n_x,\ n_y,\ n_z,\ x,\ y,\ z)$. Computes a unit vector $(n_x, n_y, n_z)$ normal to the vector $(x, y, z)$.*
- **solve_2nd_order**$(*t_0,*t_1,\ A,\ B,\ C)$. Solves the $2^{nd}$ order equation $At^2+Bt+C = 0$ and puts the solutions in $*t_0$ and $*t_1$. The smallest positive solution into pointer $*t_0$. If $t_1$=`NULL` it is ignored and the second solution is discarded.

### A.1.5. Output from detectors

Details about using these functions are given in the McXtrace User Manual.

- **DETECTOR_OUT_0D**(...). Used to output the results from a single detector. The name of the detector is output together with the simulated intensity and estimated statistical error. The output is produced in a format that can be read by McXtrace front-end programs.
- **DETECTOR_OUT_1D**(...). Used to output the results from a one-dimensional detector. Integrated intensities error etc. is also reported as for DETECTOR_OUT_0D.
- **DETECTOR_OUT_2D**(. . . ...). Used to output the results from a two-dimentional detector. Integrated intensities error etc. is also reported as for DETECTOR_OUT_0D.

- **mcinfo_simulation***(FILE *f, mcformat, char *pre, char *name)* is used to append the simulation parameters into file $f$ (see for instance **Res_monitor**). Internal variable $mcformat$ should be used as specified. Please contact the authors for further information.

### A.1.6. Ray-geometry intersections

- **inside_rectangle**($x$, $y$, $xw$, $yh$). Return 1 if $-xw/2 \leq x \leq xw/2$ AND $-yh/2 \leq y \leq yh/2$. Else return 0.

- **box_intersect**(&$l_1$, &$l_2$, $x$, $y$, $z$, $k_x$, $k_y$, $k_z$, $d_x$, $d_y$, $d_z$). Calculates the (0, 1, or 2) intersections between the x-ray path and a box of dimensions $d_x$, $d_y$, and $d_z$, centered at the origin for a x-ray with the parameters $(x, y, z, k_x, k_y, k_z)$. The intersection lengths are returned in the variables $l_1$ and $l_2$, with $l_1 < l_2$. In the case of less than two intersections, $t_1$ (and possibly $t_2$) are set to zero. The function returns true if the x-ray intersects the box, false otherwise.

- **cylinder_intersect**(&$l_1$, &$l_2$, $x$, $y$, $z$, $k_x$, $k_y$, $k_z$, $r$, $h$). Similar to **box_intersect**, but using a cylinder of height $h$ and radius $r$, centered at the origin.

- **sphere_intersect**(&$l_1$, &$l_2$, $x$, $y$, $z$, $k_x$, $k_y$, $k_z$, $r$). Similar to **box_intersect**, but using a sphere of radius $r$.

- **ellipsoid_intersect**(&$l_1$, &$l_2$, $x$, $y$, $z$, $k_x$, $k_y$, $k_z$, $a$,$b$,$c$,$Q$, ). Similar to **box_intersect**, but using an ellipsoid with half-axis $a$,$b$,$c$ oriented by the rotation matrix $Q$. If $Q = I$, $a$ is along the $x$-axis, $b$ along $y$ and $c$ along $z$

### A.1.7. Random numbers

By default McXtrace uses the included Mersenne Twister[MN98] algorithm for generating pseudo random numbers.

- **rand01**(). Returns a random number distributed uniformly between 0 and 1.

- **randnorm**(). Returns a random number from a normal distribution centered around 0 and with $\sigma = 1$. The algorithm used to sample the normal distribution is explained in Ref. [Pre+86, ch.7].

- **randpm1**(). Returns a random number distributed uniformly between -1 and 1.

- **randtriangle**(). Returns a random number from a triangular distribution between -1 and 1.

- **randvec_target_circle**(&$v_x$, &$v_y$, &$v_z$, &$d\Omega$, aim$_x$, aim$_y$, aim$_z$, $r_f$). Generates a random vector $(v_x, v_y, v_z)$, of the same length as (aim$_x$, aim$_y$, aim$_z$), which is targeted at a *disk* centered at (aim$_x$, aim$_y$, aim$_z$) with radius $r_f$ (in meters), and perpendicular to the *aim* vector.. All directions that intersect the circle are

chosen with equal probability. The solid angle of the circle as seen from the position of the x-ray is returned in $d\Omega$. This routine was previously called **randvec_target_sphere** (which still works).

- **randvec_target_rect_angular**(&$v_x$, &$v_y$, &$v_z$, &$d\Omega$, aim$_x$, aim$_y$, aim$_z$,$h, w, Rot$) does the same as randvec_target_circle but targetting at a rectangle with angular dimensions $h$ and $w$ (in **radians**, not in degrees as other angles). The rotation matrix *Rot* is the coordinate system orientation in the absolute frame, usually ROT_A_CURRENT_COMP.

- **randvec_target_rect**(&$v_x$, &$v_y$, &$v_z$, &$d\Omega$, aim$_x$, aim$_y$, aim$_z$,$height, width, Rot$) is the same as randvec_target_rect_angular but *height* and *width* dimensions are given in meters. This function is useful to e.g. target at a guide entry window or analyzer blade.

## A.2. Reading a data file into a vector/matrix (Table input, `read_table-lib`)

The `read_table-lib` library provides functionalities for reading text (and binary) data files. To use this library, add a `%include "read_table-lib"` in your component definition DECLARE or SHARE section. Tables are structures of type `t_Table` (see `read_table-lib.h` file for details):

```
1  /* t_Table structure (most important members) */
2  double *data;     /* Use Table_Index(Table, i j) to extract [i,j]
       element */
3  long    rows;     /* number of rows */
4  long    columns;  /* number of columns */
5  char   *header;   /* the header with comments */
6  char   *filename; /* file name or title */
7  double  min_x;    /* minimum value of 1st column/vector */
8  double  max_x;    /* maximum value of 1st column/vector */
```

Available functions to read *a single* vector/matrix are:

- **Table_Init**(&*Table*, *rows*, *columns*) returns an allocated Table structure. Use $rows = columns = 0$ not to allocate memory and return an empty table. Calls to Table_Init are *optional*, since initialization is being performed by other functions already.

- **Table_Read**(&*Table*, *filename*, *block*) reads numerical block number *block* (0 to catenate all) data from *text* file *filename* into *Table*, which is as well initialized in the process. The block number changes when the numerical data changes its size, or a comment is encoutered (lines starting by '`# ; % /`'). If the data could not be read, then *Table.data* is NULL and $Table.rows = 0$. You may then try to read it using Table_Read_Offset_Binary. Return value is the number of elements read.

- **Table_Read_Offset**(&*Table*, *filename*, *block*, &*offset*, $n_{rows}$) does the same as Table_Read except that it starts at offset *offset* (0 means begining of file) and reads $n_{rows}$ lines (0 for all). The *offset* is returned as the final offset reached after reading the $n_{rows}$ lines.

- **Table_Read_Offset_Binary**(&*Table*, *filename*, *type*, *block*, &*offset*, $n_{rows}$, $n_{columns}$) does the same as Table_Read_Offset, but also specifies the *type* of the file (may be "float" or "double"), the number $n_{rows}$ of rows to read, each of them having $n_{columns}$ elements. No text header should be present in the file.

- **Table_Rebin**(&*Table*) rebins all *Table* rows with increasing, evenly spaced first column (index 0), e.g. before using Table_Value. Linear interpolation is performed for all other columns. The number of bins for the rebinned table is determined from the smallest first column step.

- **Table_Info**(*Table*) print information about the table *Table*.

- **Table_Index**(*Table*, $m$, $n$) reads the *Table*$[m][n]$ element.

- **Table_Value**(*Table*, $x$, $n$) looks for the closest $x$ value in the first column (index 0), and extracts in this row the $n$-th element (starting from 0). The first column is thus the 'x' axis for the data.

- **Table_Free**(&*Table*) free allocated memory blocks.

- **Table_Value2d**(*Table*, $X$, $Y$) Uses 2D linear interpolation on a Table, from (X,Y) coordinates and returns the corresponding value.

Available functions to read *an array* of vectors/matrices in a *text* file are:

- **Table_Read_Array**(*File*, &$n$) read and split *file* into as many blocks as necessary and return a `t_Table` array. Each block contains a single vector/matrix. This only works for text files. The number of blocks is put into $n$.

- **Table_Free_Array**(&*Table*) free the *Table* array.

- **Table_Info_Array**(&*Table*) display information about all data blocks.

The format of text files is free. Lines starting by '`# ; % /`' characters are considered to be comments, and stored in *Table.header*. Data blocks are vectors and matrices. Block numbers are counted starting from 1, and changing when a comment is found, or the column number changes. For instance, the file 'MCXTRACE/data/Rh.txt' (Material data for Rhodium) looks like:

```
#Rh (Z  45)
#Atomic weight: A[r]  102.9055
#Nominal density: rho 1.2390E+01
#    sigma[a](barns/atom) = [mu/rho](cm\^2 g\^-1)  .  1.70879E+02
#    E(eV) [mu/rho](cm\^2 g\^-1) = f[2](e atom\^-1)  .  4.08922E+05
#    14 edges. Edge energies (keV):
```

```
7 #
8 #
9 #       K       2.32199E+01   L I     3.41190E+00   L II    3.14610E+00   L III
      3.00380E+00
10 #      M I     6.27100E-01   M II    5.21000E-01   M III   4.96200E-01   M IV
      3.11700E-01
11 #      M V     3.07000E-01   N I     8.10000E-02   N II    4.79000E-02   N III
      4.79000E-02
12 #      N IV    2.50000E-03   N V     2.50000E-03
13 #
14 #     Relativistic correction estimate f[rel] (H82,3/5CL) = -4.0814E-01,
15 #     -2.5440E-01 e atom\^-1
16 #     Nuclear Thomson correction f[NT] = -1.0795E-02 e atom\^-1
17 #
18 #
   ------------------------------------------------------------------------------


19 #Form Factors, Attenuation and Scattering Cross-sections
20 #Z=45, E = 0.001 - 433 keV
21 #
22 #       E               f[1]            f[2]          [mu/rho]        [sigma/rho]
          [mu/rho]        [mu/rho][K]       lambda
23 #                                                    Photoelectric Coh+inc       Total
24 #      keV          e atom\^-1       e atom\^-1    cm\^2 g\^-1        cm\^2 g\^-1
           cm\^2 g\^-1    cm\^2 g\^-1      nm
25 1.069000E-02   1.89417E+00   4.8055E+00   1.8382E+05   1.1514E-04   1.8382E+05
     0.000E+00   1.160E+02
26 1.142761E-02   2.09662E+00   5.1028E+00   1.8260E+05   1.5865E-04   1.8260E+05
     0.000E+00   1.085E+02
27 1.221612E-02   2.32705E+00   5.4019E+00   1.8082E+05   2.1741E-04   1.8082E+05
     0.000E+00   1.015E+02
28 1.305903E-02   2.58575E+00   5.6998E+00   1.7848E+05   2.9628E-04   1.7848E+05
     0.000E+00   9.494E+01
29 1.396010E-02   2.87263E+00   5.9931E+00   1.7555E+05   4.0158E-04   1.7555E+05
     0.000E+00   8.881E+01
30 1.492335E-02   3.18714E+00   6.2786E+00   1.7204E+05   5.4136E-04   1.7204E+05
     0.000E+00   8.308E+01
31 1.595306E-02   3.52819E+00   6.5531E+00   1.6797E+05   7.2588E-04   1.6797E+05
     0.000E+00   7.772E+01
32 1.705382E-02   3.89415E+00   6.8134E+00   1.6337E+05   9.6809E-04   1.6337E+05
     0.000E+00   7.270E+01
33   ...
```

Binary files should be of type "float" (i.e. REAL*32) and "double" (i.e. REAL*64), and should *not* contain text header lines. These files are platform dependent (little or big endian).

The *filename* is first searched into the current directory (and all user additional locations specified using the `-I` option, see the 'Running McXtrace ' chapter in the User Manual), and if not found, in the `data` sub-directory of the `MCXTRACE` library location. This way, you do not need to have local copies of the McXtrace Library Data files (see table 1.1).

A usage example for this library part may be:

```
t_Table Table;        // declare a t_Table structure
char file[]="Rh.txt";  // a file name
double x,y;

Table_Read(&Table, file, 1);  // initialize and read the first numerical
    block
Table_Info(Table);            // display table informations
...
x = Table_Index(Table, 2,5);  // read the 3rd row, 6th column element
                              // of the table. Indexes start at zero in C
                                 .
y = Table_Value(Table, 1.45,1);  // look for value 1.45 in 1st column (x
    axis)
                              // and extract 2nd column value of that row
Table_Free(&Table);           // free allocated memory for table
```

Additionally, if the block number (3rd) argument of **Table_Read** is 0, all blocks will be catenated. The **Table_Value** function assumes that the 'x' axis is the first column (index 0). Other functions are used the same way with a few additional parameters, e.g. specifying an offset for reading files, or reading binary data.

This other example for text files shows how to read many data blocks:

```
t_Table *Table;       // declare a t_Table structure array
long     n;
double y;

Table = Table_Read_Array("file.dat", &n); // initialize and read the all
    numerical block
n = Table_Info_Array(Table);      // display informations for all blocks (
    also returns n)

y = Table_Index(Table[0], 2,5);  // read in 1st block the 3rd row, 6th
    column element
                              // ONLY use Table[i] with i < n !
Table_Free_Array(Table);          // free allocated memory for Table
```

You may look into, for instance, the source files for **Lens_parab** or **Filter** for other implementation examples.

## A.3. Constants for unit conversion etc.

The following predefined constants are useful for conversion between units

| Name | Value | Conversion from | Conversion to |
|---|---|---|---|
| **DEG2RAD** | $2\pi/360$ | Degrees | Radians |
| **RAD2DEG** | $360/(2\pi)$ | Radians | Degrees |
| **MIN2RAD** | $2\pi/(360 \cdot 60)$ | Minutes of arc | Radians |
| **RAD2MIN** | $(360 \cdot 60)/(2\pi)$ | Radians | Minutes of arc |
| **FWHM2RMS** | $1/\sqrt{8\log(2)}$ | Full width half maximum | Root mean square (standard deviation) |
| **RMS2FWHM** | $\sqrt{8\log(2)}$ | Root mean square (standard deviation) | Full width half maximum |
| **MNEUTRON** | $1.67492 \cdot 10^{-27}$ kg | Neutron mass, $m_{\mathrm{n}}$ | |
| **HBAR** | $1.05459 \cdot 10^{-34}$ Js | Planck constant, $\hbar$ | |
| **PI** | 3.14159265... | $\pi$ | |
| **CELE** | 1.602176487e-19 | Elementary charge (C) | |
| **M_C** | 299792458 | Speed of light in vacuum (m/s) | |
| **NA** | 6.02214179e23 | Avogadro's number (#atoms/g·mole) | |
| **RE** | 2.8179402894e-5 | Thomson scattering length (AA) | |
| **E2K** | 0.506773091264796 | Wavenumber (1/AA) | Energy (keV) |
| **K2E** | 1.97326972808327 | Energy (keV) | Wavenumber (1/AA) |

# B. The McXtrace terminology

This is a short explanation of phrases and terms which have a specific meaning within McXtrace. We have tried to keep the list as short as possible running the calculated risk that the reader may occasionally miss an explanation. In this case, you are more than welcome to contact the McXtrace core team.

- **Arm** A generic McXtrace component which defines a frame of reference for other components.
- **Component** One unit (*e.g.* optical element) in an x-ray beamline. These are considered as Types of elements to be instantiated in an Instrument description.
- **Component Instance** A named Component (of a given Type) inserted in an Instrument description.
- **Definition parameter** An input parameter for a component. For example the radius of a sample component or the divergence of a collimator. Technically, a definition parameter is translated into a literal constant, which prevents it from being edited at runtime.
- **Input parameter** For a component, either a definition parameter or a setting parameter. These parameters are supplied by the user to define the characteristics of the particular instance of the component definition. For an instrument, a parameter that can be changed at simulation run-time.
- **Instrument** An assembly of McXtrace components defining an x-ray beamline.
- **Kernel** The McXtrace language definition and the associated compiler
- **Output parameter** An output parameter for a component. For example the counts in a monitor. An output parameter may be accessed from the instrument in which the component is used using `MC_GETPAR`.
- **Run-time** C code, contained in the files `mcxtrace-r.c` and `mcxtrace-r.h` included in the McXtrace distribution, that declare functions and variables used by the generated simulations.
- **Setting parameter** Similar to a definition parameter, but with the restriction that the type of the parameter must be declared unless it is a number. In technical terms, a setting parameter is translated into an actual variable (as opposed to a definition parameter) which may be dynamically updated.

# Bibliography


[Mcx] See http://www.mcxtrace.org (cit. on pp. 8, 11).

[Tra] See http://trac.edgewall.com (cit. on p. 8).

[Nis] See http://www.nist.gov/pml/data/ffast/index.cfm (cit. on pp. 10, 28, 33).

[Mcz] See http://trac.mccode.org/report (cit. on p. 11).

[Sha] See http://www.esrf.eu/computing/scientific/raytracing/ (cit. on p. 12).

[ANM11] J Als-Nielsen and D McMorrow. *Elements of modern X-ray physics*. Wiley, 2011 (cit. on p. 18).

[Bac75] G.E. Bacon. *Neutron Diffraction*. Oxford University Press, 1975 (cit. on pp. 39, 44).

[Bau+07] Sondes Trabelsi Bauer et al. "Simulation of X-ray beamlines with the new ray tracing tool¡ i¿ X¡/i¿ Trace". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 582.1 (2007), pp. 90–92 (cit. on p. 12).

[Bea67] J A Bearden. "X-ray wavelengths". In: *Reviews of Modern Physics* 39.1 (1967), p. 78 (cit. on pp. 22, 24).

[BK+13] Erik Bergbäck Knudsen et al. "McXtrace". English. In: *Journal of Applied Crystallography* 46.3 (2013), pp. 679–696. ISSN: 00218898, 16005767. DOI: 10.1107/S0021889813007991 (cit. on p. 27).

[GRR92] Grimmett, G. R., and Stirzakerand D. R. *Probability and Random Processes, 2nd Edition*. Clarendon Press, Oxford, 1992 (cit. on p. 13).

[Jam80] F. James. In: *Rep. Prog. Phys.* 43 (1980), p. 1145 (cit. on pp. 12, 13, 17).

[Knu+14] E. B. Knudsen et al. *Users' and Programmers' Guide to the X-ray tracing Package McXtrace, version 1.2*. DTU Physics, 2014 (cit. on pp. 8, 11, 38).

[Kra23] H. A. Kramers. "XCIII. On the theory of X-ray absorption and of the continuous X-ray spectrum". In: *Philosophical Magazine Series 6* 46.275 (Nov. 1923), pp. 836–871. ISSN: 1941-5982. DOI: 10.1080/14786442308565244 (cit. on p. 22).

[KO79] MO Krause and JH Oliver. "Natural widths of atomic K and L levels, Ka X-ray lines and several KLL Auger lines". In: *J. Phys. Chem. Ref. Data* 8 (1979), pp. 329–338 (cit. on pp. 22, 24).

[LN99] K. Lefmann and K. Nielsen. "McStas, a general software package for neutron ray-tracing simulations". In: *Neutron News* 10 (1999), pp. 20–23. DOI: 10.1080/10448639908233684 (cit. on p. 8).

[MN98] Makoto Matsumoto and Takuji Nishimura. “Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator”. In: *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 8.1 (1998), pp. 3–30 (cit. on p. 69).

[Pre+86] W. H. Press et al. *Numerical Recipes in C*. Cambridge University Press, 1986 (cit. on p. 69).

[Rio+11] M Sanchez del Rio et al. “SHADOW3: a new version of the synchrotron X-ray optics modelling package”. In: *Journal of Synchrotron Radiation* 18.5 (2011), p. 0 (cit. on pp. 12, 62).

[Sch08] F Schäfers. “The BESSY raytrace program RAY”. In: *Modern Developments in X-Ray and Neutron Optics* (2008), pp. 9–41 (cit. on p. 12).

[SKS96] A Snigirev, V Kohn, and I Snigireva. “A compound refractive lens for focusing high-energy X-rays”. In: *Nature* (1996) (cit. on p. 30).

[Squ78] G.L. Squires. *Thermal Neutron Scattering*. Cambridge University Press, 1978 (cit. on p. 45).

[TK01] Takashi Tanaka and Hideo Kitamura. “*SPECTRA*: a synchrotron radiation calculation code”. In: *Journal of Synchrotron Radiation* 8.6 (2001), pp. 1221–1228. DOI: 10.1107/S090904950101425X (cit. on p. 19).

[WCC94] C Welnak, G J Chen, and F Cerrina. “SHADOW: A synchrotron radiation and X-ray optics simulation tool”. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 347.1-3 (1994), pp. 344–347 (cit. on p. 12).

# Index