# Sassena Config File (=scatter.xml)

| | |
|---|---|
| Sassena Software Version | v1.4.0 |
| Document Revision: | 4 |
| Document Date: | 1/25/12 |
| Document Authors: | Benjamin Lindner (ben@benlabs.net) |

**Abstract**

The following document contains details and a reference of the sassena configuration file, which is required to perform scattering calculations on molecular dynamics data.

## Contents

**1 Introduction** **3**

**2 Content** **3**

2.1 Sample . . . . . . . . . . 4
2.1.1 Structure . . . . . . . . . . 5
2.1.2 Framesets . . . . . . . . . . 5
2.1.3 Selections . . . . . . . . . . 6
2.1.4 Motions . . . . . . . . . . 9
2.1.5 Alignments . . . . . . . . . . 12
2.2 Stager . . . . . . . . . . 14
2.2.1 Target . . . . . . . . . . 14
2.2.2 Mode . . . . . . . . . . 14
2.2.3 Dumping . . . . . . . . . . 14
2.3 Scattering . . . . . . . . . . 15
2.3.1 Target . . . . . . . . . . 15
2.3.2 Type . . . . . . . . . . 15
2.3.3 Vectors . . . . . . . . . . 16
2.3.4 Average . . . . . . . . . . 18
2.3.5 DSP . . . . . . . . . . 20
2.3.6 Signal . . . . . . . . . . 21
2.3.7 Background . . . . . . . . . . 21
2.4 Limits . . . . . . . . . . 22
2.4.1 Stage . . . . . . . . . . 23
2.4.2 Signal . . . . . . . . . . 23

2.4.3 Computation . . . . . . . . . . 24
2.4.4 Services . . . . . . . . . . 25
2.4.5 Decomposition . . . . . . . . . . 26
2.5 Debug . . . . . . . . . . 27

**3 Reference** **27**
3.1 Sample . . . . . . . . . . 27
3.1.1 Structure . . . . . . . . . . 27
3.1.2 Framesets . . . . . . . . . . 28
3.1.3 Selections . . . . . . . . . . 28
3.1.4 Alignments . . . . . . . . . . 29
3.1.5 Motions . . . . . . . . . . 29
3.2 Stager . . . . . . . . . . 30
3.3 Scattering . . . . . . . . . . 30
3.3.1 DSP . . . . . . . . . . 30
3.3.2 Average . . . . . . . . . . 30
3.3.3 Vectors . . . . . . . . . . 31
3.3.4 Signal . . . . . . . . . . 32
3.3.5 Background . . . . . . . . . . 32
3.4 Limits . . . . . . . . . . 32
3.4.1 Stage . . . . . . . . . . 32
3.4.2 Signal . . . . . . . . . . 33
3.4.3 Computation . . . . . . . . . . 33
3.4.4 Services . . . . . . . . . . 33
3.5 Debug . . . . . . . . . . 33

## 1 Introduction

The sassena configuration file enables the user to set mandatory and optional parameter which are used during execution of the software. Each optional configuration parameter is supplied with a default value. The configuration file is based on the XML format. The configuration parameters are organized into a tree hierarchy which maps a class inheritance diagram.

The configuration file is parsed for appearances of certain key sections. Sections which do not map to a valid entry are currently ignored. *The user should be aware that misspellings of sections which are not mandatory may result in those sections to exercise their default behavior.*

Some entries trigger the parsing of other sections. If that is the case, the user has to make sure that these sections are properly defined.

## 2 Content

The configuration file is structured into 5 main sections.

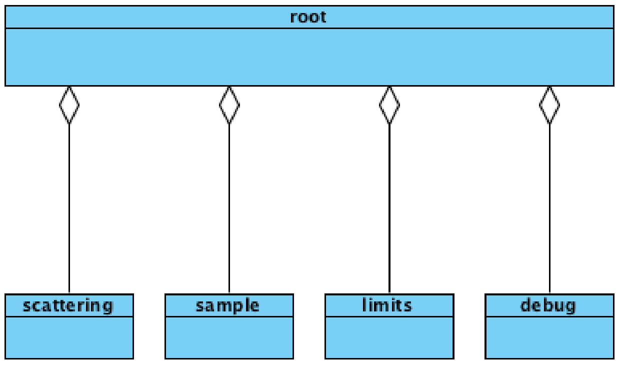

Figure 1: Main Layout for the Configuration File ( section stager was added previously)

The four different sections have the following distinct features:

**sample** contains sections which modify the available data. This includes coordinates and selection names.

**stager** contains sections which determines the staging mode of the trajectory data.

**scattering** contains sections which modify the retrieved signal, but don't affect the available data.

**limits** contains sections which neither modify the retrieved signal, nor the available data, but impacts the performance and computational aspects on how the calculation is carried out.

**debug** contains sections which provide control switches and debug information for different aspects of the software. This section is intended to allow debugging without the need for recompilation. It may affect available data, the retrieved signal and the achieved performance

Some sections may contain multiple subsections with identical names. In this case the order by which they appear in the file is preserved.

## 2.1 Sample

The sample section contains section which affect the available data.

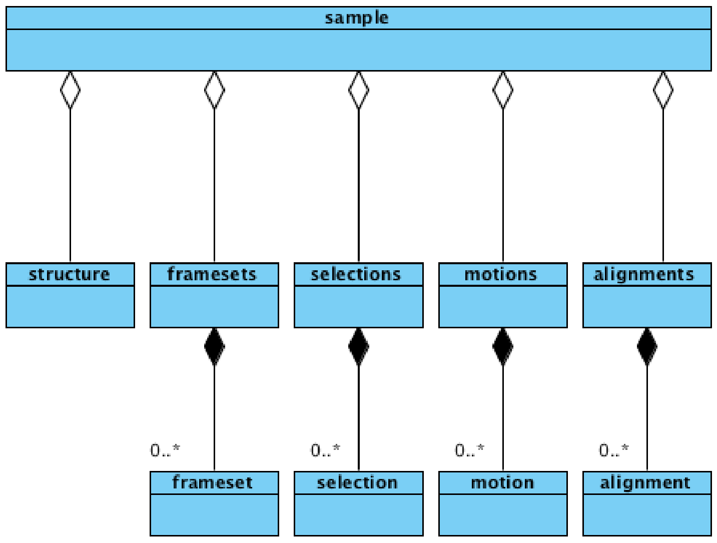

Figure 2: Layout of the sample section

The subsections have the following meaning:

**structure** defines the structural composition of the sample.

**framesets** defines the set of time dependent coordinate files matching the structure

**selections** defines names for sub groups of atoms within the structure

**motions** defines artificial motions which can be applied to groups of atoms

**alignments** defines alignment procedure for groups of atoms which can be applied before or after any artificial motions

### 2.1.1 Structure

This section contains elements which identify the structure, i.e. the atomic composition of the sample. The content currently has to be supplied as a file with PDB format. Only the ATOM entries of the file are evaluated, other lines are skipped.

EXAMPLE

```
<structure>
        <file>structure.pdb</file>
        <format>pdb</format>
</structure>
```

reads structure information from pdb file "structure.pdb".

### 2.1.2 Framesets

This section contains an arbitrary number of *frameset* elements. Each frameset element connect to exactly one file (which can contain an arbirtrary number of frames). The frameset contains elements (first,last,stride) which allow the extraction of only a portion of the contained data. The clones element specifies the number of times, the current frameset is appended to the internal list of framesets. This, in combination with the section motions, allows to generate a virtual trajectory (i.e. trajectory data which does not exist on disk). The portion specifier elements stride, first and last, can be definied within the framesets section. If done so, they will be regarded as default values for each defined frameset element.

EXAMPLE

```
<framesets>
```

```
 <frameset>
  <file>data.dcd</file>
  <format>dcd</format>
  <stride>2</stride>
  <first>10</first>
  <last>100</last>
 </frameset>
</framesets>
```

selects frames from a DCD file “data.dcd”, first frame included is number 11 (indexes start at 0), second frame included is 13 (index 10+2),... no frame with index above 100 is selected.

### 2.1.3 Selections

This section contains an arbitrary number of *selection* elements. Each selection element itself makes names for groups of atoms available. There are a number of different ways to selection subgroups of atoms. The type of a selection is specified by the type element. The type then triggers the parsing of other entries which may be defined. The selection elements parse the name element to assign labels to the selections, which can be used in other places to reference the given subgroup of atoms. If a name is not specified, the generated default name (“_#number”) is used as an identifier. If the name can be determined by other means (e.g. type file, format ndx), then those names will be used instead.

**Index** A selection of type index is the most simplistic one. It scans for index elements and adds them to the named selection. The first atom is identified by index 0. The last atom is identified by N-1, where N stands for the number of atoms.

EXAMPLE

```
<selections>
 <selection>
  <name>FourAtoms</name>
  <type>index</type>
  <index>0</index>
  <index>10</index>
  <index>11</index>
  <index>15</index>
 </selection>
</selections>
```

creates a selection with name “FourAtoms”, which contains 4 atoms with indexes 0,10,11 and 15.

**Range** A ranged selection allows to select a continous group of atoms. This type of selection is memory efficient, since it only requires the storage of two indexes.

EXAMPLE

```
<selections>
 <selection>
  <name>TenAtoms</name>
  <type>range</type>
  <from>0</from>
  <to>9</to>
</selection>
</selections>
```

creates a selection with name “TenAtoms”, which contains 10 atoms with indexes between 0 and 9 (inclusive).

**Lexical** A lexical selection allows to select atoms by their internal label, which is determined by the database. The element expression is used as a regular expression. All rules regarding regular expressions apply here (e.g. use “h.*” instead of “h*” to select all atoms which start with h).

EXAMPLE

```
<selections>
 <selection>
  <name>AllHydrogen</name>
  <type>lexical</type>
  <expression>hydro.*</expression>
 </selection>
</selections>
```

creates a selection with name “AllHydrogen”, which contains all atoms with labels starting with “hydro”.

**File** A File selection allows the definition of more complex selection type and/ore provides a more convenient way of introducing selections. Two types of formats are currently supported: PDB and NDX. The pdb selection strategy is similar to the way NAMD allows selections. The NDX format stands for gromacs style index files. Every selection of type file allows the specficiation of the selector and expression element. The selector element is used in combination with the format to indentify which part in the supplied file contains the selection criterium. The expression then is evaluated as a regular expression to identify positives matches. Only those entries which yield a positive match are added to the internal list of selections.

First example

```
<selections>
 <selection>
  <name>AtomFromPDB</name>
  <type>file</type>
  <file>selection.pdb</file>
  <format>pdb</format>
  <selector>beta</selector>
  <expression>2|2\.0|2\.00</expression>
 </selection>
</selections>
```

creates a selection with name "AtomFromPDB", which contains all atoms which have a value in the BETA column matching the regular expression testing for the number 2.

second example

```
<selections>
 <selection>
  <name>AtomFromNDX</name>
  <type>file</type>
  <file>index.ndx</file>
  <format>ndx</format>
  <selector>name</selector>
  <expression>AGLC.*</expression>
 </selection>
</selections>
```

IMPORTS selections from file "index.ndx" (name "AtomFromNDX" is NOT used!). Only selections with names starting with "AGLC" will be imported.

### 2.1.4 Motions

This section contains an arbitrary number of *motion* elements. Each motion element itself applies a type of motion to the reference selection of atoms. Various types of motions are available. Motions are applied in the same order as they are defined in the configuration file. Currently only translational types of motions are supported. Each motion adds a time-dependent (exception: fixed) position vector $\vec{r}(t)$ to each selected atom. The time $t$ is given in units of frames. Because the user is allowed to change the time unit easily by applying a stride factor within the frameset section, each motion allows the definition of an integer *sampling* element. which should match the stride factor used. This guarantees that the same displacement is added to the selected frames. The direction of motion is normalized is any case. For instance, a linear motion with a displacement $\delta = 5$ along $\vec{d} = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$ would yield a position vector of $\vec{r} = \frac{5}{\sqrt{3}} \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$.

**Fixed** This adds a constant position vector $\vec{r}$ to each selected atom.

EXAMPLE

```
<motions>
 <motion>
  <type>fixed</type>
  <selection>MotionAtoms</selection>
  <displace>5</displace>
  <direction>
   <x>1</x>
   <y>1</y>
   <z>1</z>
  </direction>
</motion>
</motions>
```

moves atoms within the selection with name "MotionAtoms" by $\vec{r}(t) = \frac{5}{\sqrt{3}} \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$

**Linear** The position vector for linear types of motions is given by $\vec{r}(t) = \delta \cdot \vec{d} \cdot s \cdot t$. The displacement $\delta$ is given in time units of frames. The direction $\vec{d}$ specifies the normalized base vector. The sampling factor s allows correction related to the usage of strides. The time $t$ is given in units of frames.

EXAMPLE

```
<motions>
 <motion>
  <type>linear </type>
  <selection >MotionAtoms</selection >
  <displace >5</displace >
  <direction >
   <x>1</x>
   <y>1</y>
   <z>1</z>
  </direction >
</motion>
</motions>
```

moves atoms within the selection with name “MotionAtoms” by $\vec{r}(t) = \frac{5}{\sqrt{3}} \begin{pmatrix} 1 & 1 & 1 \end{pmatrix} \cdot t$

**Oscillation** The position vector for oscillations is given by $\vec{r}(t) = \delta \cdot \vec{d} \cdot \sin(2 \cdot \pi \cdot f \cdot s \cdot t)$, where the frequency$f$ is given in units of per frame. The other symbols are similar to the linear type of motion.

EXAMPLE

```
<motions>
 <motion>
  <type>oscillation </type>
  <selection >MotionAtoms</selection >
  <displace >5</displace >
  <seed >11</seed >
</motion>
</motions>
```

moves atoms within the selection with name “MotionAtoms” by $\vec{r}(t) = \frac{5}{\sqrt{3}} \begin{pmatrix} 1 & 1 & 1 \end{pmatrix} \cdot \sin(2 \cdot \pi \cdot 0.01 \cdot t)$

**Random** The position vector for random motion is given by $\vec{r}(t) = \sum_0^t \vec{\delta}(t)$, which corresponds to a sum of individual increments $\vec{\delta}$. The increments are generated from a uniform distribution on a 3 dimensional sphere $P_{Sphere}$. To guarantee that the position vector for two different times incorporate the same increments for the overlapping time region, the complete time series of the random motion is computed once and held in computer memory in form of a motion trajectory. A manual defintion of a seed value allows the user to reproduce a particular random motion and to generate different ones, based on needs. The sampling factor is supported and selects every $s$ random direction. The increments are computed corresponding to $\vec{\delta}(t) = \delta \cdot \vec{d}_{Sphere}(seed, s \cdot t)$.

EXAMPLE

```
<motions>
 <motion>
  <type>randomwalk</type>
  <selection>MotionAtoms</selection>
  <displace>5</displace>
  <seed>11</seed>
</motions>
</motion>
```

moves atoms in selection with name "MotionAtoms" by $\vec{r}(t) = \sum_0^t 5 \cdot \vec{\delta}(t)$, with $\vec{\delta}$drawn from uniform distrution on a sphere (normalized vector). The random generator is initalized with 11.

**Brownian** The position vector for brownian motion is similar to random motion with the exception that the displacement $\delta$ is generated from a gaussian distribution. The corresponding increments are therefore $\vec{\delta}(t) = \delta_{gaussian}(seed, s \cdot t) \cdot \vec{d}_{Sphere}(seed, s \cdot t)$.

EXAMPLE

```
<motions>
 <motion>
  <type>brownian</type>
  <selection>MotionAtoms</selection>
  <displace>5</displace>
  <seed>11</seed>
</motion>
</motions>
```

moves atoms within the selection with name "MotionAtoms" by $\vec{r}(t) = \sum_0^t 5 \cdot \vec{\delta}(t)$, with $\vec{\delta}(t) = \delta_{gaussian} \cdot \vec{d}_{Sphere}$. The gaussian distribution is seed with 11, the spherical distribution by 12.

**LocalBrownian** The position vector for localized brownian motion is similar to brownian motion with the execption that the total displacement $\vec{r}(t)$ is restricted to a value smaller than the radius $R$. The implementation currently guarantees this by dropping increments which result in the total displacement to grow beyond $R$. Care must be taken when using stride factors, since the combination of a restriction criterium with a sampling factor might break the correct assignment of random displacements to the original coordinates.

EXAMPLE

```
<motions>
 <motion>
  <type>fixed </type>
  <selection >MotionAtoms</selection >
  <displace >5</displace >
  <seed >11</seed >
  <radius >15</radius >
</motion>
</motions>
```

moves atoms within the selection with name "MotionAtoms" identical to the example with brownian motion, but with the additional constrain that $||\vec{r}(t)|| \leq R$ .

### 2.1.5 Alignments

Alignments can be applied before (pre) or after (post) the addition of artifical motions.

**Center** Center alignment computes the center of mass for the given selection of atoms and moves that center into the origin. The alignment vector is given by $\vec{d}(t) = \sum_n m_n \cdot r_n(t)/M$, where $M = \sum_n m_n$ is the total mass of the selection of atoms. The position of each selected atom is then determined by $\vec{r}^*_n(t) = \vec{r}_n(t) - \vec{d}(t)$.

EXAMPLE

```
<alignments>
 <alignment>
  <type>center </type>
  <selection >AlignedAtoms</selection >
  <order>pre</order>
```

```
</alignment>
</alignments>
```

moves atoms within the selection with name “AlignedAtoms” by the center of mass distance *before* any artificial motion is added.

**Fittrans** Fittrans alignment works like center, but moves the center of mass for each frame to the center of mass point of a reference frame. The reference frame can be either supplied by a seperate coordinate file, or is given a a specified frame number within the trajectory.

**Fitrottrans** Fitrottrans alignment does least square fitting as practiced by many other software packages as well. It first performs the same operation as the Fittrans procedure, but then also removes the rotation of the system with respect to a reference frame.

EXAMPLE

```
<alignments>
 <alignment>
  <type>fitrottrans </type>
  <selection>AlignedAtoms</selection>
  <order>pre</order>
  <reference>
   <type>frame</type>
   <frame>0</frame>
   <selection>AlignedAtoms</selection>
  </reference>
</alignment>
</alignments>
```

rotates and translates atoms within the selection with name “AlignedAtoms” to match the position and orientation of the atoms within the first frame of the trajectory. Effectivly reduces the dynamics to internal motion.

**Fitrot** Fitrot alignment does the same as the Fitrottrans operation, but moves the center of mass back to its original position for each frame. This procedure removes any rotation of the system, but preserves the translational motion.

## 2.2 Stager

This section mainly effects the staging procedure. However, the scattering type (all/self) may enforce a specific staging mode. The seperation of the staging mode into its own section allows future extension of the software towards other uses and allows the use of the data staging modes without analysis for distinct purpose (the tool s_stager simply stages data without analysis), e.g. parallel reading and processing of the trajectory data with a subsequent parallel write, or the inversion of the trajectory data layout (atoms <-> frames).

### 2.2.1 Target

This section allows the definition of a target selection. This enables the user to compute the scattering or perform an analysis on a subgroup of atoms without the need of producing a reduced trajectory. The default is to include all atoms (system). The selection has to be defined within the section sample.selections.

EXAMPLE

```
<target>SelectedAtoms</target>
```

declares that only atoms of the selection with name “SelectedAtoms” are considered when computing the scattering diagram.

### 2.2.2 Mode

Mode can be either “atoms” or “frames”. The mode is usually enforced by the specific analysis (thus overwritten). However, data processing which only operates on the trajectory data (like the tool s_stager), requires the specification of the proper staging mode. Mode “frames” distributes complete frames among the available nodes, while “atoms” assigns atoms to nodes.

EXAMPLE

```
<mode>frames</mode>
```

declares that the data will be staged by frames, UNLESS the analysis enforces a specific staging mode.

### 2.2.3 Dumping

The in-memory trajectory data can be written to a new file. This allows the use of the sassena package to extract and post-process trajectory data efficiently. If the target only specified a sub-selection of atoms in the system, then the written trajectory only contains those.

EXAMPLE

```
<dump>true</dump>
<file>dump.dcd</file>
<format>dcd</format>
```

writes the in-memory trajectory data to the file dump.dcd in DCD format. If the staging mode is “atoms” the trajectory is in effect transposed, which means that the first frame of the new trajectory contains the positions of the first atom, the second frame contains all positions of the second atom, and so on.

## 2.3 Scattering

The section contains parameters which affect the resulting scattering signal.

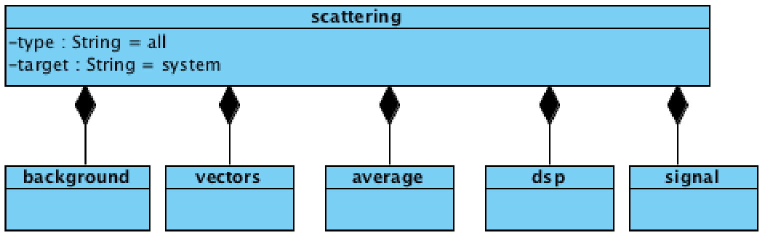

Figure 3: Layout of the scattering section

### 2.3.1 Target

obsolete! has been moved to the stager section.

### 2.3.2 Type

Two types of scattering functions are currently supported: Coherent (all) and Incoherent (self) scattering. The calculation schemes for the two types of scattering are fundamentally different.

EXAMPLE

```
<type>all</type>
```

declares that the computed scattering diagram represents coherent scattering.

### 2.3.3 Vectors

The scattering diagram contains scattering intensities as a function of the direction of observation (q vector). This section allows the defintion of q vectors which are used to compute the scattering diagram. The vectors type defines how the q vectors are generated/supplied.

**Single** When using the type single, the scattering diagram contains only one q vector.

EXAMPLE

```
<vectors>
  <type>single</type>
  <x>1</x>
  <y>0</y>
  <z>0</z>
</vectors>
```

defines the q vector to be $\vec{q} = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}$

**Scans** The scans type currently allows the definition of up to three scan elements, which provide ranges of q vectors. The different scan elements are combined to yield multi-dimensional scans. This may yield a large number of q vectors. For instance, when defining three scan elements along the x, y and z axis, respectively, with 100 points each, the total number of q vectors will be 1000000. Additionally, the exponent element allows the generation of non-uniform q vector ranges. This is helpful in cases where some q regions have to be more densly sampled than others. Q vectors are generated from a range by setting the first and last point to the supplied from and last. Other q vector values are determined based on their point assignment $i$: $q_i = (\frac{i}{N})^E \cdot (q_{to} - q_{from})$. The direction of each q vector is determined by the supplied base vectors (normalized).

FIRST EXAMPLE

```
<vectors>
 <type>scans</type>
 <scans>
  <scan>
  <x>1</x><y>0</y><z>0</z>
  <from>-2</from>
  <to>2</to>
  <points>50</points>
  </scan>
  <scan>
   <x>0</x><y>1</y><z>0</z>
  <from>-2</from>
  <to>2</to>
  <points>50</points>
  </scan>
 </scans>
 </vectors>
```

creates a list of q vectors, corresponding to a scattering diagram in the xy plane. The resulting diagram has a 50 pixel resolution in each direction with 49 steps of 4/49 from -2 to 2.

### Second example

```
<vectors>
 <type>scans</type>
 <scans>
  <scan>
  <x>1</x><y>0</y><z>0</z>
  <from>0.001</from>
  <to>1</to>
  <points>100</points>
  <exponent>3</exponent>
  </scan>
 </scans>
 </vectors>
```

creates a list of q vectors, corresponding to a scattering diagram in the along the x axis. The resulting diagram has a 100 pixel resolution 99 steps. The step size is non-linear with an exponent of 3.

**File** The file type allows to read q vectors from a source text file ( line-by-line, whitespace delimited). This way the user can use their own algorithms to generate complex sets of q vectors.

Second example

```
<vectors>
 <type>file</type>
 <file>qvectors.txt</file>
 </vectors>
```

creates 5 q vectors based on the contents in file “qvectors.txt”.

Contents of “qvectors.txt”

```
1 0 0
1 1 1
1 0 1
0 1 0
0.3 0.3 0.1
```

### 2.3.4 Average

This section defines the type of averaging procedures which are applied in-place. Currently only orientational averaging is supported. There are two types of orientational averaging procedures (Monte Carlo, Multipole). The Monte Carlo scheme performs oriental averaging by recomputing and integrating the scattering signal for a set of random directions. This corresponds to stochastic integration, which is generally good when the intensities do not vary signficantly. For highly crystalline samples, which feature strong Bragg peaks, a large number of vectors may be necessary to reach convergence ( O(5)-O(6) ). The Multipole scheme employs a multipole expansion of the exponential terms involved in the scattering calculation. It performs superior for low q values ( q<<1 ). At large q values, multipole moments of high order become dominant, which requires to increase resolution incrementally.

**Vectors** The vectors type triggers the Monte Carlo scheme for orientational averaging and the parsing of the vectors sections in scattering.average.orientations. The q vector orientations are determined in one of three ways. When using file type, the orientations are determined from a file, similar to “qvectors.txt” in section scattering.vectors.file. The other two methods allow spherical or cylindrical averaging using an internal algorithm to generate random orientations.

The resolution specifies the number of directions which contribute to the integral of the orientationally averaged scattering intensity. In each case, the q vector length is taken from the original q vector. When computing orientationally averaged scattering diagrams with more than one q vector, the same set of random orientations is used in each case.

**First example**

```
<average>
 <orientation>
  <type>vectors</type>
  <vectors>
   <type>sphere</type>
   <algorithm>boost_uniform_on_sphere</algorithm>
   <resolution>1000</resolution>
   <seed>5</seed>
  </vectors>
 <orientation>
</average>
```

triggers isotropic (sphere) orientational averaging, using 1000 random directions and a seed value of 5 for the random number generator.

**Second example**

```
<average>
 <orientation>
  <type>vectors</type>
  <vectors>
   <type>cylinder</type>
   <algorithm>boost_uniform_on_sphere</algorithm>
   <resolution>1000</resolution>
   <seed>5</seed>
   <axis>
    <x>1</x><y>1</y><z>1</z>
   <axis>
  </vectors>
 <orientation>
</average>
```

triggers anisotropic (cylindrical) orientational averaging, using 1000 random directions and a seed value of 5 for the random number generator. The cylinder axis points towards $\begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$.

**Multipole** ...

### 2.3.5 DSP

In the first stage the software computes complex scattering amplitudes. For coherent scattering (all) the results on each nodes are then communicated and gathered on a selected node. For incoherent scattering (self) this is not necessary. The aggregated data corresponds to the full time series of the complex scattering amplitue for the system and the individual atoms, for coherent and incoherent scattering, respectively. At this stage, the user may employ a time series analysis or manipulation of the data. Currently two types of routines are implemented, one computing the autocorrelation and the other one doing an element-wise complex conjugate multiplication.

**Autocorrelate** When using the dsp type autocorrelate, the signal is replaced by its autorrelation. Autocorrelation can be either computed with a direct algorithm or with fftw routines. The fftw routines usually feature a superior scaling for large number of timesteps.

Second example

```
<dsp>
 <type>autocorrelate</type>
 <method>fftw</method>
</dsp>
```

will trigger the autocorrelation of the scattering signal. The fftw method is used.

**Square** When using the dsp type square, each element of the signal is multiplied with itsc onjugate complex value. The resulting signal is pure real and can be regarded as the time series of the zero time delay value of the signal autocorrelation.

Second example

```
<dsp>
 <type>square</type>
</dsp>
```

triggers the squaring of the scattering signal. The resulting signal will be purely real valued.

### 2.3.6 Signal

Each q vector yields a complete time series of the scattering intensity with the exact form depending on any settings in the dsp section. For some scattering calculations the time dependent information can be eliminated and replaced by a mean value, i.e. the scattering diagram becomes a function of only the q vector. In that case the total time-dependent signal "fqt" may be discarded and only some aspects of this function be preserved. Currently 3 additional values are computed for each "fqt": "fq" which is the total time integral of "fqt", "fq0" which is the zero-time element (for correlation, the zero time delay element) of "fqt" and "fq2", which corresponds to the complex conjugate multiplication of "fq". Whether or not these data element are written to the final signal file, can be triggered by activating their corresponding values. The default is that each value is written to the output signal file. The output file is written in the hdf5 format.

Second example

```
<signal>
 <file>mysignal.h5</file>
 <fqt>false</fqt>
 <fq0>true</fq0>
 <fq>true</fq>
 <fq2>false</fq2>
</signal>
```

will write dataset entries for fq0 and fq, but not for fqt and fq2. The output ist stored in hdf5 format in the file with name "mysignal.h5".

### 2.3.7 Background

Each atom has an assigned atomic scattering length. In case of x-ray scattering, the scattering length depends on the q vector length. Since the scattering from molecular structure data only incorporates atoms which are explicitly modeled, the final scattering diagram is missing scattering from the surrounding and the solvent. For small values of q, the surrounding can be approximated by substracting an effective scattering length density of the typical system from the individual atomic scattering lengths. The correction requires to approximate the excluded volume effect of the particular atom. One of two major contributions comes from the size of the particular atom, the other from the molecular phase it is incorporated in. The database defines excluded volumes for common atom types. Additionally the user may scale these volumes dependent on the particular material the atoms are incorporated in by specifying a kappa value (scaling coefficient) and the respective atom selection. For instance an oxygen atom within water may displace more volume than an average oxygen atom within a protein.

A major idea of this type of correction is that the scattering of a disordered system, e.g. water, should not produce a scattering intensity for low q values (q=0). However, the scattering calculation of a finite box of water will result in a non-zero scattering intensity at low q values, which is an artificat due to the missing surrounding. The surrounding can be approximated by offsetting the individual atomic scattering lengths so that the overall scattering becomes zero at low q values.

Second example

```
<background>
 <factor>0.005</factor>
 <kappas>
  <kappa>
   <selection>Water</selection>
   <value>1.42</value>
  </kappa>
 </kappas>
</background>
```

set the background scattering length density to 0.005 and scales the volumes for atoms incorpated in the selection "Water" by a factor of 1.42.

## 2.4 Limits

The parameters specified in the limits section allow to adjust threshold values and performance figures. Threshold values exist to guarantee that the software does not crash due to resource starvation. It also protects the compute nodes from abusive configurations. However, the threshold value might not fit any possible use case, in which cases the user may may want to overwrite theses values. Some values have limited lifetime within the application and/or limited scope. The section limits is organized into contexts. Each context has a certain lifetime during the application, see Figure X for details. Parameters are usually declared in the context in which they are instantiated. However, the lifetime of the particular parameter may exceed the lifetime of the context (e.g. setting the buffer size for coordinates during staging, which will remain during the computation). The default values are tuned to allow for a wide range of use case and applicability on the state-of-the art cluster designs. When changing the default values, the user should take care to guarantee that the available hardware resources match the computational requirements.

### 2.4.1 Stage

Before any computation is performed, the cartesian coordinates are read into local memory. This staging of the data is split into two phases.

In the first phase, the first partition reads the trajectory data from the storage device (disk,network) and stores them into the internal buffer for the coordinates data (limits.stage.memory.data). When computing coherent scattering, the data alignment in the trajectory files coiincites with the partitioning scheme, allowing each node to read the coordinates directly into the local buffer. For incoherent scattering, the data has to be aligned by atoms, thus requiring a tranposition of the data during the initial read. This requires additional buffers (limits.stage.memory.buffer). The transposition of the data is carried out through a collective MPI all-to-all, which results in a synchronization point. To minimize the number of synchronization points, the internal buffer is filled completely, before the data gets transposed. The internal buffer has to have a minimum size to hold at least one frame of the data. The size of the partition determines the number of nodes which access the trajectory data in parallel, thus increasing the partition size results in a more aggressive IO behavior.

In the second phase, the coordinates stored in the first partition are cloned to all other partitions. This is implemented through the MPI collective broadcast.

Example

```
<stage>
 <memory>
  <data>600000000</data>
  <buffer>50000000</buffer>
 </memory>
</stage>
```

The available memory for the local storage of coordinates is set to about 600MB. The buffer during data exchange is about 50MB.

### 2.4.2 Signal

The output file is written in hdf5 format. Parameters which are related to the content of the signal file are given in scattering.signal. The parameters in this section (limits.stage) determine "how" the signal file is written. This can affect overall performance. The default parameters should yield good performance in most cases. Currently, only the chunksize parameter is adjustable. It determines the minimum size of a data element existiting on the disk. Please refer to the HDF5 manual for details on chunks. The default value is 10000 (corresponds to complex value entries, e.g. 16 bytes each). For the "fqt" signal, the cunks are aligned the time dimension. If the time dimension has significantly less than 10000 entries, the chunks contain more than one q vector. In general large chunksizes are prefered for large datasets, because each chunk element has to be managed within the HF5 file. Millions of chunks may slow down the reading and writing of the data considerably. For acceptable performance, the number of chunk elements should be kept to be smaller than 50000. With a default chunksize of 10000 (160kbyte) this corresponds to a file size of 8GB. If larger datasets have to be stored, the chunksize should be increased.

EXAMPLE

```
<signal>
 <chunksize>20000</chunksize>
</signal>
```

doubles the chunksize

### 2.4.3 Computation

During the calculation the incoherent (self) and coherent (all) scattering, the total time signal for each q vector orientation has to be aggregated on one node. Also, each node has to keep a local cache of the total time signal to avoid unneccessary communication. This can consume a considerable amount of computer memory, which might lead to resource starvation. The parameter in section computation.memory protects the user from accidental memory overconsumption. The current defaults allows for storage of up to 4 million time steps (frames). If longer trajectories have to be examined and the necessary hardware requirements are met, adjusting the parameters under computation.memory allows for an arbitrary long time signal. The software also has experimental support for threads. By default only one worker thread per MPI node is active. Setting computation.threads to higher values allows the use of multiple threads to utilize local parallelism. To avoid synchronization between the threads, each thread has own its own memory space. Thus the use of threads may be memory limited. The utility of threads is scoped to averaging prodecures, i.e. it enables parallelism for the computation of orientational averages. Not all buffers are used at the same time and by all modes. Sassena has a memory check routine which anticipates the memory use and provides guidance on the recommended limits. It is thus recommended to only increase the limits if the software asks for it. The convience parameter "scale" provides a means to simply increase the memory limits by the specified factor, which is useful for underallocation of compute nodes for the sake of providing more memory per MPI process (e.g. a 12-core computer node may have 12GB RAM. Allocating 12 MPI processes would provide a maximum of 1GB memory to each process. If allocating 3 MPI processes we allow for 4GB per process and simply increase the sassena software memory limits by setting "scale" to 4.)

EXAMPLE

```
<computation>
 <memory>
  <signal_buffer>200000000<signal_buffer>
  <result_buffer>200000000<result_buffer>
  <exchange_buffer>200000000<exchange_buffer>
  <alignpad_buffer>200000000<alignpad_buffer>
  <scale>4</scale>
 </memory>
 <threads>4</threads>
</computation>
```

increases the internal memory threshold for each buffer to 800MB. Also the number of worker threads is set to 4.

### 2.4.4 Services

To avoid synchronization points between the partitions when writing data to the output file and when reporting progress to the console, the necessary services have been implemented as seperate network protocols. When the software starts up, it initializes the services and starts the corresponding threads. The server threads are located on MPI node rank 0. Each MPI node then sends signal output and progress information via these interfaces. These interfaces bind to the tcp ethernet. This is OK, since the amount of progress information and the final signal data fits well within the capacities of gigabit ethernets. The effect of network latencies is reduced by the implementation of signal output buffers, which can be adjusted by the parameters in limits.services.signal.memory. The parameters in limits.services.signal.times allows to specificies time intervals in seconds for which the signal data should be communicated an written to disk. Using the MPI layer for progress and signal output data, would allow better performance, however it requires to dedicate MPI nodes (threading support for MPI is still not supporting on all machines).

EXAMPLE

```
<services>
 <signal>
  <memory>
   <server>30000000</server>
   <client>2000000</client>
  </memory>
  <times>
   <serverflush >300</serverflush >
   <clientflush >300</clientflush >
  </times>
 </signal>
</services>
```

sets the signal output buffer sizes to about 30MB for the server and 2MB for the clients. The timeouts for flushing data to disk or to the server is set to 300 seconds for the server and the client, respectivley.

### 2.4.5 Decomposition

The efficient utilization of the parallel environment requires the partitioning of the problem based on some metrics. For coherent (all) scattering the best partitioning strategy is frame based, for incoherent (self) it is atom based. However, the number of frames and atoms may limit the scalability. In that case more than one q vector may be processed in parallel. The software uses a heuristic deterministic partitioning scheme to calculate absolute utilization factors. It will find the parititoning with the global best utilization. When more than one best solution (utilization) is available, the paritioning algorithm favors large parititons. The user may want to manually specifiy the partition size by setting limits.decomposition.partitions.automatic to false and set the partition size with limits.decomposition.partitions.size. This may be necessary when a pariticular partitioning is favored, e.g. to match the number of cores per machine with the partition size, thus eliminating inter-node communication. The algorithm does not take this into account. Another reason to fix the partition size is to achieve a specific IO performance, since the parallel bandwidth (number of nodes which read the trajectory) during the staging is determined by the partition size.

EXAMPLE

```
<decomposition>
 <partitions>
  <automatic>false </automatic>
  <size >8</size >
 </partitions >
</decomposition>
```

disables automatic decomposition and set the partition size to 8. Given that at least as many q vectors are to be computed, this yields a total of 5 active paritions when using 40 nodes.

## 2.5 Debug

...

# 3 Reference

The configuraton file is organized into various section. The final parameter have either string, integer or double type. More complex types are defined as their own section. The following list of tables allows gives an overview of the hierarchical organization.

| root | | | | |
|---|---|---|---|---|
| name | type | instances | default | allowed |
| sample | sample | 0..1 | - | - |
| stager | stager | 0..1 | - | - |
| scattering | scattering | 0..1 | - | - |
| limits | limits | 0..1 | - | - |
| debug | debug | 0..1 | - | - |

Some sections may require an element of type vector:

| vector | | | | |
|---|---|---|---|---|
| name | type | instances | default | allowed |
| x | double | 0..1 | 0 | any floating point number |
| y | double | 0..1 | 0 | any floating point number |
| z | double | 0..1 | 0 | any floating point number |

## 3.1 Sample

| sample | | | | |
|---|---|---|---|---|
| name | type | instances | default | allowed |
| structure | sample.structure | 0..1 | - | - |
| framesets | sample.framesets | 0..* | - | - |
| selections | sample.selections | 0..* | - | - |
| alignments | sample.alignments | 0..* | - | - |
| motions | sample.motions | 0..* | - | - |

### 3.1.1 Structure

| sample.structure | | | | |
|---|---|---|---|---|
| name | type | instances | default | allowed |
| file | string | 0..1 | structure.pdb | any valid filename |
| format | string | 0..1 | pdb | pdb |

### 3.1.2 Framesets

| sample.framesets | | | | |
|---|---|---|---|---|
| name | type | instances | default | allowed |
| stride | int | 0..1 | 1 | positive int |
| first | int | 0..1 | 0 | positive int |
| last | int | 0..1 | - | positive int |
| frameset | sample.framesets.frameset | 0..* | - | - |

| sample.framesets.frameset | | | | |
|---|---|---|---|---|
| name | type | instances | default | allowed |
| file | string | 0..1 | sample.dcd | any valid filename |
| format | string | 0..1 | dcd | dcd, xtc, trr, pdb |
| first | int | 0..1 | 1 | positive int |
| last | int | 0..1 | 1 | positive int |
| clones | int | 0..1 | 1 | positive int |
| index | strong | 0..1 | “file” with tnx | any valid filename |

### 3.1.3 Selections

| sample.selections | | | | |
|---|---|---|---|---|
| name | type | instances | default | allowed |
| selection | sample.selections.selection | 0..* | - | - |

| sample.selections.selection | | | | |
|---|---|---|---|---|
| name | type | instances | default | allowed |
| type | string | 0..1 | index | index, range, lexical, file |
| name | string | 0..1 | _#instance | any XML text string |
| index | int | 0..* | - | positive int |
| from | int | 0..1 | 0 | positive int |
| to | int | 0..1 | 0 | positive int |
| file | string | 0..1 | selection.pdb | any valid filename |
| format | string | 0..1 | pdb | pdb, ndx |
| selector | string | 0..1 | beta | format: allowed<br>pdb: beta<br>ndx: name |
| expression | string | 0..1 | type: default<br>lexical: “”<br>file: (format: default; pdb: 1\|1\\\\.0\|1\\\\.00; ndx: .*) | any regular expression |

### 3.1.4 Alignments

| sample.alignments | | | | |
|---|---|---|---|---|
| name | type | instances | default | allowed |
| alignment | sample.alignments.alignment | 0..* | - | - |

| sample.alignments.alignment | | | | |
|---|---|---|---|---|
| name | type | instances | default | allowed |
| type | string | 0..1 | center | center,fittrans,fitrot,fitrottrans |
| selection | string | 0..1 | system | any predefined selection |
| order | string | 0..1 | pre | pre, post |
| reference | sample.alignments.alignment.reference | 0..1 | - | - |

| sample.alignments.alignment.reference | | | | |
|---|---|---|---|---|
| name | type | instances | default | allowed |
| type | string | 0..1 | frame | frame, file |
| frame | integer | 0..1 | 0 | positive int |
| file | string | 0..1 | sample.structure.file | any valid filename |
| format | string | 0..1 | sample.structure.format | sample.structure.format |
| selection | string | 0..1 | sample.alignments.alignment.selection | any valid selection |

### 3.1.5 Motions

| sample.motions | | | | |
|---|---|---|---|---|
| name | type | instances | default | allowed |
| motion | sample.motions.motion | 0..* | - | - |

| sample.motions.motion | | | | |
|---|---|---|---|---|
| name | type | instances | default | allowed |
| type | string | 0..1 | linear | fixed, linear, oscillation, randomwalk, brownian, localbrownian |
| displace | double | 0..1 | 0.0 | any floating point number |
| direction | vector | 0..1 | x=1, y=0, z=0 | valid vector defintion |
| selection | string | 0..1 | system | any predefined selection |
| seed | int | 0..1 | 0 | positive int |
| sampling | int | 0..1 | 1 | positive int |
| frequency | double | 0..1 | 0.001 | any floating point number |
| radius | double | 0..1 | "displace" * 10 | any floating point number |

## 3.2 Stager

| stager | | | | |
|---|---|---|---|---|
| name | type | instances | default | allowed |
| target | string | 0..1 | system | any valid selection |
| mode | string | 0..1 | frames | frames, atoms |
| dump | bool | 0..1 | false | true, false |
| file | string | 0..1 | dump.dcd | any valid filename string |
| format | string | 0..1 | dcd | dcd |

## 3.3 Scattering

| scattering | | | | |
|---|---|---|---|---|
| name | type | instances | default | allowed |
| type | string | 0..1 | all | all, self |
| dsp | scattering.dsp | 0..1 | - | - |
| average | scattering.average | 0..1 | - | - |
| vectors | scattering.vectors | 0..1 | - | - |
| background | scattering.background | 0..1 | - | - |
| signal | scattering.signal | 0..1 | - | - |
| | | | | |

### 3.3.1 DSP

| scattering.dsp | | | | |
|---|---|---|---|---|
| name | type | instances | default | allowed |
| type | string | 0..1 | autocorrelate | autocorrelate, square, plain |
| method | string | 0..1 | fftw | direct, fftw |

### 3.3.2 Average

| scattering.average | | | | |
|---|---|---|---|---|
| name | type | instances | default | allowed |
| orientation | scattering.average.orientation | 0..1 | - | - |

| scattering.average.orientation | | | | |
|---|---|---|---|---|
| name | type | instances | default | allowed |
| type | string | 0..1 | vectors | vectors, multipole |
| vectors | scattering.average.orientation.vectors | 0..1 | - | - |
| multipole | scattering.average.orientation.multipole | 0..1 | - | - |

| scattering.average.orientation.vectors | | | | |
|---|---|---|---|---|
| name | type | instances | default | allowed |
| type | string | 0..1 | sphere | sphere, cylinder, file |
| algorithm | string | 0..1 | boost_uniform_on_sphere | type: allowed<br>sphere: boost_uniform_on_sphere<br>cylinder: boost_uniform_on_sphere, raster_linear |
| file | string | 0..1 | qvector-orientations.txt | any valid filename |
| seed | int | 0..1 | 0 | positive int |
| resolution | int | 0..1 | 100 | positive int |
| axis | vector | 0..1 | x=0, y=0, z=1 | valid vector definition |

| scattering.average.orientation.multipole | | | | |
|---|---|---|---|---|
| name | type | instances | default | allowed |
| type | string | 0..1 | sphere | sphere, cylinder |
| resolution | int | 0..1 | 20 | positive int |
| axis | vector | 0..1 | x=0, y=0, z=1 | valid vector definition |

### 3.3.3 Vectors

| scattering.vectors | | | | |
|---|---|---|---|---|
| name | type | instances | default | allowed |
| type | string | 0..1 | single | single, scans, file |
| single | vector | 0..1 | x=0, y=0, z=1 | valid vector definition |
| scans | scattering.vectors.scans | 0..1 | - | - |
| file | string | 0..1 | qvectors.txt | any valid filename |

| scattering.vectors.scans | | | | |
|---|---|---|---|---|
| name | type | instances | default | allowed |
| scan | scattering.vectors.scans.scan | 0..3 | - | - |

| scattering.vectors.scans.scan | | | | |
|---|---|---|---|---|
| name | type | instances | default | allowed |
| from | double | 0..1 | 0 | any floating point number |
| to | double | 0..1 | 1 | any floating point number |
| points | int | 0..1 | 100 | positive int |
| exponent | double | 0..1 | 1.0 | any floating point number |
| base | vector | 0..1 | x=1, y=0, z=0 | valid vector definition |

### 3.3.4 Signal

| scattering.signal | | | | |
|---|---|---|---|---|
| name | type | instances | default | allowed |
| fqt | bool | 0..1 | true | true, false |
| fq0 | bool | 0..1 | true | true, false |
| fq | bool | 0..1 | true | true, false |
| fq2 | bool | 0..1 | true | true, false |

### 3.3.5 Background

| scattering.background | | | | |
|---|---|---|---|---|
| name | type | instances | default | allowed |
| factor | double | 0..1 | 0 | any floating point number |
| kappas | scattering.background.kappas | 0..1 | - | - |

| scattering.background.kappas | | | | |
|---|---|---|---|---|
| name | type | instances | default | allowed |
| kappa | scattering.background.kappas.kappa | 0..1 | - | - |

| scattering.background.kappas.kappa | | | | |
|---|---|---|---|---|
| name | type | instances | default | allowed |
| selection | string | 0..1 | system | any predefined selection |
| value | double | 0..1 | 1.0 | any floating point number |

## 3.4 Limits

| limits | | | | |
|---|---|---|---|---|
| name | type | instances | default | allowed |
| stage | limits.stage | 0..1 | - | - |
| signal | limits.signal | 0..1 | - | - |
| computation | limits.computation | 0..1 | - | - |
| services | limits.services | 0..1 | - | - |
| decomposition | limits.decomposition | 0..1 | - | - |

### 3.4.1 Stage

| limits.stage | | | | |
|---|---|---|---|---|
| name | type | instances | default | allowed |
| memory | limits.stage.memory | 0..1 | - | - |

| limits.stage.memory | | | | |
|---|---|---|---|---|
| name | type | instances | default | allowed |
| data | int | 0..1 | 524288000 | positive int |
| buffer | int | 0..1 | 104857600 | positive int |

### 3.4.2 Signal

| limits.signal | | | | |
|---|---|---|---|---|
| name | type | instances | default | allowed |
| chunksize | int | 0..1 | 10000 | positive int |

### 3.4.3 Computation

| limits.computation | | | | |
|---|---|---|---|---|
| name | type | instances | default | allowed |
| memory | limits.computation.memory | 0..1 | - | - |
| threads | int | 0..1 | 1 | positive int |

| limits.computation.memory | | | | |
|---|---|---|---|---|
| name | type | instances | default | allowed |
| signal_buffer | int | 0..1 | 104857600 | positive int |
| result_buffer | int | 0..1 | 104857600 | positive int |
| exchange_buffer | int | 0..1 | 104857600 | positive int |
| alignpad_buffer | int | 0..1 | 209715200 | positive int |
| scale | int | 0..1 | 1 | positive int |

### 3.4.4 Services

| limits.services | | | | |
|---|---|---|---|---|
| name | type | instances | default | allowed |
| signal | limits.services.signal | 0..1 | - | - |

| limits.services.signal | | | | |
|---|---|---|---|---|
| name | type | instances | default | allowed |
| memory | limits.services.signal.memory | 0..1 | - | - |
| times | limites.services.signal.times | 0..1 | - | - |

| limits.services.signal.memory | | | | |
|---|---|---|---|---|
| name | type | instances | default | allowed |
| server | int | 0..1 | 104857600 | positive int |
| client | int | 0..1 | 10485760 | positive int |

| limits.services.signal.times | | | | |
|---|---|---|---|---|
| name | type | instances | default | allowed |
| serverflush | int | 0..1 | 600 | positive int |
| clientflush | int | 0..1 | 600 | positive int |

## 3.5 Debug

not documented... (yet)