Neper Reference Manual

The documentation for Neper 2.0.0
A software package for polycrystal generation and meshing

10 January 2014

Romain Quey

Table of Contents

Conditions of Use

Copying Conditions

Neper is "free software"; this means that everyone is free to use it and to redistribute it on a free basis. Neper is not in the public domain; it is copyrighted and there are restrictions on its distribution, but these restrictions are designed to permit everything that a good cooperating citizen would want to do. What is not allowed is to try to prevent others from further sharing any version of Neper that they might get from you.

Specifically, we want to make sure that you have the right to give away copies of Neper, that you receive source code or else can get it if you want it, that you can change Neper or use pieces of Neper in new free programs, and that you know you can do these things.

To make sure that everyone has such rights, we have to forbid you to deprive anyone else of these rights. For example, if you distribute copies of Neper, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must tell them their rights.

Also, for our own protection, we must make certain that everyone finds out that there is no warranty for Neper. If Neper is modified by someone else and passed on, we want their recipients to know that what they have is not what we distributed, so that any problems introduced by others will not reflect on our reputation.

The precise conditions of the license for Neper are found in the General Public License that accompanies the source code (see Appendix E [GNU General Public License], page 63). Further information about this license is available from the GNU Project webpage `http://www.gnu.org/copyleft/gpl-faq.html`.

The Neper software package can be downloaded from `http://neper.sourceforge.net`. It also has two dedicated mailing lists,

- neper-announce: the "read-only" list for important news: new releases, bug fixes, etc. (low traffic, highly recommended!)

  To subscribe, visit `https://lists.sourceforge.net/lists/listinfo/neper-announce`. The list is archived at `http://sourceforge.net/mailarchive/forum.php?forum_name=neper-announce`.
- neper-users: the "read-write" list for users. Please send all questions, bug reports, requests or any errors or omissions in this manual to this list.

  To subscribe, visit `https://lists.sourceforge.net/lists/listinfo/neper-users`; to send a message, use `neper-users@lists.sourceforge.net`. The list is archived at `http://sourceforge.net/mailarchive/forum.php?forum_name=neper-users`.

The best way to get help is by checking out the list archives or by sending a message to the neper-users list. There is no need to subscribe to the list to send a message nor receive a reply. If you wish to contact the developer directly, use `rquey@users.sourceforge.net`.

User Guidelines

If you use Neper for your own work, please,

- mention it explicitly in your reports (books, papers, talks, . . .).
- cite the following article: *R. Quey, P.R. Dawson, F. Barbe. Large-scale 3D random polycrystals for the finite element method: Generation, meshing and remeshing. Computer Methods in Applied Mechanics and Engineering, vol. 200, pp. 1729–1745, 2011.*

1 Introduction

1.1 The Neper Project

1.1.1 Description

Neper is a software package for polycrystal generation and meshing. The polycrystals can be 1D, 2D or 3D. Neper is built around three modules,

- Module -T is the module for generating polycrystals, which is carried out by tessellation of a bounded domain of space. Several types of tessellations are available: Voronoi tessellations, hardcore Voronoi tessellations, centroidal Voronoi tessellations, Laguerre-Voronoi tessellations and regular tessellations. The tessellated domain can be cubic, cylindrical or of any other convex shape. The tessellations can then be deformed for generating elongated cells. Crystal orientations are provided for the grains. Module -T also enables to generate 2-scale polycrystals, which are obtained by generating tessellations into the cells of a primary tessellation. Finally, module -T includes a "regularization" capability to remove the small features of the tessellations, which then enables better-quality meshing. The output is a tessellation file at scalar or raster format.
- Module -M is the module for meshing polycrystals described as tessellation files. Two capabilities are available: meshing into tetrahedral elements that conform to the tessellation morphology or meshing into regular hexahedral elements. Tetrahedron meshing includes features that are necessary to get good-quality meshes: optimized meshing rules and multi-meshing (the concurrent use of several meshing algorithms). Remeshing is also available to generate a new, good-quality mesh from a deformed mesh. Tetrahedron meshing of raster tessellations works for 1D and 2D tessellations only.
- Module -V is the visualization module. It enables to generate publication-quality images of the tessellations and meshes. It is also possible to visualize data on them using colours or displacements of the nodes (for meshes) and to plot data on slices of the mesh. The output is a PNG image file.

Neper aims to be an easy-to-use, efficient and robust tool. All the input data are prescribed non-interactively, using command lines and/or ASCII files. This makes it possible to automate all treatments.

1.1.2 Resources

Several, complementary resources describing the Neper capabilities are available,

- The Neper reference manual, which is the present document. It describes all the Neper capabilities. It is made of one chapter for each module, which describes the available commands and result files and provides some examples. The manual comes both as a PDF file and an info file. Provided that the info file is properly installed at your site, it can be accessed by the command: `info neper`.
- The Neper homepage, http://neper.sourceforge.net. It is where the Neper distribution can be downloaded from. The page also provides an introduction to Neper.
- The Neper article, "*R. Quey, P.R. Dawson and F. Barbe, Large-scale 3D random polycrystals for the finite element method: Generation, meshing and remeshing, Comput. Methods Appl. Mech. Engrg., vol. 200, pp. 1729-1745, 2011.*". It provides details on the algorithms. It can be downloaded from the Neper homepage or by following this link: http://neper.sourceforge.net/docs/neper-reference-paper.pdf.

1.2 Installing Neper

Neper is written in (mostly ANSI) C and a very little C++. It can run on any Unix-like system. Neper can be compiled using CMake, a standard open-source build system. The main steps are as follows,

- Create a '`build`' directory, for example as a subdirectory of Neper's '`src`' directory,
  ```
  $ mkdir build
  ```
- Run CMake from within the '`build`' directory, pointing to Neper's '`src`' directory,
  ```
  $ cd build
  $ cmake ..
  ```
- To build and install Neper, then simply type,
  ```
  $ make
  $ make install (as root)
  ```

This will use the default configuration options and should work out of the box on condition that the required libraries are available and installed in standard system locations. A finer configuration can be achieved before building Neper, as described in the following.

Neper has mandatory as well as optional dependencies. Some dependencies are managed at compilation time,

- the GSL library (mandatory)

  It is likely to be available on your system or from your system package manager (binary and development packages). Alternatively, the source code version can be obtained from the GSL homepage, `http://www.gnu.org/software/gsl`.
- the libmatheval library (optional, *included* by default)

  It is likely to be available on your system or from your system package manager (binary and development packages). Alternatively, the source code version can be obtained from the libmatheval homepage, `http://www.gnu.org/software/libmatheval`.
- the libScotch library (optional, *not included* by default)

  Module -M includes mesh partitioning, which requires the libScotch library (version `5.1.12` or later). It can be downloaded from the Scotch homepage, `www.labri.fr/perso/pelegrin/scotch`.

Optional dependencies can be toggled on / off using CMake's GUI (`cmake-gui`) or `ccmake`, by setting variables `HAVE_`*`LIBRARYNAME`* to ON or OFF, respectively. The commands would be,

```
$ cmake-gui ..
```

or,

```
$ ccmake ..
```

The additional variables '`HAVE_DEBUGGING`', '`HAVE_PROFILING`' and '`HAVE_OPTIMIZATION`' are for development and should not be changed from their default values for production use ('`OFF`', '`OFF`' and '`ON`', respectively); warnings are printed at Neper execution otherwise.

The following dependencies are only needed at run time,

- the Gmsh program (mandatory for module -M)

  This version on Neper is intended to work with Gmsh (version `2.4.2` or later), which can be downloaded from `http://www.geuz.org/gmsh`. A working Gmsh installation must be available on your system. (Compiling Gmsh from the source code provides faster meshing.)
- the POV-Ray program (mandatory for module -V)

  Module -V uses POV-Ray to produce publication-quality images of the tessellations and meshes. POV-Ray can be downloaded from `http://www.povray.org`. POV-Ray must be available in the terminal through the command: `povray`.

Finally, the Neper installation can be tested out by running a series of tests, as follows,

```
$ neper --test
```

For a proper Neper installation, all tests should pass. (The command actually is an alias for '`neper -D all -runmode fast`', see Appendix C [Developer's Guide], page 53 for details.)

1.3 Getting Started

Using Neper consists in running the command '`neper`' in a terminal, with a list of arguments,

```
$ neper list_of_arguments
```

The arguments define the problem for Neper to solve. Neper then returns output in ASCII files, together with some messages in the terminal. Neper includes some general-purpose self-explanatory commands,

```
$ neper --help
$ neper --version
$ neper --license
```

1.3.1 Calling a Module

A typical Neper invocation consists in calling a module and providing it with a number of arguments,

```
$ neper module_name module_arguments
```

The module names are '`-T`', '`-M`' and '`-V`'. The module arguments can include both required input data and options. Input data (when not a file name) and options start by '`-`'. The options can be given in arbitrary order and are to be specified as follows: '*`option_name option_argument`*'. Any option takes one and only one argument. The options can be written both in British English and in American English, even if only the British English versions are provided in this manual. String completion is available for all options, so they may be abbreviated as long as the abbreviation is not ambiguous. For example, in module `-T`, option '`-regularization`' can be abbreviated to '`-reg`'. Logical options can be selected or disabled by providing as argument the value '`1`' or '`0`', respectively. For option arguments that are integer or real numbers, mathematical or logical expressions can be used if libmatheval is available (for the list and format of mathematical and logical expressions, see Section A.1 [Mathematical and Logical Expressions], page 41). For example, in module -T, option '`-rcl 0.5`' can also be written as '`-rcl 1/2`' or '`-rcl "cos(pi/3)"`'. Options are tagged by importance level in the reference manual: '[Option]' or '[Secondary option]'. Prerequisites are tagged '[Prerequisite]', input data are tagged '[Input Data]' and post-processing options are tagged '[Post-processing]'. Module -V has some exceptions with respect to the previous rules: the argument cannot be given in arbitrary number, string completion is not available and option '`-loop`' takes several arguments.

1.3.2 Initialization File

When Neper is started, it reads commands from an initialization file, '`$HOME/.neperrc`', if that file exists. This behaviour can be modified using option '`--rcfile`', which must be done *before* calling a module,

```
$ neper --rcfile my_file module_name module_arguments
```

To disable the reading of an initialization file, provide '`none`' as value of *`my_file`*.

When a module of Neper is called, Neper looks for the occurrence of '`neper` *`module_name`*' in the initialization file, then reads all arguments until the subsequent occurrence of '`neper`' (which should denote the beginning of another module option field) or the end of the file. Moreover, any comments can be written using '`neper comments`'. The arguments may be any legal arguments, but are typically limited to frequently-used options.

An example of initialization file is provided below,

```
neper comments ------------------------------------------------------
This is my initialization file (~/.neperrc).
neper -T -reg 1
neper -M -gmsh my_gmsh_path
neper comments ------------------------------------------------------
```

If the initialization file is not found, or if '`neper` *`module_name`*' is not found in the file, Neper will only consider the command line arguments. Also note that if an argument is initialized several times (for example, both in the initialization file and at the command line), the last specified value is retained.

1.3.3 Conventions

1.3.3.1 Manual

The Neper documentation is maintained as a Texinfo manual. Here are the writing conventions used in the document,

1. A command that can be typed in a terminal is printed like `this`, or, in the case of a major command, like

   ```
   $ this
   ```

2. a program (or command) option is printed like '`this`';
3. The name of a variable is printed like `this`;
4. A meta-syntactic variable (i.e. something that stands for another piece of text) is printed like *this*;
5. Literal examples are printed like '`this`';
6. File names are printed like '`this`'.

1.3.3.2 Argument Separators

Some options may take several values, which can be combined using separators, as specified in the option descriptions (see Chapter 2 [Tessellation Module (-T)], page 7, Chapter 3 [Meshing Module (-M)], page 17 and Chapter 4 [Visualization Module (-V)], page 29). There actually are two separators,

1. The '`,`' separator is used to provide several arguments with no dependency between each other, that is, whose corresponding actions can be processed independently. For instance, in module -T, the '`-format`' option can take argument '`tess,ply`' to get the tessellation both in Neper's tessellation format '`.tess`' and in Ply format '`.ply`'.
2. The '`:`' separator is used to combine several arguments that show dependency between each other, that is, whose corresponding actions cannot be processed independently. For instance, in module -M, the meshing option '`-mesh3dalgo`' takes as argument a combination of a meshing algorithm, '*`mesh`*', and a meshing optimization algorithm, '*`opti`*', under the form '*`mesh`*`:`*`opti`*'. Optimization is applied after meshing, hence the dependency.

In module -T, the '`::`' super-separator is used for 2-scale tessellation generation.

2 Tessellation Module (-T)

Module -T enables to generate *tessellations* of a bounded domain of space, in 1D, 2D or 3D. It also enables to generate *2-scale tessellations*, which are tessellations for which each cell is in turn subdivided into a tessellation. Module -T also enables to *regularize* the tessellations for better-quality meshing. The tessellations are provided in scalar (vector) or raster formats.

Tessellation is achieved by using various kinds of *Voronoi tessellation algorithms*: *Voronoi tessellation*, *hardcore Voronoi tessellation*, *centroidal Voronoi tessellation* or *Laguerre-Voronoi tessellation*. This enables to generate a wide variety of morphologies. The *germs* of the Voronoi cells can be distributed randomly in the domain, which leads to random equiaxed morphology, or else so as to get columnar, bamboo, lamellar or regular morphologies (in 3D, cubes or truncated octahedra). The last possibility is to load a custom distribution of germs. For Laguerre-Voronoi tessellations, the weight distribution can be specified (option '`-weight`'). Finally, the tessellations can be scaled to get flat or elongated cells (option '`-scale`'). *2-scale tessellations* can be obtained by providing 2 levels of tessellation information to the options, using the '`::`' separator.

The *domain* of space in which the tessellation is created can be of any convex shape. In 3D, cuboidal and cylindrical shapes are directly supported while other morphologies can be defined from a set of planes (option '`-domain`'). Regarding the bounded nature of the domain, three types of tessellation can be created: *standard tessellations*, for which all germs are located inside the domain, *periodic tessellations*, whose cells show periodicity conditions at the domain boundary, and *subdomain-type tessellations*, for which germs can be located inside or outside the domain (option '`-ttype`'; periodic and subdomain tessellations are only available in raster format).

Crystal orientations are also provided for the cells. The orientations are randomly distributed according to a uniform distribution, either in the 3D space or along a specific orientation fibre (option '`-ori`', which can be used in conjunction with '`-morpho`'). They can be provided according to different descriptors (option '`-oridescriptor`').

Regularization can be applied to the tessellations, which consists in removing their small edges and faces (option '`-regularization`'). This is a necessary step for getting good-quality meshes using module -M (see Chapter 3 [Meshing Module (-M)], page 17).

Output files describe the tessellation either are the scalar format '`.tess`' or raster format '`.tesr`' (see Appendix B [File Formats], page 47). Both are input files of module -M (see Chapter 3 [Meshing Module (-M)], page 17) and module -V (see Chapter 4 [Visualization Module (-V)], page 29). Third-party software file formats are also available.

Here is what a typical run of module -T looks like,

```
$ neper -T -n 10 -id 1 -reg 1
========================    N   e   p   e   r    ======================
Info   : A software package for polycrystal generation and meshing.
Info   : Version 2.0.0
Info   : Built with: gsl libmatheval
Info   : Loading initialization file '/foo/bar/.neperrc'...
Info   : ---------------------------------------------------------------
Info   : MODULE  -T loaded with arguments:
Info   : [ini file]
Info   : [com line] -n 10 -id 1 -reg 1
Info   : ---------------------------------------------------------------
Info   : Reading input data...
Info   : Creating domain...
```

```
Info   : Creating tessellation...
Info   :   - Distributing germs...
Info   :   - Running tessellation...
Info   : Regularizing tessellation...
Info   :   - loop 2/2: 100% del=14
Info   : Writing tessellation...
Info   :     [o] Writing file 'n10-id1.tess'...
Info   :     [o] Wrote file 'n10-id1.tess'.
Info   : Elapsed time: 0.019 secs.
========================================================================
```

2.1 Arguments

2.1.1 Input Data

Options are detailed below for regular, 1-scale tessellations. For 2-scale tessellations, combine the option argument values at the 2 successive scales with '`::`'. An example is '`-n 10::10`'.

`-n` *integer or char_string* [Input data]
Number of cells of the tessellation. The argument can be a mathematical expression based on the *vol* variable, which is the volume of the domain. For regular morphologies (truncated octahedra, etc., see option '`-morpho`'), the argument must be the number of cells along a dimension of the domain. For lamella morphology, the argument must be of the form '`w=w`' where `w` is the width of the lamellae.
Possible values: `any`. Default value: `none`.

`-id` *integer* [Input data]
Identifier of the tessellation. It defines the seed used by the random number generator to compute the positions (optionally weights) of the germs.
Possible values: `any`. Default value: *`random`*.

`-dim` *integer* [Option]
Specify the dimension of the tessellation.
Possible values: `1, 2 or 3`. Default value: `3`.

`-domain` *char_string* [Option]
Specify the type and size of the domain. In 3D, for a cuboidal shape, provide '`cube(`*`size_x`*`,`*`size_y`*`,`*`size_z`*`)`' and for a cylindrical shape, provide '`cylinder(`*`height`*`,`*`diameter`*`)`'. In 2D, for a rectangular shape, provide '`square(`*`size_x`*`,`*`size_y`*`)`' and for a circular shape, provide '`circle(`*`radius`*`)`'. In 1D, provide '`segment(`*`size_x`*`)`'. To specify the number of facets, *`facet_nb`*, of a circle or cylinder domain, use '`circle(`*`radius`*`,`*`facet_nb`*`)`' or '`cylinder(`*`height`*`,`*`diameter`*`,`*`facet_nb`*`)`'. For an arbitrary 3D shape, provide '`planes(`*`file_name`*`)`', where *`file_name`* is the name of a file containing the total number of planes then, for each plane, the parameters of its equation (a, b, c and d for an equation of the form $a\,x + b\,y + c\,z = d$). The plane normal must be an outgoing vector of the cell.
Possible values: `see above list`. Default value: `cube(1,1,1) in 3D, square(1,1) in 2D and segment(1) in 1D`.

Is it also possible to load a tessellation or a raster tessellation from a file,

`-loadtess` *file_name* [Input data]
Load a tessellation from a file. Provide as argument the file name.
Possible values: `any`. Default value: `none`.

`-loadtesr` *file_name* [Input data]
Load a raster tessellation from a file. Provide as argument the file name. To load only a subregion of a raster tessellation, use the syntax '*file_name*`:crop(`*xmin*`|`*xmax*`|`*ymin*`|`*ymax*`|`*zmin*`|`*zmax*`)`', where *xmin*, *xmax*, *ymin*, *ymax*, *zmin* and *zmax* are the minimum and maximum positions along *x*, *y* and *z*, respectively. For 2D raster tessellations, the *z* boundaries can be omitted. For 1D raster tessellations, the *y* and *z* boundaries can be omitted. To scale the number of points of a raster tessellation, use the syntax '*file_name*`:scale(`*factor*`)`', where *factor* is the scaling factor, or '*file_name*`:scale(`*factor_x*`|`*factor_y*`|`*factor_z*`)`', where *factor_x*, *factor_y* and *factor_z* are the scaling factor along *x*, *y* and *z*, respectively. For 2D raster tessellations, the *z* factor can be omitted. For 1D raster tessellations, the *y* and *z* factors can be omitted.
Possible values: `any`. Default value: `none`.

Finally, it is possible to load a set of points (useful for statistics, see option '`-statpoint`'),

`-loadpoint` *file_name* [Input data]
Load points from a file. See Section B.3 [Position File], page 51 for the file format. Provide as argument the file name.
Possible values: `any`. Default value: `none`.

2.1.2 Tessellation Options

`-morpho` *char_string* [Input data]
Type of morphology of the cells. For random Voronoi tessellations, it can be either equiaxed ('`equiaxed`'), columnar ('`columnar(`*dir*`)`', where *dir* is the columnar direction and can be '`x`', '`y`' or '`z`') or bamboo-like ('`bamboo(`*dir*`)`', where *dir* is the bamboo direction and can be '`x`', '`y`' or '`z`'). To get a lamella morphology, provide '`lamella`'. Regular morphologies also are available: in 3D, cubes ('`cube`', only with '`-format tesr`') and truncated octahedra ('`tocta`'). To load a particular set of germs, use the syntax '`@`*file_name*' where *file_name* is the name of the germ position file (see Section B.3 [Position File], page 51).
Possible values: `see above list`. Default value: `equiaxed`.

`-hardcore` *real* [Input data]
Use this option to get a hardcore Voronoi tessellation. Provide as argument the radius of the exclusion sphere surrounding each of the germs.
Possible values: `any`. Default value: `0`.

`-centroid` *logical* [Input data]
Use this option to get a centroidal Voronoi tessellation. Note that the generation of a centroidal Voronoi tessellation is based on Lloyd's algorithm and can be two to three orders of magnitude as long as for the equivalent Voronoi tessellation. See options '`-centroidfact`', '`-centroidconv`' and '`-centroiditermax`' for convergence criteria.
Possible values: `0 or 1`. Default value: `0`.

`-centroidfact` *real* [Secondary option]
This option can be used with option '`-morpho centroid`', to specify the factor by which the germ positions are shifted between their current positions and the centroid positions, at each iteration. (Lloyd's algorithm is obtained for a value of 1, but a lower value can lead to faster convergence.)
Possible values: `0 to 1`. Default value: `0.5`.

`-centroidconv` *real* [Secondary option]
This option can be used with option '`-morpho centroid`', to specify the maximum tolerance on the distance between the germs and the cell centroids. The tolerance is relative to the

average cell radius.
Possible values: `any > 0`. Default value: `0.02`.

`-centroiditermax` *integer* [Secondary option]
This option can be used with option '`-morpho centroid`', to specify the maximum number of iterations.
Possible values: `any`. Default value: `1000`.

`-weight` *char_string* [Option]
Use this option to generate a Laguerre-Voronoi tessellation. To define the weights associated to the germs, the first way is to load values from a file using the syntax '`@`*file_name*', where *file_name* is the name of the file. The second way is to provide as argument an expression for the distribution of the weight values. The current algorithm generates the weights then distributes the germs by decreasing weight values. As a weight also stands for the radius of a sphere of exclusion, the algorithm may fail to distribute all germs, in which case an error is returned. The expression is a probability density function of the form '*function*`(`*param1*`,`*param2*`,...)`'. The available functions are: '*dirac*`(`*mean*`)`' for the Dirac distribution, '*gaussian*`(`*mean*`,`*sig*`)`' for the Gaussian distribution of mean *mean* and standard deviation *sig*, '*flat*`(`*mean*`,`*radius*`)`' for the flat distribution of mean *mean* and radius *radius*, '*bernoulli*`(`*val1*`,`*val2*`,`*p*`)`' for a Bernoulli-type distribution of probability p, where *val1* is the value associated with probability $p-1$ and *val2* is the value associated with probability p. The distributions can be truncated by passing a minimum and a maximum value as parameter to the function: '*function*`(`*param1*`,`*param2*`,...,`*min*`,`*max*`)`'. It is also possible to provide a weighted sum of probability density functions, which is particularly appropriate to get multimodal distributions. The argument must be of the form: '*function1_weight*`*`*function1*`(...)+`*function2_weight*`*`*function2*`(...)+...`', where *functionid_weight* is a real number representing the additive weight of the distribution.
Possible values: `any`. Default value: `none`.

`-ttype` *char_string* [Option]
Specify the type of tessellation. '`standard`' means that all germs are located inside the domain. '`periodic`' means that cells show periodicity conditions at the domain boundary. '`subdomain`' means that germs can be located inside and outside the domain. '`periodic`' and '`subdomain`' are only available for raster tessellations ('`-format tesr`').
Possible values: `standard, periodic or subdomain`. Default value: `standard`.

`-scale` *real:real:real* [Option]
Specify the factors in the x, y and z directions by which the tessellation is to be scaled once generated. In 2D, the z value can be omitted. In 1D, the y and z values can be omitted.
Possible values: `any`. Default value: `none`.

`-randomize` *real:integer* [Secondary option]
This option can be used to randomize the positions of the germs. Provide as argument the radius of the sphere (circle in 2D, segment in 1D) in which each position is randomly randomized, according to a uniform distribution, and an identifier for the randomization, combined with '`:`'.
Possible values: `any:any`. Default value: `none`.

`-sort` *char_string* [Secondary option]
This option can be used to sort the tessellation cells (typically to facilitate data post-processing). Provide as argument the mathematical expression used for sorting (see Section A.1 [Mathematical and Logical Expressions], page 41).
Possible values: `any`. Default value: `none`.

2.1.3 Crystal Orientation Options

`-ori` *`char_string`* [Option]

Specify the type of crystal orientation distribution. It can be either '`3D`' for orientations in the 3D space, or '`fibre(`*`dir`*`,`*`coo_x`*`,`*`coo_y`*`,`*`coo_z`*`)`' for orientations along a specific fibre (crystal direction (*`coo_x`*, *`coo_y`*, *`coo_z`*) parallel to sample direction *`dir`*, where *`dir`* can be '`x`', '`y`', or '`z`'). Crystal orientations are distributed randomly according to a uniform distribution in the specified space.
Possible values: `3D` or `fibre(...)`. Default value: `3D`.

`-oricrysym` *`char_string`* [Secondary option]

Specify the crystal symmetry. This is used to reduce the domain of definition of the orientation descriptors.
Possible values: `triclinic` or `cubic`. Default value: `triclinic`.

2.1.4 Regularization Options

`-regularization` *`logical`* [Option]

Enable tessellation regularization. Regularization removes the small edges and, indirectly, the small faces. In return, faces can become non-planar (in 3D). This is controlled by options '`-fmax`', '`-sel`' and '`-mloop`'. Using regularization enables to get better-quality meshes using module -M (see Chapter 3 [Meshing Module (-M)], page 17).
Possible values: `0` or `1`. Default value: `0`.

`-fmax` *`real`* [Option]

Maximum allowed face flatness fault (in degrees). The flatness fault is the maximum angle between the normals at two locations on a face.
Possible values: `0` to `180`. Default value: `20`.

`-sel` or `-rsel` *`real`* [Secondary option]

Absolute or relative small edge (maximum) length. `rsel` is defined relative to the average cell size (volume in 3D, area in 2D and length in 1D). The default `-rsel 1` leads to a length of 0.25 for a unit volume cell in 3D, 0.125 for a unit area cell in 2D and 0.2 for a unit length cell in 1D. The value also enables to avoid mesh refinement with the default meshing parameters (see Chapter 3 [Meshing Module (-M)], page 17). It is also possible to specify values on a per-cell basis. The first way is to use the syntax *`default_sel`*`,`*`cell_expr1`*`:`*`cell_sel1`*`,`*`cell_expr2`*`:`*`cell_sel2`*`...`, where *`default_sel`* is the default small edge length, *`cell_expri`* is an expression defining the set of cells *i* and *`cell_seli`* is the corresponding small edge length. '*`cell_expri`*' can be any expression based on variables provided in Section A.2 [Tessellation Keys], page 41. The expressions are processed one after the other. When processing expression *`cell_expri`*, the matching cells are assigned *`cell_seli`* as small edge length. Typically, option '`-rsel`' should be passed the same argument than option '`-rcl`' of module -M, see Chapter 3 [Meshing Module (-M)], page 17. The second way is to load values from an external file using the syntax '`@`*`file_name`*', where *`file_name`* is the name of the file containing the length values.
Possible values: `any`. Default value: `-rsel 1`.

`-mloop` *`integer`* [Secondary option]

Maximum number of regularization loops. During each loop, the small edges are considered in turn from the shortest to the largest. Regularization stops when the maximum number of loops is reached or no edges are deleted within a loop.
Possible values: `any`. Default value: `2`.

2.1.5 Filtering Options

The following option can be used to clean up a raster tessellation. For meshing of a 2D raster tessellation, make sure to use '`-filter size`'.

`-filter` *`char_string`* [Option]
Filter a raster tessellation. The available filters can be combined with '`:`'. Filter '`size`' removes, for each cell, the parts of a cell that are not properly connected to the rest of the cell, namely, that do not share with it at least a raster point edge for a 2D tessellation and a raster point face for a 3D tessellation. Use argument '`size(`*`dim`*`)`' to override the type of neighbouring, where *`dim`* is equal to 1 for edge and 2 for face. Filter '`noise`' removes points (or bunch of points) indexed 0 by progressively collapsing them.
Possible values: `any`. Default value: `none`.

2.1.6 Output Options

`-o` *`file_name`* [Option]
Specify the output file name.
Possible values: `any`. Default value: `none`.

`-format` *`char_string`* [Option]
Specify the format of the output file(s). The available formats are the Neper '`tess`' and '`tesr`', the Gmsh '`geo`', the Ply '`ply`' and the 3DEC '`3dec`'. Orientations for the cells can be obtained using '`ori`' (see also options starting by '`-ori`'). Combine the values with '`,`'.
Possible values: `tess, tesr, geo, ply, 3dec, ori`. Default value: `tess`.

`-tesrformat` *`char_string`* [Option]
Specify the format of the raster output file(s). The available formats are ASCII ('`ascii`'), 8-bit binary ('`binary8`'), 16-bit binary ('`binary16`') and 32-bit binary ('`binary32`').
Possible values: `ascii, binary8, binary16, binary32`. Default value: `binary16`.

`-tesrsize` *`integer`* [Option]
Specify the number of points of a raster tessellation along a direction of the domain. In case of a domain of different lengths along the different directions, the argument stands for the geometrical average of the number of points along the different directions, so that the raster points are as cubic as possible. To specify different values along the x, y and z directions, combine the values with '`:`'.
Possible values: `any`. Default value: `20`.

`-oridescriptor` *`char_string`* [Option]
Select the orientation descriptor used in the '`.tess`', '`.tesr`' and '`.ori`' files. It can be Euler angles in Bunge, Kocks or Roe convention (`e`, `ek`, `er`), rotation matrix (`g`), axis / angle or rotation (`rtheta`), Rodrigues vector (`R`) or quaternion (`q`).
Possible values: `above-mentioned values`. Default value: `e`.

`-oriformat` *`char_string`* [Option]
Specify the format of the '`.ori`' output file. The available formats are: the Neper-native `plain` (i.e. only the descriptors on successive lines), the Zset/Zébulon `geof` and the FEPX `fepx`.
Possible values: `above-mentioned values`. Default value: `plain`.

2.1.7 Post-Processing Options

The first options apply to the cells and germs of a tessellation or a raster tessellation, independently of its dimension,

`-statcell` *char_string* [Post-processing]
Provide statistics on the tessellation cells. Give as argument the keys as described in Section A.2 [Tessellation Keys], page 41 for a tessellation and Section A.3 [Raster Tessellation Keys], page 43 for a raster tessellation (combine with '`,`').
Possible values: `any`. Default value: `none`.
Result file: extension '`.stcell`'.

`-statgerm` *char_string* [Post-processing]
Provide statistics on the tessellation germs. Give as argument the keys as described in Section A.2 [Tessellation Keys], page 41 for a tessellation and Section A.3 [Raster Tessellation Keys], page 43 for a raster tessellation (combine with '`,`').
Possible values: `any`. Default value: `none`.
Result file: extension '`.stgerm`'.

For tessellations, it is also possible to get statistics on an entity-basis,

`-statver` *char_string* [Post-processing]
Provide statistics on the tessellation vertices. Give as argument the keys as described in Section A.2 [Tessellation Keys], page 41 (combine with '`,`').
Possible values: `any`. Default value: `none`.
Result file: extension '`.stver`'.

`-statedge` *char_string* [Post-processing]
Provide statistics on the tessellation edges. Give as argument the keys as described in Section A.2 [Tessellation Keys], page 41 (combine with '`,`').
Possible values: `any`. Default value: `none`.
Result file: extension '`.stedge`'.

`-statface` *char_string* [Post-processing]
Provide statistics on the tessellation faces. Give as argument the keys as described in Section A.2 [Tessellation Keys], page 41 (combine with '`,`').
Possible values: `any`. Default value: `none`.
Result file: extension '`.stface`'.

`-statpoly` *char_string* [Post-processing]
Provide statistics on the tessellation polyhedra. Give as argument the keys as described in Section A.2 [Tessellation Keys], page 41 (combine with '`,`').
Possible values: `any`. Default value: `none`.
Result file: extension '`.stpoly`'.

Finally, it is possible to get statistics for a particular set of points. The option applies to a tessellation.

`-statpoint` *char_string* [Post-processing]
Provide statistics on points. The points must be loaded with option '`-loadpoint`'. Give as argument the keys as described in Section A.2 [Tessellation Keys], page 41 (combine with '`,`').
Possible values: `any`. Default value: `none`.
Result file: extension '`.stpoint`'.

2.1.8 Debugging Options

`-checktess` *file_name* [Input data]
Check a tessellation file. Provide as argument the file name. Use this option if the tessellation file fails to load using option '`-loadtess`' or in other modules.
Possible values: `any`. Default value: `none`.

2.2 Output Files

2.2.1 Tessellation

- Neper tessellation file: '`.tess`'

  It contains a scalar description of the tessellation. See Appendix B [File Formats], page 47 for the file syntax.
- Neper raster tessellation: '`.tesr`'

  It contains a raster description of the tessellation. See Appendix B [File Formats], page 47 for the file syntax.
- Gmsh geometry file: '`.geo`'

  It contains a minimal description of the tessellation written under the Gmsh geometry file format '`.geo`'. This file can be opened with Gmsh for visualization.
- Ply file: '`.ply`'

  It contains a description of the tessellation written under the standard "Polygon File Format" '`.ply`'.
- 3DEC file: '`.3dec`'

  It contains a description of the tessellation written under the 3DEC format '`.3dec`'.
- Orientation file: '`.ori`'

  It contains crystal orientations for the tessellation cells. The orientations are written on successive lines, using the descriptor specified by option '`-oridescriptor`' (see also Section A.5 [Rotations and Orientations], page 44) and the writing convention specified by option '`-oriformat`'.

2.2.2 Statistics

Statistics files are provided for cells, germs, vertices, edges, faces, polyhedra and points. They are formatted with one entity per line. Each line contains the data specified to the corresponding '`-stat`' option and described in Section A.2 [Tessellation Keys], page 41 and Section A.3 [Raster Tessellation Keys], page 43 (files '`.stcell`' and '`.stgerm`' only).

- Tessellation cell statistics file, '`.stcell`'.
- Tessellation germ statistics file, '`.stgerm`'.
- Tessellation vertex statistics file, '`.stver`'.
- Tessellation edge statistics file, '`.stedge`'.
- Tessellation face statistics file, '`.stface`'.
- Tessellation polyhedron statistics file, '`.stpoly`'.
- Point statistics file, '`.stpoint`'.

2.3 Examples

Below are some examples of use of neper -T.

1. Generate a Voronoi tessellation containing 100 cells (with identifier = 1).

   ```
   $ neper -T -n 100 -id 1
   ```
2. Use an elongated domain and generate a Voronoi tessellation containing 100 cells.

   ```
   $ neper -T -n 100 -id 1 -domain "cube(3,1,0.33)"
   ```
3. Generate a Voronoi tessellation containing 100 cells and scale it to get elongated cells.

   ```
   $ neper -T -n 100 -id 1 -scale 3:1:0.33
   ```
4. Generate a 2-scale Voronoi tessellation containing 100×10 cells.

   ```
   $ neper -T -n 100::10 -id 1::1
   ```

5. Generate a Voronoi tessellation containing 100 cells and apply regularization.

   ```
   $ neper -T -n 100 -id 1 -reg 1
   ```

6. Generate a 2D Voronoi tessellation containing 100 cells.

   ```
   $ neper -T -n 100 -id 1 -dim 2
   ```

7. Generate a 2D Voronoi tessellation containing 100 cells at raster format with 50 points along each coordinate axis; apply filtering to prepare it for meshing.

   ```
   $ neper -T -n 100 -id 1 -dim 2 -format tesr -tesrsize 50 -filter size
   ```

8. Generate a Voronoi tessellation containing 100 cells and get, for each cell, its volume and its number of faces.

   ```
   $ neper -T -n 100 -id 1 -statcell vol,facenb
   ```

3 Meshing Module (-M)

Module -M is the module for meshing scalar and raster tessellations. Two meshing strategies are available. *Free (or unstructured) meshing* creates a conforming mesh into tetrahedral elements (triangular in 2D, line in 1D). *Mapped meshing* generates a non-conforming mesh into regular hexahedral elements (rectangular in 2D, line in 1D). Free meshing is carried out so that the elements have sizes as close as possible to a desired target value, and show high quality, that is, equilateral shapes. The input file is a tessellation file ('`.tess`') or a raster tessellation file ('`.tesr`'), as provided by module -T. Standard, 1-scale tessellations and 2-scale tessellations are supported. Free meshing of raster tessellations works for 1D and 2D tessellations only. The output mesh can be written in various formats.

The target element size of the mesh can be specified through the element *characteristic length ('`cl`')*. It stands for the length of a 1D element, the length of the edge of a triangle or quadrilateral element (2D) and the length of an edge of a tetrahedral or hexahedral element (3D). For convenience, a *relative characteristic length (`rcl`)* is also defined, whose value is relative to the average cell size and provides a medium number of elements. It is also possible to specify a `cl` (or `rcl`) value on a per-cell basis, or to specify different values along the three coordinate axes.

For free meshing, mesh quality is ensured to the greatest extent possible using several advanced capabilities,

- Optimized meshing rules. The mesh properties are controlled by size parameters (options '`-cl`', '`-rcl`', etc.) and a size gradient parameter used for 1D meshing (option '`-pl`').
- Multimeshing. Each tessellation face and volume is meshed separately of the others, with several meshing algorithms, until a target mesh quality is reached. This is controlled by options starting by '`-meshqual`', and options '`-mesh2dalgo`' and '`-mesh3dalgo`'.

Note that, in general, tessellation *regularization* is also necessary to ensure good-quality meshing, see Chapter 2 [Tessellation Module (-T)], page 7.

Remeshing can also be applied to generate a new, good-quality mesh from a mesh containing poor-quality elements. The variables defined on the old mesh can be transported on the new mesh (options starting by '`-transport`').

For mapped meshing, mesh cleaning options enable to remove isolated elements or duplicate nodes, or to duplicate nodes subjected to singularity behaviour (options '`-clean`', '`-dupnodemerge`' and '`-singnodedup`').

Mesh partitioning enables to divide the mesh nodes and elements into several sets while minimizing the interfaces between them[1], for parallel finite element calculations. Partitioning can return any number of partitions, or more efficiently, can be carried out according to a given parallel computer architecture (option '`-part`').

In the output mesh, the individual entities of the tessellations (the vertices, edges, faces and polyhedra) can be described by element sets (option '`-dim`'). Node sets of the vertices, edges and faces of the boundary of the tessellation are also provided for prescribing the boundary conditions (option '`-nset`'). The surface element sets are also provided (option '`-faset`'). The mesh order can be 1 or 2 (option '`-order`'). Statistical data can be obtained on the meshes (options starting by '`-stat`').

[1] Each partition being assigned to a processor in the finite element calculation, the minimization of the interfaces between the partitions is done in terms of the number of necessary communications between processors.

Here is what a typical run of module -M looks like,

```
$ neper -M n10-id1.tess

=======================    N   e   p   e   r    =======================
Info   : A software package for polycrystal generation and meshing.
Info   : Version 2.0.0
Info   : Built with: gsl libmatheval
Info   : Loading initialization file '/foo/bar/.neperrc'...
Info   : ---------------------------------------------------------------
Info   : MODULE  -M loaded with arguments:
Info   : [ini file] -gmsh /foo/bar/bin/gmsh
Info   : [com line] n10-id1.tess
Info   : ---------------------------------------------------------------
Info   : Reading input data...
Info   :   - Reading arguments...
Info   : Loading input data...
Info   :   - Loading tessellation...
Info   :     [i] Parsing file 'n10-id1.tess'...
Info   :     [i] Parsed file 'n10-id1.tess'.
Info   : Meshing...
Info   :   - Preparing... (cl = 0.2321) 100%
Info   :   - 0D meshing... 100%
Info   :   - 1D meshing... 100%
Info   :   - 2D meshing... 100% (0.69|0.86/96%| 4%| 0%)
Info   :     > Checking 2D mesh for pinching out...
Info   :   - 3D meshing... 100% (0.89|0.91/100%| 0%| 0%)
Info   :   - Searching nsets...
Info   : Writing mesh results...
Info   :   - Preparing mesh...
Info   :   - Mesh properties:
Info   :     > Node number:      289
Info   :     > Elt  number:      996
Info   :     > Mesh volume:    1.000
Info   :   - Writing mesh...
Info   :     [o] Writing file 'n10-id1.msh'...
Info   :     [o] Wrote file 'n10-id1.msh'.
Info   : Elapsed time: 8.414 secs.
===============================================================================
```

3.1 Arguments

3.1.1 Prerequisites

`-gmsh` *`path_name`* [Prerequisite]
Specify the path of the Gmsh binary (for meshing into triangle and tetrahedral elements).
Possible values: `any`. Default value: `gmsh`.

3.1.2 Input Data

In normal use, the input data is a tessellation file, a raster tessellation file or a mesh file,

`file_name` [Input data]
Name of the input file. It can be a tessellation file (‘`.tess`’), a raster tessellation file (‘`.tesr`’) or a mesh file for remeshing (‘`.msh`’). To load several of them (namely, both a tessellation file and a mesh file for remeshing), combine them with ‘`,`’. To overwrite the coordinates of the nodes of a mesh, use the syntax ‘*`file_name`*`:`*`nodecoo_file_name`*’, where ‘*`file_name`*’ is the name of the mesh file and ‘*`nodecoo_file_name`*’ is the name of the file containing the coordinates of the nodes (see Section B.3 [Position File], page 51). To load only a subregion of a raster tessellation, use the syntax ‘*`file_name`*`:crop(`*`xmin`*`|`*`xmax`*`|`*`ymin`*`|`*`ymax`*`|`*`zmin`*`|`*`zmax`*`)`’, where ‘*`xmin`*’, ‘*`xmax`*’, ‘*`ymin`*’, ‘*`ymax`*’, ‘*`zmin`*’ and ‘*`zmax`*’ are the minimum and maximum positions along x, y and z, respectively. For 2D raster tessellations, the z boundaries can be omitted. For 1D raster tessellations, the y and z boundaries can be omitted. To scale the number of points of a raster tessellation, use the syntax ‘*`file_name`*`:scale(`*`factor`*`)`’, where *`factor`* is the scaling factor, or ‘*`file_name`*`:scale(`*`factor_x`*`|`*`factor_y`*`|`*`factor_z`*`)`’, where *`factor_x`*, *`factor_y`* and *`factor_z`* are the scaling factor along x, y and z, respectively. For 2D raster tessellations, the z factor can be omitted. For 1D raster tessellations, the y and z factors can be omitted.
Possible values: `any`. Default value: `none`.

It is also possible to load a result mesh from a file. (Using option ‘`-o`’ along with this capability avoids overwriting the input data.)

`-loadmesh` *`file_name`* [Input data]
Load a mesh from a file (‘`.msh`’ format).
Possible values: `any`. Default value: `none`.

Finally, it is possible to load a set of points (useful for statistics, see option ‘`-statpoint`’),

`-loadpoint` *`file_name`* [Input data]
Load points from a file. See Section B.3 [Position File], page 51 for the file format. Provide as argument the file name.
Possible values: `any`. Default value: `none`.

3.1.3 Meshing Options

`-elttype` *`char_string`* [Option]
Type of elements, among tetrahedral (‘`tet`’) and hexahedral (‘`hex`’). (The 2D counterparts, ‘`tri`’ and ‘`quad`’, can also be used and are equivalent.)
Possible values: `tet, hex`. Default value: `tet`.

`-cl or -rcl` *`real`* [Option]
Absolute or relative characteristic length of the elements. `rcl` is defined relative to the average cell size. The default `-rcl 1` leads to a mesh with about 100 elements per cell in average (64 in 2D, 5 in 1D). For free meshing, it is also possible to get non-uniform characteristic

length distributions, as detailed in the following. To define a characteristic length on a per-cell basis, the first way is to use the syntax *default_cl*,*cell_expr1*:*cell_cl1*,*cell_expr2*:*cell_cl2*..., where *default_cl* is the default characteristic length, *cell_expri* is an expression defining the set of cells *i* and *cell_cli* is the corresponding characteristic length. '*cell_expri*' can be any expression based on variables provided in Section A.2 [Tessellation Keys], page 41 for tessellations, Section A.3 [Raster Tessellation Keys], page 43 for raster tessellations and Section A.4 [Mesh Keys], page 43 for meshes. The expressions are processed one after the other. When processing expression *cell_expri*, the matching cells are assigned *cell_cli* as characteristic length. A typical use is '`-rcl` *val1*`,body==0:`*val2*' to get interior cells meshed with *rcl*`=`*val1* and boundary cells meshed with *rcl*`=`*val2*. The second way is to load values from an external file using the syntax '`@`*file_name*', where *file_name* is the name of the file containing the characteristic length values.
Possible values: `any`. Default value: `-rcl 1`.

`-dim` *char_string* [Option]
Specify the meshing dimension. By default, it is equal to the input data dimension (`'inputdim'`). To get meshes of several dimensions in output, provide the values combined with '`,`'. Provide `'all'` for all and `'none'` for none. Note that the meshes of all dimensions are systematically written into a '`.msh`' mesh file unless '`:msh`' is appended to the option argument.
Possible values: `0 to 3, all, none, inputdim`. Default value: `inputdim`.

`-order` *integer* [Option]
Specify the mesh order. 1 means 2-node line elements, 3-node triangle elements, 4-node quadrangle elements, 4-node tetrahedral elements and 8-node hexahedral elements. 2 means 3-node line elements, 6-node triangle elements, 8-node quadrangle elements, 10-node tetrahedral elements and 20-node hexahedral elements.
Possible values: `1 or 2`. Default value: `1`.

`-pl` *real* [Secondary option]
Progression factor for the element characteristic lengths. This value is the maximum ratio between the lengths of two adjacent 1D elements.
Possible values: `any >= 1`. Default value: `2`.

`-clratio` *char_string* [Secondary option]
Specify ratios between the *cl*-values along the different coordinate axes. Provide the values combined with '`:`'. For example, '`2:1:1`' leads to elements twice as long in the x direction as in the y and z directions.
Possible values: `none`. Default value: `any`.

`-clmin` *real* [Secondary option]
Minimum characteristic length of the elements. Using this option is not recommended.
Possible values: `any`. Default value: `none`.

The following options define the multimeshing algorithm (for 2D and 3D free meshings). *Multimeshing* consists in using several meshing algorithms concurrently, for each face or polyhedron, until a minimum, target mesh quality is reached. The mesh quality factor, O, accounts for both the element sizes and aspect ratios. It is given by $O = {O_{dis}}^{\alpha} \times {O_{size}}^{1-\alpha}$, where O_{dis} and O_{size} range from 0 (poor quality) to 1 (high quality) and α is a factor equal to 0.8. Therefore, O also ranges from 0 (poor quality) to 1 (high quality). See the Neper reference paper for further information. The minimum quality value can be modified using option '`-meshqualmin`'. The values of O and O_{dis} can be modified using options '`-meshqualexpr`' and '`-meshqualdisexpr`'. The value of the target mesh quality significantly influences meshing speed and output mesh

quality. While a value of 0 provides fastest meshing, a value of 1 provides best-quality meshing. The default value provides an effective balance. The meshing algorithms are taken from the Gmsh[1] and Netgen[2] libraries (options '`-mesh2dalgo`' and '`-mesh3dalgo`').

`-meshqualmin` *`real`* [Option]
Specify the minimum, target value of mesh quality, O, as defined by option '`-meshqualexpr`'.
Possible values: `0 to 1`. Default value: `0.9`.

`-meshqualexpr` *`char_string`* [Option]
Specify the expression of mesh quality, O, as a function of *Odis* and *Osize*.
Possible values: `any`. Default value: `Odis^0.8*Osize^0.2`.

`-meshqualdisexpr` *`char_string`* [Secondary option]
Specify the expression of the mesh element distortion parameter, *Odis*, as a function of the element distortion parameter *dis* (see the Neper reference paper).
Possible values: `any`. Default value: `dis^(exp((dis^0.1)/(dis^0.1-1)))`.

`-mesh2dalgo` *`char_string`* [Secondary option]
Specify the 2D meshing algorithms (combine with '`,`'). The available values are `mead` (MeshAdapt), `dela` (Delaunay) and `fron` (Frontal).
Possible values: `mead, dela, fron`. Default value: `mead,dela,fron`.

`-mesh3dalgo` *`char_string`* [Secondary option]
Specify the 3D meshing algorithms (combine with '`,`'). Each algorithm has format '*`mesh`*`:`*`opti`*', where *`mesh`* and *`opti`* stand for the meshing and mesh optimization algorithms. The available values of *`mesh`* are currently limited to `netg` (Netgen). The available values of *`opti`* are '`gmsh`' (Gmsh), '`netg`' (Netgen) and '`gmne`' (Gmsh + Netgen). Use '`none`' for none.
Possible values: `netg:none, netg:gmsh, netg:netg, netg:gmne`. Default value: `netg:gmsh,netg:netg,netg:gmne`.

3.1.4 Raster Tessellation Meshing Options

Raster tessellation meshing implies interface reconstruction, interface mesh smoothing then remeshing. The following options enable to control interface smoothing.

`-tesrsmooth` *`char_string`* [Secondary option]
Method for smoothing the interface meshes reconstructed from raster tessellations. Laplacian smoothing ('`laplacian`') is an iterative method that consists in modifying the coordinates of a node using the coordinates of the neighbouring nodes. At iteration i, the position of a node, X_i, is calculated from its previous position, X_{i-1}, and the position of the barycentre of the neighbouring nodes (weighted barycentre, considering the inverse of the distance between the node and the neighbour), X^n_{i-1}, as follows: $X_i = (1 - A)\, X_{i-1} + A\, X^n_{i-1}$. $A \in [0, 1]$ an adjustable parameter (see option '`-tesrsmoothfact`'). The number of iterations is set by option '`-tesrsmoothitermax`'. There is no stop criterion, so `itermax` will always be reached.
Possible values: `laplacian or none`. Default value: `laplacian`.

`-tesrsmoothfact` *`real`* [Secondary option]
Factor used for the interface mesh smoothing (A in option '`-tesrsmooth`').
Possible values: `0 to 1`. Default value: `0.5`.

---

[1] *Ch. Geuzaine and J.-F. Remacle, Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities, International Journal for Numerical Methods in Engineering, 79, 1309–1331, 2009.*

[2] *J. Schoberl, Netgen, an advancing front 2d/3d-mesh generator based on abstract rules. Comput. Visual. Sci., 52, 1–41, 1997.*

`-tesrsmoothitermax` *integer* [Secondary option]
Number of iterations used for interface mesh smoothing.
Possible values: `any >= 0`. Default value: `5`.

3.1.5 Mesh Cleaning Options

The following options are specific to mapped meshing of raster tessellations containing voids.

`-clean` *integer* [Secondary option]
Clean the mesh so that it consists of a set of connected elements. Provide as argument the level of cleaning. A value of `1` indicates that two elements should be considered connected to each other if they share at least a vertex. A value of `2` indicates that two elements should be considered connected to each other if they share at least a face. Using this option, the elements (or bunches of elements) that are not connected to the main skeleton are removed.
Possible values: `0 to 2`. Default value: `0`.

`-singnodedup` *logical* [Secondary option]
Duplicate nodes which are the subject of singularity. Such a node belongs to several elements which share only a node or an edge, which provides a singularity behaviour. In Mechanics, it corresponds to imposing a common displacement, while the point can carry no stress. In Thermics, it corresponds to imposing a given temperature at a particular location shared by two bodies, but not enabling heat flux to operate at that location. When this option is enabled, such a node is duplicated, so that each body has its own node. Option '`-dupnodemerge`' enables to merge back duplicate nodes.
Possible values: `0 or 1`. Default value: `0`.

`-dupnodemerge` *real* [Secondary option]
Merge duplicate nodes. Provide as argument the distance between nodes below which two nodes are merged. Note that Neper does not generate meshes with duplicate nodes, except using option '`-singnodedup`'.
Possible values: `0`. Default value: `any>0`.

3.1.6 Mesh Partitioning Options

Mesh partitioning is achieved using the libScotch library[3]. The principle of mesh partitioning is to create partitions of same size while minimizing the interfaces between them. This affects the same load to all computation units and minimizes communications between them, therefore minimizes the total computation time. There are two available strategies for mesh partitioning. The first one creates partitions and arranges them independently of each other. The other one consists in optimizing the size and arrangment of the partitions based on a given computer cluster architecture to minimize computation time further. For those clusters that contain processors each with several cores, the communication time between cores of a processor is much lower than the communication time between cores of different processors. To minimize the global communication time, partitions which are processed by cores of the same processor can be grouped together. Partitioning is applied to the higher-dimension mesh. On top of defining the partitions, it renumbers the nodes and elements by increasing partition identifier and writes partitions as element and node sets ('`geof`' and '`inp`' formats). This can be managed using option '`-part`'.

`-part` *integer or char_string* [Option]
Specify the number of partitions or a computer cluster architecture. Using the first option, the number of partition can be any. At the opposite, for a computer cluster architecture, the

[3] *F. Pellegrini, Scotch and libScotch 5.1 User's Guide, INRIA Bordeaux Sud-Ouest, ENSEIRB & LaBRI, UMR CNRS 5800, 2008.*

total number of partitions must be a power of 2. An architecture can be specified in two ways. First, for clusters that contain processors each with several cores, the number of processors and the number of cores per processor can be combined using the '`:`' separator. A ratio of 10 is considered between the computation time between cores located on different processors and the one between cores of the same processor. Second, the name of a file describing the cluster architecture at the Scotch format can be provided.
Possible values: `any`. Default value: `none`.

`-partbalancing` *`real`* [Secondary option]
Provide the rate of element partition balancing. The partitioning algorithm applies to the nodes; the element partitions are determined afterwards and can be somewhat unbalanced. This option enables to enforce balancing. It is highly CPU-sensitive.
Possible values: `0 to 1`. Default value: `0.5`.

`-partmethod` *`char_string`* [Secondary option]
Specify the partitioning method. Provide the partitioning expression in Scotch's jargon, or '`none`' for none.
Possible values: `any`. Default value: `see_the_source`.

3.1.7 Field Transport Options

`-transport` *`char_string:char_string:file_name,...`* [Option]
Use this option for transporting data from a parent mesh to a child mesh (both 3D). The parent mesh is the input mesh and the child mesh is the result mesh (created by remeshing or loaded with '`-loadmesh`'). A transport entry must have format '*`entity_type`*`:`*`data_type`*`:`*`file_name`*', where '*`entity_type`*' must be '`elt`', '*`data_type`*' is the type of data, under format '`integer`*`X`*' or '`real`*`X`*', where *`X`* is the dimension, and *`file_name`* is the name of the file containing the parent data. For several data transports, combine the transport entries with '`,`'.
Possible values: `any`. Default value: `none`.

3.1.8 Output Options

`-o` *`file_name`* [Option]
Specify the output file name.
Possible values: `any`. Default value: `none`.

`-format` *`char_string`* [Option]
Specify the format of the output file(s). Mesh formats are: the Gmsh '`msh`', the Abaqus '`inp`', the Zset/Zébulon '`geof`' and the FEPX '`fepx`' (all of the '`.parms`', '`.mesh`', '`.surf`', '`.opt`' and '`.bcs`' files are written by default; to restrict the list, provide the extensions). The tessellation file format `tess` is also available. Combine arguments with '`,`'.
Possible values: `anyone of the above list`. Default value: `msh`.

`-nset` *`char_string`* [Option]
Specify the node sets to provide, among: `faces`, `edges`, `vertices` for all domain faces, edges and vertices, and `facebodies` and `edgebodies` for all face and edge bodies. Provide `all` for all and `none` for none. To get the node sets corresponding to individual entities, provide their labels. For a cuboidal domain, they are `[x-z][0,1]` for the domain faces, `[x-z][0,1][x-z][0,1]` for the edges, and `[x-z][0,1][x-z][0,1][x-z][0,1]` for the vertices. For a cylindrical domain, they are `z[0,1]` for the z faces, and `f[1,2,...]` for the faces on the circular part of the domain. For other domains, they are `f[1,2,...]` for the faces. For cylindrical and other types of domains, the edge and vertex labels are obtained from the face labels as for cuboidal domains. For a 2D mesh (generated from a 2D tessellation),

the labels are `[x-y][0,1]` for the edges and `[x-y][0,1][x-y][0,1]` for the vertices. For a 1D mesh (generated from a 1D tessellation), the labels are `x[0,1]` for the vertices. Append '`body`' to a label to get only the body nodes of the set. Combine labels with '`,`'.
Possible values: `see above`. Default value: `faces in 3D, edges in 2D and vertices in 1D`.

`-faset` *char_string* [Option]
Specify the domain surface meshes to provide. Use '`faces`' for all faces. To get the fasets corresponding to individual faces, provide their labels (see option '`-nset`'). Combine them with '`,`'. Provide `none` for none.
Possible values: `see above`. Default value: `none`.

3.1.9 Post-Processing Options

The following options provide statistics on the nodes ('`nodes`'), 0D elements ('`elt0d`') and element sets ('`elset0d`'), 1D elements ('`elt1d`') and element sets ('`elset1d`'), 2D elements ('`elt2d`') and element sets ('`elset2d`') and 3D elements ('`elt3d`') and element sets ('`elset3d`'). Also note that the '*`elt`*' and '*`elset`*' labels can be used in place of '`eltnd`' and '`elsetnd`', where *`n`* is the higher mesh dimension. This enables to use the same command whatever the higher mesh dimension is.

`-statnode` *char_string* [Post-processing]
Provide statistics on the nodes. Provide as argument the keys as described in Section A.4 [Mesh Keys], page 43 (combine with '`,`').
Possible values: `any`. Default value: `none`.
Result file: extension '`.stnode`'.

`-statelt0d` *char_string* [Post-processing]
Provide statistics on the 0D elements. Provide as argument the keys as described in Section A.4 [Mesh Keys], page 43 (combine with '`,`').
Possible values: `any`. Default value: `none`.
Result file: extension '`.stelt0d`'.

`-statelt1d` *char_string* [Post-processing]
Provide statistics on the 1D elements. Provide as argument the keys as described in Section A.4 [Mesh Keys], page 43 (combine with '`,`').
Possible values: `any`. Default value: `none`.
Result file: extension '`.stelt1d`'.

`-statelt2d` *char_string* [Post-processing]
Provide statistics on the 2D elements. Provide as argument the keys as described in Section A.4 [Mesh Keys], page 43 (combine with '`,`').
Possible values: `any`. Default value: `none`.
Result file: extension '`.stelt2d`'.

`-statelt3d` *char_string* [Post-processing]
Provide statistics on the 3D elements. Provide as argument the keys as described in Section A.4 [Mesh Keys], page 43 (combine with '`,`').
Possible values: `any`. Default value: `none`.
Result file: extension '`.stelt3d`'.

`-statelset0d` *char_string* [Post-processing]
Provide statistics on the 0D element sets. Provide as argument the keys as described in Section A.4 [Mesh Keys], page 43 (combine with '`,`').
Possible values: `any`. Default value: `none`.
Result file: extension '`.stelset0d`'.

`-statelset1d` *char_string* [Post-processing]
Provide statistics on the 1D element sets. Provide as argument the keys as described in Section A.4 [Mesh Keys], page 43 (combine with '`,`').
Possible values: `any`. Default value: `none`.
Result file: extension '`.stelset1d`'.

`-statelset2d` *char_string* [Post-processing]
Provide statistics on the 2D element sets. Provide as argument the keys as described in Section A.4 [Mesh Keys], page 43 (combine with '`,`').
Possible values: `any`. Default value: `none`.
Result file: extension '`.stelset2d`'.

`-statelset3d` *char_string* [Post-processing]
Provide statistics on the 3D element sets. Provide as argument the keys as described in Section A.4 [Mesh Keys], page 43 (combine with '`,`').
Possible values: `any`. Default value: `none`.
Result file: extension '`.stelset3d`'.

`-statpoint` *char_string* [Post-processing]
Provide statistics on points. The points must be loaded with option '`-loadpoint`'. Provide as argument the keys as described in Section A.4 [Mesh Keys], page 43 (combine with '`,`').
Possible values: `any`. Default value: `none`.
Result file: extension '`.stpoint`'.

3.1.10 Advanced Options

These advanced options set running conditions for the meshing libraries (triangle and tetrahedral meshing),

`-mesh3dclconv` *real* [Secondary option]
Maximum difference between the characteristic length `cl` and the average element length (for each polyhedron). Neper tries its best to get the average element size to match `cl`. Use this option to change the tolerance on the relative difference between the two. This is a highly CPU-sensitive capability (increasing this value can be efficient to speed up meshing).
Possible values: `any`. Default value: `0.02`.

`-mesh2dmaxtime` *real* [Secondary option]
Maximum processing time allowed to the meshing library for meshing a tessellation face (in seconds).
Possible values: `any`. Default value: `1000`.

`-mesh2drmaxtime` *real* [Secondary option]
This option is similar to '`-mesh2dmaxtime`', but the actual maximum time is the product of the maximum processing time of the previous meshings by the value provided in argument.
Possible values: `any`. Default value: `100`.

`-mesh2diter` *integer* [Secondary option]
Maximum 2D meshing attempts for a particular face (in case of failure).
Possible values: `any`. Default value: `3`.

`-mesh3dmaxtime` *real* [Secondary option]
Maximum processing time allowed to the meshing library for meshing a tessellation volume (in seconds).
Possible values: `any`. Default value: `1000`.

`-mesh3drmaxtime` *real* [Secondary option]
This option is similar to '`-mesh3dmaxtime`', but the actual maximum time is the product of the maximum processing time of the previous meshings by the value provided in argument. Possible values: `any`. Default value: `100`.

`-mesh3diter` *integer* [Secondary option]
Maximum 3D meshing attempts for a particular volume (in case of failure). Possible values: `any`. Default value: `3`.

3.2 Output Files

3.2.1 Mesh

The mesh can be written in the following formats,

- Gmsh format: file '`.msh`'
- Abaqus format: file '`.inp`'
- Zset/Zébulon format: file '`.geof`'
- FEPX format: files '`.parms`', '`.mesh`', '`.surf`', '`.opt`' and '`.bcs`'

3.2.2 Statistics

Statistics files are provided for nodes, elements, element sets and points. They are formatted with one entity per line. Each line contains the data specified to the corresponding '`-stat`' option and described in Section A.4 [Mesh Keys], page 43.

- Node statistics file, '`.stnode`'.
- (Higher-dimension) element statistics file, '`.stelt`'.
- (Higher-dimension) element set statistics file, '`.stelset`'.
- 0D element statistics file, '`.stelt0d`'.
- 1D element statistics file, '`.stelt1d`'.
- 2D element statistics file, '`.stelt2d`'.
- 3D element statistics file, '`.stelt3d`'.
- 0D element set statistics file, '`.stelset0d`'.
- 1D element set statistics file, '`.stelset1d`'.
- 2D element set statistics file, '`.stelset2d`'.
- 3D element set statistics file, '`.stelset3d`'.
- Point statistics file, '`.stpoint`'.

3.3 Examples

Below are some examples of use of neper -M,

1. Mesh tessellation '`n100-id1.tess`'.
```
$ neper -M n100-id1.tess
```
2. Mesh 2D raster tessellation '`n100-id1.tesr`'.
```
$ neper -M n100-id1.tesr
```
3. Mesh tessellation '`n100-id1.tess`' with a mesh size of rcl = 0.5 and in 2nd-order elements.
```
$ neper -M n100-id1.tess -rcl 0.5 -order 2
```
4. Mesh tessellation '`n100-id1.tess`' with small elements for the interior cells and bigger elements for the boundary cells.
```
$ neper -M n100-id1.tess -rcl "0.2,body==0:0.5"
```

5. Remesh mesh '`n150_def.msh`' (comprising poor-quality elements) into a clean, new mesh. Transport the scalar data of file '`n150_def.data`' from the deformed mesh to the new mesh.

```
$ neper -M n150.tess,n150_def.msh -transport elt:real1:n150_def.data
  -rcl 0.5 -o n150_new
```

6. Mesh tessellation '`n100-id1.tess`' and partition the mesh into 8 partitions.

```
$ neper -M n100-id1.tess -part 8
```

7. Mesh tessellation '`n100-id1.tess`' into regular hexahedral elements (non-conformal mesh).

```
$ neper -M n100-id1.tess -elt hex
```

8. Mesh tessellation '`n100-id1.tess`' and get, for each element, its radius ratio and its volume.

```
$ neper -M n100-id1.tess -statelt rr,vol
```

4 Visualization Module (-V)

Module -V is the Neper visualization module, with which the tessellations and meshes can be printed as publication-quality images. It is also possible to visualize data on them using colours and transparency, or displacements of the nodes and to plot data on slices of the mesh. The output is a PNG image file. The POV-Ray ray-tracing library is used for generating the images.

Contrary to other modules, module -V processes the command arguments one after the other. Typically, using module -V consists in loading a tessellation and / or a mesh, then data fields to render them. The data can apply to the tessellation entities: polyhedra, faces, edges and vertices, and to the mesh entities: 3D elements, 2D elements, 1D elements, 0D elements and nodes (options starting by '`-data`'). The entities that are to be visible, for example particular tessellation cells, element sets or elements, can also be specified (options starting by '`-show`'). The way they are plotted: camera position and angle, projection type, image size, etc., can be set up too (options starting by '`-camera`' or '`-image`').

Here is what a typical run of module -V looks like,

```
$ neper -V n10-id1.tess,n10-id1.msh -dataelsetcol id -print img

=======================    N   e   p   e   r    =======================
Info   : A software package for polycrystal generation and meshing.
Info   : Version 2.0.0
Info   : Built with: gsl libmatheval
Info   : Loading initialization file '/foo/bar/.neperrc'...
Info   : ---------------------------------------------------------------
Info   : MODULE  -V loaded with arguments:
Info   : [ini file]
Info   : [com line] n10-id1.tess,n10-id1.msh -dataelsetcol id -print img
Info   : ---------------------------------------------------------------
Info   : Loading tessellation...
Info   :     [i] Parsing file 'n10-id1.tess'...
Info   :     [i] Parsed file 'n10-id1.tess'.
Info   : Loading mesh...
Info   :     [i] Parsing file 'n10-id1.msh'...
Info   :     [i] Parsed file 'n10-id1.msh'.
Info   : Reconstructing mesh...
Info   : Reading data (elset3d, col)...
Info   : Printing image...
Info   :     [o] Writing file 'img.pov'...
Info   :   - Printing mesh...
Info   :     > Reducing data...
Info   :       . Number of 3D elt faces reduced by 89% (to 422).
Info   :       . Number of 3D elt edges reduced by 50% (to 633).
Info   :       . Number of 0D elts    reduced by 100% (to 0).
Info   :     [o] Wrote file 'img.pov'.
Info   :   - Generating png file (1200x900 pixels)...
Info   :     [o] Writing file 'img.png'...
Info   :     [o] Wrote file 'img.png'.
Info   : Printing scale...
Info   : Elapsed time: 0.759 secs.
=========================================================================
```

4.1 Arguments

4.1.1 Input Data

file_name [Input data]
Name of the input file. It can be a tessellation file ('`.tess`'), a raster tessellation file ('`.tesr`') or a mesh file ('`.msh`'). To load several of them, combine them with '`,`'.
Possible values: `any`. Default value: `none`.

4.1.2 Tessellation Data Loading and Rendering

The following options enable to define the properties (colour and size) of the tessellation cells or entities (polyhedra, faces, edges and vertices). This can be done either directly, by specifying the property values (e.g. the RGB channel values for colour) or indirectly, e.g. using scalar values that are rendered in colour using a given *colour scheme*. In this case, a scale image is generated in addition to the tessellation image. The scale properties can be set up (minimum, maximum and tick values).

The following options apply to the cells of a tessellation or a raster tessellation, independently of its dimension,

`-datacellcol` *char_string* [Option]
Set the colours of the tessellation cells. The argument can be one of the following: *(i)* '`id`' for colouring based on the identifier, using a colour palette (see Section A.6 [Colours], page 44), *(ii)* '`ori`' for colouring based on the crystal orientations, *(iii)* the name of a colour that will be used for all cells (see Section A.6 [Colours], page 44), *(iv)* the name of a file containing a list of colours (provided as RGB channel values), or *(v)* a string indicating how the colours can be obtained. The string has the format '*var*`:`*file_name*', where *var* can be '`ori`' for crystal orientations or '`scal`' for scalar values, and '*file_name*' is the name of the file containing the data. The colour schemes used to derive the colours from the data can be specified with option '`-datacellcolscheme`'.
Possible values: `any`. Default value: `white`.

`-datacellcolscheme` *char_string* [Option]
Set the colour scheme used to get colours from the data of the tessellation cells loaded with option '`-datacellcol`'. The type of colour scheme depends on the type of data. For crystal orientations, the colour scheme can be: '`R`' for Rodrigues vector colouring; for scalar data, the colour scheme can be any list of colours.
Possible values: `"R" for crystal orientations and any list of colours for scalars`. Default value: `"R" for crystal orientations and "blue,cyan,yellow,green" for scalars`.

`-datacellltrs` *real* [Option]
Set the transparency of the tessellation cells. It ranges between 0 (no transparency) to 1 (full transparency).
Possible values: `0 to 1`. Default value: `0`.

`-datacellscale` *char_string* [Option]
Set the scale relative to the '`-datacellcol scal`' data. Provide as argument the minimum and maximum values combined with '`:`'. To specify the tick values, provide as argument the minimum, the intermediate tick values then the maximum, combined with '`:`'.
Possible values: `any`. Default value: `data minimum:data maximum`.

For tessellations, it is also possible to set data on an entity-basis,

`-datapolycol` *`char_string`* [Option]
Set the colours of the tessellation polyhedra. The argument can be one of the following: *(i)* '`id`' for colouring based on the identifier, using a colour palette (see Section A.6 [Colours], page 44), *(ii)* the name of a colour that will be used for all polyhedra (see Section A.6 [Colours], page 44), *(iii)* the name of a file containing a list of colours (provided as RGB channel values), or *(iv)* a string indicating how the colours can be obtained. The string has the format '*`var:file_name`*', where *`var`* can be '`ori`' for crystal orientations or '`scal`' for scalar values, and '*`file_name`*' is the name of the file containing the data. The colour schemes used to derive the colours from the data can be specified with option '`-datapolycolscheme`'.
Possible values: `any`. Default value: `white`.

`-datapolycolscheme` *`char_string`* [Option]
Set the colour scheme used to get colours from the data of the tessellation polyhedra loaded with option '`-datapolycol`'. The type of colour scheme depends on the type of data. For crystal orientations, the colour scheme can be: '`R`' for Rodrigues vector colouring; for scalar data, the colour scheme can be any list of colours.
Possible values: `"R" for crystal orientations and any list of colours for scalars`. Default value: `"R" for crystal orientations and "blue,cyan,yellow,green" for scalars`.

`-datapolytrs` *`real`* [Option]
Set the transparency of the tessellation polyhedra. It ranges between 0 (no transparency) to 1 (full transparency).
Possible values: `0 to 1`. Default value: `0`.

`-datapolyscale` *`char_string`* [Option]
Set the scale relative to the '`-datapolycol scal`' data. Provide as argument the minimum and maximum values combined with '`:`'. To specify the tick values, provide as argument the minimum, the intermediate tick values then the maximum, combined with '`:`'.
Possible values: `any`. Default value: `data minimum:data maximum`.

`-datafacecol` *`char_string`* [Option]
Set the colours of the tessellation faces. See option '`-datapolycol`' for the argument format.
Possible values: `any`. Default value: `white`.

`-datafacecolscheme` *`char_string`* [Option]
Set the colour scheme used to get colours from the data of the tessellation faces loaded with option '`-datafacecol`'. See option '`-datapolycolscheme`' for the argument format.
Possible values: `see option '-datapolycolscheme'.` Default value: `see option '-datapolycolscheme'.`

`-datafacetrs` *`real`* [Option]
Set the transparency of the tessellation faces. It ranges between 0 (no transparency) to 1 (full transparency).
Possible values: `0 to 1`. Default value: `0`.

`-datafacescale` *`char_string`* [Option]
Set the scale relative to the '`-datafacecol scal`' data. Provide as argument the minimum and maximum values combined with '`:`'. To specify the tick values, provide as argument the minimum, the intermediate tick values then the maximum, combined with '`:`'.
Possible values: `any`. Default value: `data minimum:data maximum`.

`-dataedgerad` *`char_string`* [Option]
Set the radii of the tessellation edges. The argument can be one of the following: a real value that will be used for all entities or the name of a file containing a list of radii.
Possible values: `any`. Default value: `tessellation dependent`.

`-dataedgecol` *char_string* [Option]
Set the colours of the tessellation edges. See option '`-datapolycol`' for the argument format.
Possible values: `any`. Default value: `black`.

`-dataedgetrs` *real* [Option]
Set the transparency of the tessellation edges. It ranges between 0 (no transparency) to 1 (full transparency).
Possible values: `0 to 1`. Default value: `0`.

`-dataedgecolscheme` *char_string* [Option]
Set the colour scheme used to get colours from the data of the tessellation edges loaded with option '`-dataedgecol`'. See option '`-datapolycolscheme`' for the argument format.
Possible values: `see option '-datapolycolscheme'`. Default value: `see option '-datapolycolscheme'`.

`-dataedgescale` *char_string* [Option]
Set the scale relative to the '`-dataedgecol scal`' data. Provide as argument the minimum and maximum values combined with '`:`'. To specify the tick values, provide as argument the minimum, the intermediate tick values then the maximum, combined with '`:`'.
Possible values: `any`. Default value: `data minimum:data maximum`.

`-dataverrad` *char_string* [Option]
Set the radii of the tessellation vertices. See option '`-dataedgerad`' for the argument format.
Possible values: `any`. Default value: `tessellation dependent`.

`-datavercol` *char_string* [Option]
Set the colours of the tessellation vertices. See option '`-datapolycol`' for the argument format.
Possible values: `any`. Default value: `black`.

`-datavercolscheme` *char_string* [Option]
Set the colour scheme used to get colours from the data of the tessellation vertices loaded with option '`-datavercol`'. See option '`-datapolycolscheme`' for the argument format.
Possible values: `see option '-datapolycolscheme'`. Default value: `see option '-datapolycolscheme'`.

`-dataverscale` *char_string* [Option]
Set the scale relative to the '`-datavercol scal`' data. Provide as argument the minimum and maximum values combined with '`:`'. To specify the tick values, provide as argument the minimum, the intermediate tick values then the maximum, combined with '`:`'.
Possible values: `any`. Default value: `data minimum:data maximum`.

`-datagermrad` *char_string* [Option]
Set the radii of the tessellation germs. See option '`-dataedgerad`' for the argument format.
Possible values: `any`. Default value: `tessellation dependent`.

`-datagermcol` *char_string* [Option]
Set the colours of the tessellation germs. See option '`-datapolycol`' for the argument format.
Possible values: `any`. Default value: `grey`.

`-datagermcolscheme` *char_string* [Option]
Set the colour scheme used to get colours from the data of the tessellation germs loaded with option '`-datagermcol`'. See option '`-datapolycolscheme`' for the argument format.
Possible values: `see option '-datapolycolscheme'`. Default value: `see option '-datapolycolscheme'`.

`-datagermscale` *char_string* [Option]
Set the scale relative to the '`-datagermcol scal`' data. Provide as argument the minimum and maximum values combined with '`:`'. To specify the tick values, provide as argument the minimum, the intermediate tick values then the maximum, combined with '`:`'.
Possible values: `any`. Default value: `data minimum:data maximum`.

Below are options specific to raster tessellations,

`-datarptedgecol` *char_string* [Option]
Set the colour of the edges of the raster points. Provide as argument the name of a colour that will be used for all points (see Section A.6 [Colours], page 44).
Possible values: `any`. Default value: `black`.

`-datarptedgerad` *real* [Option]
Set the radius of the edges of the raster points.
Possible values: `any`. Default value: `proportional to the raster point size`.

4.1.3 Mesh Data Loading and Rendering

The following options enable to define the properties (colour, size, etc.) of the mesh entities (3D, 2D, 1D and 0D elements, and nodes). This can be done either directly, by specifying the property values (e.g. the RGB channel values for colour) or indirectly, e.g. using scalar values that are rendered in colour using a given *colour scheme*. In this case, a scale image is generated in addition to the mesh image. The scale properties can be set up (minimum, maximum and tick values).

The options are listed below for 3D elements ('`elt3d`') and element sets ('`elset3d`'), 2D elements ('`elt2d`') and element sets ('`elset2d`'), 1D elements ('`elt1d`') and element sets ('`elset1d`'), 0D elements ('`elt0d`') and element sets ('`elset0d`') and nodes ('`nodes`'). Also note that the '*`elt`*' and '*`elset`*' labels can be used in place of '`eltnd`' and '`elsetnd`', where *`n`* is the highest mesh dimension. This enables to use the same command whatever the highest mesh dimension is.

The following options enable to load data relative to the 3D mesh elements. Note that the options can be applied to element sets by changing '`elt`' to '`elset`'.

`-dataelt3dcol` *char_string* [Option]
Set the colours of the 3D elements. The argument can be one of the following: *(i)* '`id`' for the default colour palette (see Section A.6 [Colours], page 44), *(ii)* the name of a colour that will be used for all elements (see Section A.6 [Colours], page 44), *(iii)* the name of a file containing a list of colours (provided as RGB channel values), *(iv)* a string indicating how the colours can be obtained, or *(v)* '`from_nodes`' to derive the colours of the elements from the colours of the nodes (the node colours must be loaded using option '`-datanodecol`'). In case *(iv)*, the string has the format '*`var:file_name`*', where *`var`* can be '`ori`' for crystal orientations or '`scal`' for scalar values, and '*`file_name`*' is the name of the file containing the data. The colour schemes used to derive the colours from the data can be specified with option '`-dataelt3dcolscheme`'.
Possible values: `any`. Default value: `white`.

`-dataelt3dcolscheme` *char_string* [Option]
Set the colour scheme used to get colours from the data of the 3D elements loaded with option '`-dataelt3dcol`'. The type of colour scheme depends on the type of data. For crystal orientations, the colour scheme can be: '`R`' for Rodrigues vector colouring; for scalar data, the colour scheme can be any list of colours.
Possible values: `"R" for crystal orientations and any list of colours for scalars`. Default value: `"R" for crystal orientations and "blue,cyan,yellow,green" for scalars`.

`-dataelt3dscale` *char_string* [Option]
Set the scale relative to the '`-dataelt3dcol scal`' data. Provide as argument the minimum and maximum values combined with '`:`'. To specify the tick values, provide as argument the minimum, the intermediate tick values then the maximum, combined with '`:`'.
Possible values: `any`. Default value: `data minimum:data maximum`.

`-dataelt3dedgecol` *char_string* [Option]
Set the colour of the edges of the 3D elements. Provide as argument the name of a colour that will be used for all elements (see Section A.6 [Colours], page 44).
Possible values: `any`. Default value: `black`.

`-dataelt3dedgerad` *real* [Option]
Set the radius of the edges of the 3D elements.
Possible values: `any`. Default value: `mesh dependent`.

The following options enable to load data relative to the 2D elements. Note that the options can be applied to element sets by changing '`elt`' to '`elset`'.

`-dataelt2dcol` *char_string* [Option]
Set the colours of the 2D elements. See option '`-dataelt3dcol`' for the argument format.
Possible values: `any`. Default value: `white`.

`-dataelt2dcolscheme` *char_string* [Option]
Set the colour scheme used to get colours from the data of the 2D elements loaded with option '`-dataelt2dcol`'. See option '`-dataelt3dcolscheme`' for the argument format.
Possible values: `see option '-dataelt3dcolscheme'`. Default value: `see option '-dataelt3dcolscheme'`.

`-dataelt2dscale` *char_string* [Option]
Set the scale relative to the '`-dataelt2dcol scal`' data. Provide as argument the minimum and maximum values combined with '`:`'. To specify the tick values, provide as argument the minimum, the intermediate tick values then the maximum, combined with '`:`'.
Possible values: `any`. Default value: `data minimum:data maximum`.

`-dataelt2dedgecol` *char_string* [Option]
Set the colours of the edges of the 3D elements. See option '`-dataelt3dedgecol`' for the argument format.
Possible values: `any`. Default value: `black`.

`-dataelt2dedgerad` *real* [Option]
Set the radius of the edges of the 2D elements.
Possible values: `any`. Default value: `mesh dependent`.

The following options enable to load data relative to the 1D elements. Note that the options can be applied to element sets by changing '`elt`' to '`elset`'.

`-dataelt1dcol` *char_string* [Option]
Set the colours of the 1D elements. See option '`-dataelt3dcol`' for the argument format.
Possible values: `any`. Default value: `black`.

`-dataelt1dcolscheme` *char_string* [Option]
Set the colour scheme used to get colours from the data of the 1D elements loaded with option '`-dataelt1dcol`'. See option '`-dataelt3dcolscheme`' for the argument format.
Possible values: `see option '-dataelt3dcolscheme'`. Default value: `see option '-dataelt3dcolscheme'`.

`-dataelt1dscale` *char_string* [Option]
Set the scale relative to the '`-dataelt1dcol scal`' data. Provide as argument the minimum and maximum values combined with '`:`'. To specify the tick values, provide as argument the minimum, the intermediate tick values then the maximum, combined with '`:`'.
Possible values: `any`. Default value: `data minimum:data maximum`.

`-dataelt1drad` *char_string* [Option]
Set the radii of the 1D element. See option '`-dataelt3dedgerad`' for the argument format.
Possible values: `any`. Default value: `mesh dependent`.

The following options enable to load data relative to the 0D mesh elements. Note that the options can be applied to element sets by changing '`elt`' to '`elset`'.

`-dataelt0dcol` *char_string* [Option]
Set the colours of the 0D elements. See option '`-dataelt3dcol`' for the argument format.
Possible values: `any`. Default value: `black`.

`-dataelt0dcolscheme` *char_string* [Option]
Set the colour scheme used to get colours from the data of the 0D elements loaded with option '`-dataelt0dcol`'. See option '`-dataelt3dcolscheme`' for the argument format.
Possible values: `see option '-dataelt3dcolscheme'`. Default value: `see option '-dataelt3dcolscheme'`.

`-dataelt0dscale` *char_string* [Option]
Set the scale relative to the '`-dataelt0dcol scal`' data. Provide as argument the minimum and maximum values combined with '`:`'. To specify the tick values, provide as argument the minimum, the intermediate tick values then the maximum, combined with '`:`'.
Possible values: `any`. Default value: `data minimum:data maximum`.

`-dataelt0drad` *char_string* [Option]
Set the radii of the 0D element. See option '`-dataelt3dedgerad`' for the argument format.
Possible values: `any`. Default value: `mesh dependent`.

The following options enable to load data relative to the nodes.

`-datanodecoo` *char_string* [Option]
Set the coordinates of the nodes. The argument can be the name of a file containing a list of coordinates, or a string indicating how the coordinates can be obtained. The string has the format '*`var`*`:file_name`', where *`var`* can be '`disp`' for displacements, and '`file_name`' is the name of the file containing the data.
Possible values: `any`. Default value: `none`.

`-datanodecoofact` *real* [Option]
Set the value of the scaling factor to apply to the displacements of the nodes.
Possible values: `any`. Default value: `1`.

`-datanodecol` *file_name* [Option]
Set the colours of the nodes. See option '`-dataelt3dcol`' for the argument format.
Possible values: `any`. Default value: `black`.

`-datanodecolscheme` *char_string* [Option]
Set the colour scheme used to get colours from the data of the nodes loaded with option '`-datanodecol`'. See option '`-dataelt3dcolscheme`' for the argument format.
Possible values: `see option '-dataelt3dcolscheme'`. Default value: `see option '-dataelt3dcolscheme'`.

`-datanodescale` *char_string* [Option]
Set the scale relative to the '`-datanodecol scal`' data. Provide as argument the minimum and maximum values combined with '`:`'. To specify the tick values, provide as argument the minimum, the intermediate tick values then the maximum, combined with '`:`'.
Possible values: `any`. Default value: `data minimum:data maximum`.

`-datanoderad` *file_name* [Option]
Set the radii of the nodes. See option '`-dataeltedgerad`' for the argument format.
Possible values: `any`. Default value: `mesh dependent`.

4.1.4 Slice Settings

`-slicemesh` *char_string* [Option]
Use this option to plot one (or several) slice(s) of the mesh. Provide as argument the equation(s) of the plane(s), under the form *`dir=val`*, where *`dir`* can be '`x`', '`y`' or '`z`' and *`val`* is a real number (combine with '`,`').
Possible values: `any`. Default value: `none`.

4.1.5 Show Settings

The following options apply to the full tessellations or mesh.

`-showtess` *logical* [Option]
Use this option to show or hide the tessellation.
Possible values: `0 or 1`. Default value: `1 if tess loaded and no mesh`.

`-showtesr` *logical* [Option]
Use this option to show or hide the raster tessellation.
Possible values: `0 or 1`. Default value: `1 if tesr loaded and no mesh`.

`-showmesh` *logical* [Option]
Use this option to show or hide the mesh.
Possible values: `0 or 1`. Default value: `1 if mesh loaded and no slice`.

`-showmeshslice` *logical* [Option]
Use this option to show or hide the mesh slice(s).
Possible values: `0 or 1`. Default value: `1 if existing slice(s)`.

The following option applies to the cells of a tessellation or a raster tessellation, independently of its dimension,

`-showcell` *char_string* [Option]
Specify the cells to show. The argument can be: '`all`' for all, '`none`' for none, '*`@file_name`*' to load polyhedron identifiers from a file, or any expression based on the keys listed in Section A.2 [Tessellation Keys], page 41 or Section A.3 [Raster Tessellation Keys], page 43.
Possible values: `any`. Default value: `all`.

For tessellations, it is also possible to set visibility on an entity-basis,

`-showpoly` *char_string* [Option]
Specify the polyhedra to show. The argument can be: '`all`' for all, '`none`' for none, '*`@file_name`*' to load polyhedron identifiers from a file, or any expression based on the keys listed in Section A.2 [Tessellation Keys], page 41.
Possible values: `any`. Default value: `all`.

`-showface` *char_string* [Option]
Specify the faces to show. The argument can be: '`all`' for all, '`none`' for none, '`@`*file_name*' to load face identifiers from a file, or any expression based on the keys listed in Section A.2 [Tessellation Keys], page 41. The following specific keys are also available: '`cell_shown`' and '`poly_shown`'.
Possible values: `any`. Default value: `none`.

`-showedge` *char_string* [Option]
Specify the edges to show. The argument can be: '`all`' for all, '`none`' for none, '`@`*file_name*' to load edge numbers from a file, or any expression based on the keys listed in Section A.2 [Tessellation Keys], page 41. The following specific keys are also available: '`cell_shown`', '`poly_shown`' and '`face_shown`'.
Possible values: `any`. Default value: `cell_shown`.

`-showver` *char_string* [Option]
Specify the vertices to show. The argument can be: '`all`' for all, '`none`' for none, '`@`*file_name*' to load vertex numbers from a file, or any expression based on the keys listed in Section A.2 [Tessellation Keys], page 41. The following specific keys are also available: '`cell_shown`', '`poly_shown`', '`face_shown`' and '`edge_shown`'.
Possible values: `any`. Default value: `none`.

`-showgerm` *char_string* [Option]
Specify the germs to show. The argument can be: '`all`' for all, '`none`' for none, '`@`*file_name*' to load germ numbers from a file, or any expression based on the keys listed in Section A.2 [Tessellation Keys], page 41. The following specific key is also available: '`cell_shown`'.
Possible values: `any`. Default value: `none`.

`-showfaceinter` *logical* [Secondary option]
Show the interpolations of the tessellation faces (if any). The interpolation edges are printed in grey with a radius equal to the radius of the face edges.
Possible values: `0 or 1`. Default value: `0`.

The following options apply to the entities of the mesh. The options apply to 3D elements ('`elt3d`') and element sets ('`elset3d`'), 2D elements ('`elt2d`') and element sets ('`elset2d`'), 1D elements ('`elt1d`') and element sets ('`elset1d`'), 0D elements ('`elt0d`') and element sets ('`elset0d`'), and nodes ('`nodes`'). Also note that the '`elt`' and '`elset`' labels can be used in place of '`elt`*n*`d`' and '`elset`*n*`d`', where *n* is the highest mesh dimension. This enables to use the same command whatever the highest mesh dimension is.

In the following option descriptions, note that any options can be applied to element *sets* by changing '`elt`' to '`elset`'.

`-showelt3d` *char_string* [Option]
Specify the 3D elements to show. The argument can be: '`all`' for all, '`none`' for none, '`@`*file_name*' to load element identifiers from a file, or any expression based on the keys listed in Section A.4 [Mesh Keys], page 43.
Possible values: `any`. Default value: `all if highest mesh dim. is 3 and none otherwise`.

`-showelt2d` *char_string* [Option]
Specify the 2D elements to show. The argument can be: '`all`' for all, '`none`' for none, '`@`*file_name*' to load element identifiers from a file, or any expression based on the keys listed in Section A.4 [Mesh Keys], page 43. The following specific key is also available: '`elt3d_shown`'.
Possible values: `any`. Default value: `all if highest mesh dim. is 2 and none otherwise`.

`-showelt1d` *char_string* [Option]
Specify the 1D elements to show. The argument can be: '`all`' for all , '`none`' for none, '`@`*file_name*' to load element numbers from a file, or any expression based on the keys listed in Section A.4 [Mesh Keys], page 43. The following specific keys are also available: '`elt2d_shown`' and '`elt3d_shown`'.
Possible values: `any`. Default value: `all if highest mesh dim. is 1 and none otherwise`.

`-showelt0d` *char_string* [Option]
Specify the 0D elements to show. The argument can be: '`all`' for all '`none`' for none, '`@`*file_name*' to load element numbers from a file, or any expression based on the keys listed in Section A.4 [Mesh Keys], page 43. The following specific keys are also available: '`elt1d_shown`', '`elt2d_shown`' and '`elt3d_shown`'.
Possible values: `any`. Default value: `all if highest mesh dim. is 0 and none otherwise`.

`-shownode` *char_string* [Option]
Specify the nodes to show. The argument can be: '`all`' for all , '`none`' for none, '`@`*file_name*' to load node numbers from a file, or any expression based on the keys listed in Section A.4 [Mesh Keys], page 43. The following specific keys are also available: '`elt0d_shown`', '`elt1d_shown`', '`elt2d_shown`' and '`elt3d_shown`'.
Possible values: `any`. Default value: `none`.

`-showshadow` *logical* [Option]
Show the shadows. If you want colours not affected by shadowing, switch this option off.
Possible values: `0 or 1`. Default value: `1 in 3D and 0 in 2D and 1D`.

4.1.6 Camera Settings

`-cameracoo` *char_string:char_string:char_string* [Option]
Specify the camera coordinates. By default, the camera is shifted by a vector v from the tessellation or mesh centre. The coordinates of vector v are denoted as vx, vy and vz (= 3.462, -5.770 and 4.327, respectively, in 3D and 0, 0 and 8, respectively, in 2D and 1D). The coordinates of the tessellation or mesh centre are denoted as x, y and z (if both a tessellation and a mesh have been loaded, the mesh is considered). Provide as argument the expression for the 3 coordinates, combined with '`:`'.
Possible values: `any`. Default value: `x+vx:y+vy:z+vz`.

`-cameralookat` *char_string:char_string:char_string* [Option]
Specify the location the camera looks at. By default, the camera looks at the tessellation or mesh centre. The coordinates of the tessellation or mesh centre are denoted as x, y and z (if both a tessellation and a mesh have been loaded, the mesh is considered). Provide as argument the expression for the 3 coordinates, combined with '`:`'.
Possible values: `any`. Default value: `x:y:z`.

`-cameraangle` *real* [Option]
Specify the opening angle of the camera along the horizontal direction (in degrees). The opening angle along the vertical direction is determined from the opening along the horizontal direction and the image size ratio.
Possible values: `any`. Default value: `25`.

`-camerasky` *real:real:real* [Option]
Specify the sky vector of the camera (vertical direction). Provide as argument the coordinates combined with '`:`'.
Possible values: `any`. Default value: `0:0:1`.

`-cameraprojection` *char_string* [Option]
Specify the type of projection of the camera.
Possible values: `perspective or orthographic`. Default value: `perspective for 3D and orthographic for 2D and 1D`.

4.1.7 Output Image Settings

`-imagesize` *int:int* [Option]
Specify the size of the image (in pixels). Provide as argument the width and height, combined with ':'.
Possible values: `any`. Default value: `1200:900`.

`-imagebackground` *char_string* [Option]
Specify the colour of the background. Provide as argument any colour as detailed in Section A.6 [Colours], page 44.
Possible values: `any`. Default value: `white`.

`-imageantialias` *logical* [Option]
Use antialiasing to produce a smoother image. Switch antialiasing off for faster image generation or smaller image file.
Possible values: `0 or 1`. Default value: `1`.

`-imageformat` *char_string* [Option]
Specify the output image format. It can be the PNG format ('`.png`') or the POV-Ray format ('`.pov`'). Combine with ','.
Possible values: `png or pov`. Default value: `png`.

4.1.8 Scripting

`-loop char_string real real real ... -endloop` [Option]
Use this option to make a loop. Provide as argument the name of the loop variable, its initial value, the loop increment value, the final value then the commands to execute. An example of use of the `-loop` / `-endloop` capability is provided in the Examples Section.
Possible values: `any`. Default value: `none`.

4.1.9 Advanced Options

`-includepov` *char_string* [Option]
Use this option to include additional objects to the image, under the form of a POV-Ray file. Provide as argument the name of the POV-Ray file.
Possible values: `any`. Default value: `none`.

4.2 Output Files

The output files are,

- PNG file, '`.png`': a bitmapped image (the alpha channel is off).
- POV-Ray file, '`.pov`': a POV-Ray script file.

A PNG image can be obtained from a POV-Ray file by invoquing POV-Ray as follows (see the POV-Ray documentation for details and further commands),

```
$ povray Input_File_Name=file.pov +Wimage_width +Himage_height -D +A0.2
```

4.3 Examples

Below are some examples of use of neper -V.

1. Print out tessellation '`n10-id1.tess`' with cells coloured from their identifiers and an image size of 900×450 pixels (for printing out a raster tessellation or mesh, load them in place of the '`.tess`' file).
```
$ neper -V n10-id1.tess -datacellcol id \
        -imagesize 900:450 -print img
```
2. Print out tessellation '`n10-id1.tess`' with cells coloured from crystal orientations and semi-transparency.
```
$ neper -V n10-id1.tess -datacellcol ori -datacelltrs 0.5 -print img
```
3. Print out mesh '`n10-id1.msh`' with elements coloured from scalar values written in file '`v`' and a scale ranging from 0 to 100.
```
$ neper -V n10-id1.msh -dataeltcol scal:v -dataeltscale 0:100 \
        -print img
```
4. Print out mesh '`n10-id1.msh`' with elements coloured from nodal scalar values written in file '`v`' and a scale ranging from 0 to 100.
```
$ neper -V n10-id1.msh -datanodecol scal:v -dataeltcol from_nodes \
        -dataeltscale 0:100 -print img
```
5. Print out the 10 first cells of a 100-cell tessellation, coloured from their identifiers and semi-transparency, and with edges shown in red and vertices shown as green spheres of radius 0.01.
```
$ neper -V n100-id1.tess -showcell "id<=10" \
        -datacellcol id -datacelltrs 0.5 \
        -showedge cell_shown -showver cell_shown \
        -dataverrad 0.01 -dataedgecol red -datavercol green \
        -print img
```
6. Print out mesh '`n100-id1.msh`', but only the interior element sets and shown 1D elements.
```
$ neper -V n100-id1.tess,n100-id1.msh -dataelsetcol id \
        -showelset 'body>0' -showelt1d elt3d_shown -print img
```
7. Print out 3 slices of mesh '`n100-id1.msh`'.
```
$ neper -V n100-id1.msh -dataelsetcol id \
        -slicemesh "x=0.5,y=0.5,z=0.5" -print img
```
8. Print out slices of mesh '`n100-id1.msh`', at z coordinates ranging from 0.1 to 0.9 by step of 0.1, each slice being printed in a separate file.
```
$ neper -V n100-id1.msh -dataelsetcol id \
        -loop Z 0.1 0.1 0.9 \
        -slicemesh "z=Z" -print imgZ \
        -endloop
```

Appendix A Expressions and Keys

A.1 Mathematical and Logical Expressions

Neper can handle mathematical expressions. It makes use of the GNU `libmatheval` library. The expression must contain no space, tabulation or new-line characters, and match the following syntax[1]:

Supported constants are (names that should be used are given in parenthesis): e (`e`), log2(e) (`log2e`), log10(e) (`log10e`), ln(2) (`ln2`), ln(10) (`ln10`), pi (`pi`), pi / 2 (`pi_2`), pi / 4 (`pi_4`), 1 / pi (`1_pi`), 2 / pi (`2_pi`), 2 / sqrt(pi) (`2_sqrtpi`), sqrt(2) (`sqrt`) and sqrt(1 / 2) (`sqrt1_2`).

Variable name is any combination of alphanumericals and `_` characters beginning with a non-digit that is not elementary function name.

Supported elementary functions are (names that should be used are given in parenthesis): exponential (`exp`), logarithmic (`log`), square root (`sqrt`), sine (`sin`), cosine (`cos`), tangent (`tan`), cotangent (`cot`), secant (`sec`), cosecant (`csc`), inverse sine (`asin`), inverse cosine (`acos`), inverse tangent (`atan`), inverse cotangent (`acot`), inverse secant (`asec`), inverse cosecant (`acsc`), hyperbolic sine (`sinh`), cosine (`cosh`), hyperbolic tangent (`tanh`), hyperbolic cotangent (`coth`), hyperbolic secant (`sech`), hyperbolic cosecant (`csch`), hyperbolic inverse sine (`asinh`), hyperbolic inverse cosine (`acosh`), hyperbolic inverse tangent (`atanh`), hyperbolic inverse cotangent (`acoth`), hyperbolic inverse secant (`asech`), hyperbolic inverse cosecant (`acsch`), absolute value (`abs`), Heaviside step function (`step`) with value 1 defined for x = 0, Dirac delta function with infinity (`delta`) and not-a-number (`nandelta`) values defined for x = 0, and error function (`erf`).

Supported unary operation is unary minus (`'-'`).

Supported binary operations are addition (`'+'`), subtraction (`'+'`), multiplication (`'*'`), division multiplication (`'/'`) and exponentiation (`'^'`).

Usual mathematical rules regarding operation precedence apply. Parenthesis (`'('` and `')'`) could be used to change priority order.

Neper includes additional functions: the minimum and maximum functions (`min(a,b)` and `max(a,b)`, respectively). `a` and `b` can be any expression as described above. Moreover, square brackets (`'['` and `']'`) and curly brackets (`'{'` and `'}'`) can be used instead of the parentheses.

The logical operators supported are: $=$ (`==`), $\neq$ (`!=`), $\geq$ (`>=`), $\leq$ (`<=`), $>$ (`>`), $<$ (`<`), AND (`&&`) and OR (`||`).

A.2 Tessellation Keys

Available keys for tessellation germs, vertices, edges, faces, polyhedra and points are provided below. Also note that the descriptors apply to *cells* if they are tagged to apply to *polyhedra* and the tessellation is 3D, *faces* and the tessellation is 2D or *edges* and the tessellation is 1D.

Key	**Descriptor**	**Apply to ...**
`id`	Identifier	germ, ver, edge, face, poly, point
`x`	x coordinate	germ, ver, edge, face, poly, point
`y`	y coordinate	germ, ver, edge, face, poly, point
`z`	z coordinate	germ, ver, edge, face, poly, point
`w`	Laguerre weight	germ
`poly`	polyhedron	point
`true`	true level	ver, edge, face, poly
`body`	body level	ver, edge, face, poly

[1] Taken from the `libmatheval` documentation.

`state`	state	ver, edge, face, poly
`domtype`	type of domain (0 if on a domain vertex, 1 if on a domain edge and 2 if on a domain face)	ver, edge, face
`length`	length	edge
`area`	area	face
`vol`	volume	poly
`size`	size (length/area/volume in 1D/2D/3D)	cell
`diameq`	diameter of the equivalent circle/sphere in 2D/3D, length in 1D	cell, poly, face, edge
`ff`	flatness fault (in degrees)	face
`cyl`	whether or not is used to describe the circular part of a cylinder domain	edge
`vernb`	number of vertices	edge, face, poly
`edgenb`	number of edges	ver, face, poly
`facenb`	number of faces	ver, edge, poly
`polynb`	number of polyhedra	ver, edge, face
`verlist`	vertex list	poly
`edgelist`	edge list	poly
`facelist`	face list	poly
`npolylist`	neighbouring polyhedron list[2]	poly
`facearealist`	face area list	poly
`faceeqlist`	face equation list[3]	poly

The list variables ('`verlist`', etc.) are not available for sorting (option '`-sort`').

For a cell, the `body` and `true` variables are defined as follows,

- *`body`* is an integer equal to `0` if the cell is at the domain boundary, i.e. if it shares at least one face with it (edge in 2D, vertex in 1D), and is equal to `1` or higher otherwise. This is determined as follows: if a cell is surrounded by cells with *`body`* values equal to or higher than *`n`*, its *`body`* value is equal to *`n`* `+ 1`. Therefore, `body` tends to increase with the distance to the domain boundary and can be used to define cells that may suffer from boundary effects.
- *`true`* is an integer equal to `0` it the cell shape is biased by the domain boundary, and is equal to `1` or higher otherwise. A value higher than `0` is achieved if and only if any germ that would have been located outside the domain (where it could not be) would not have affected the shape of the cell. This condition is fulfilled if the distance between the germ of the cell and any of its vertices is lower than the minimum distance between a vertex of the cell and the domain boundary. *`true`* is extended to values higher than `1` in the same way as body: if a cell is surrounded by cells with *`true`* values equal to or higher than *`n`*, its *`true`* value is equal to *`n`* `+ 1`. As *`body`*, *`true`* tends to increase with the distance to the domain boundary, and *true* $\leq$ *body*. *`true`* is especially useful for statistics on the cells (morphology, mesh, etc.), for which only cells with *true* ≥ 1 should be considered.

For entities of lower dimension than cells (vertices, edges and faces), *`body`* and *`true`* are equal to the maximum *`body`* or *`true`* values of the cells they belong to.

[2] If a polyhedron has no neighbour on a face, a negative value is returned instead of the neighbour id.

[3] A face equation is specified by the parameters d, a, b and c, with the equation being: $ax + by + cz = d$. The vector (a, b, c) is pointing outwards of the polyhedron.

A.3 Raster Tessellation Keys

Available keys for raster tessellation germs and cells are provided below.

Key	Descriptor	Apply to ...
`id`	Identifier	germ, cell
`x`	x coordinate	germ, cell
`y`	y coordinate	germ, cell
`z`	z coordinate	germ, cell
`w`	Laguerre weight	germ
`size`	size (length/area/volume in 1D/2D/3D)	cell

A.4 Mesh Keys

Available keys for mesh node, elements and element sets (of all dimensions) and points are provided below.

Key	Descriptor	Apply to ...
`id`	Identifier	node, *n*D elt, *n*D elset, point
`x`	x coordinate	node, *n*D elt, *n*D elset, point
`y`	y coordinate	node, *n*D elt, *n*D elset, point
`z`	z coordinate	node, *n*D elt, *n*D elset, point
`dim`	dimension (= lowest parent elt dimension)	node
`elt3d`	3D element	point
`elset0d`	0D elset	0D elt
`elset1d`	1D elset	1D elt
`elset2d`	2D elset	2D elt
`elset3d`	3D elset	3D elt, point
`part`	partition	*n*D elt, node
`cyl`	whether or not is used to describe the circular part of a cylinder domain	1D elt, 1D elset
`vol`	volume	3D elt, 3D elset
`area`	area	2D elt
`length`	length	1D elt, 3D elt, 1D elset
`rr`	radius ratio	3D elt
`rrav`, `rrmin`, `rrmax`	average, min and max radius ratios	3D elset
`Osize`	Osize	3D elset
`eltnb`	number of elements	*n*D elset
`true`	true level	*n*D elt, *n*D elset
`body`	body level	*n*D elt, *n*D elset
`2dmeshp`	coordinates of the closest point of the 2D mesh	node, 3D elt, point
`2dmeshd`	distance to '`2dmeshp`'	node, 3D elt, point
`2dmeshv`	vector to '`2dmeshp`'	node, 3D elt, point
`2dmeshn`	outgoing normal vector of the 2D mesh at '`2dmeshp`'	node, 3D elt, point

nD stands for an arbitrary dimension (from 0D to 3D). Variables starting by '`2dmesh`' are only available for statistics (options starting by '`-stat`' of module -M); for elements, they apply to the centroids.

A.5 Rotations and Orientations

Rotations and orientations can be described using the following descriptors (see Orilib, http://orilib.sourceforge.net, for more information).

Key	Descriptor	Number of variables
g	Rotation matrix	9
rtheta	Rotation axis / angle pair	4
R	Rodrigues vector	3
q	Quaternion	4
e	Euler angles (Bunge convention)	3
ek	Euler angles (Kocks convention)	3
er	Euler angles (Roe convention)	3

A.6 Colours

The available colours are provided below, with their corresponding RGB channel values. Any other colour can be defined from the RGB channel values, under format '*R_value*|*G_value*|*B_value*'.

(0, 0, 0)	black	(255, 0, 0)	red
(0, 255, 0)	green	(0, 0, 255)	blue
(255, 255, 0)	yellow	(255, 0, 255)	magenta
(0, 255, 255)	cyan	(255, 255, 255)	white
(128, 0, 0)	maroon	(0, 0, 128)	navy
(127, 255, 0)	chartreuse	(0, 255, 127)	springgreen
(128, 128, 0)	olive	(128, 0, 128)	purple
(0, 128, 128)	teal	(128, 128, 128)	grey
(0, 191, 255)	deepskyblue	(124, 252, 0)	lawngreen
(64, 64, 64)	darkgrey	(255, 69, 0)	orangered
(192, 192, 192)	silver	(255, 250, 250)	snow
(139, 0, 0)	darkred	(0, 0, 139)	darkblue
(255, 140, 0)	darkorange	(240, 255, 255)	azure
(248, 248, 255)	ghostwhite	(255, 255, 240)	ivory
(0, 0, 205)	mediumblue	(255, 182, 193)	lightpink
(245, 255, 250)	mintcream	(75, 0, 130)	indigo
(240, 128, 128)	lightcoral	(255, 192, 203)	pink
(255, 127, 80)	coral	(250, 128, 114)	salmon
(255, 250, 240)	floralwhite	(127, 255, 212)	aquamarine
(255, 250, 205)	lemonchiffon	(255, 215, 0)	gold
(0, 100, 0)	darkgreen	(255, 165, 0)	orange
(240, 248, 255)	aliceblue	(224, 255, 255)	lightcyan
(255, 255, 224)	lightyellow	(139, 0, 139)	darkmagenta
(0, 139, 139)	darkcyan	(205, 133, 63)	peru
(70, 130, 180)	steelblue	(255, 240, 245)	lavenderblush
(255, 245, 238)	seashell	(0, 250, 154)	mediumspringgreen
(72, 61, 139)	darkslateblue	(184, 134, 11)	darkgoldenrod
(255, 160, 122)	lightsalmon	(255, 228, 196)	bisque
(135, 206, 250)	lightskyblue	(250, 250, 210)	lightgoldenrodyellow
(240, 255, 240)	honeydew	(255, 248, 220)	cornsilk
(255, 218, 185)	peachpuff	(245, 245, 245)	whitesmoke
(255, 99, 71)	tomato	(112, 128, 144)	slategrey
(255, 105, 180)	hotpink	(253, 245, 230)	oldlace

(255, 235, 205)	blanchedalmond	(189, 183, 107)	darkkhaki
(255, 228, 181)	moccasin	(0, 206, 209)	darkturquoise
(60, 179, 113)	mediumseagreen	(199, 21, 133)	mediumvioletred
(238, 130, 238)	violet	(173, 255, 47)	greenyellow
(255, 239, 213)	papayawhip	(143, 188, 143)	darkseagreen
(188, 143, 143)	rosybrown	(255, 20, 147)	deeppink
(139, 69, 19)	saddlebrown	(148, 0, 211)	darkviolet
(30, 144, 255)	dodgerblue	(119, 136, 153)	lightslategrey
(222, 184, 135)	burlywood	(255, 222, 173)	navajowhite
(250, 240, 230)	linen	(123, 104, 238)	mediumslateblue
(64, 224, 208)	turquoise	(135, 206, 235)	skyblue
(72, 209, 204)	mediumturquoise	(245, 245, 220)	beige
(255, 228, 225)	mistyrose	(210, 180, 140)	tan
(250, 235, 215)	antiquewhite	(216, 191, 216)	thistle
(50, 205, 50)	limegreen	(233, 150, 122)	darksalmon
(176, 196, 222)	lightsteelblue	(65, 105, 225)	royalblue
(152, 251, 152)	palegreen	(220, 20, 60)	crimson
(245, 222, 179)	wheat	(186, 85, 211)	mediumorchid
(230, 230, 250)	lavender	(240, 230, 140)	khaki
(144, 238, 144)	lightgreen	(175, 238, 238)	paleturquoise
(47, 79, 79)	darkslategrey	(153, 50, 204)	darkorchid
(46, 139, 87)	seagreen	(154, 205, 50)	yellowgreen
(138, 43, 226)	blueviolet	(219, 112, 147)	palevioletred
(107, 142, 35)	olivedrab	(147, 112, 219)	mediumpurple
(244, 164, 96)	sandybrown	(85, 107, 47)	darkolivegreen
(102, 205, 170)	mediumaquamarine	(106, 90, 205)	slateblue
(238, 232, 170)	palegoldenrod	(34, 139, 34)	forestgreen
(25, 25, 112)	midnightblue	(32, 178, 170)	lightseagreen
(211, 211, 211)	lightgrey	(218, 112, 214)	orchid
(100, 149, 237)	cornflowerblue	(160, 82, 45)	sienna
(178, 34, 34)	firebrick	(176, 224, 230)	powderblue
(205, 92, 92)	indianred	(105, 105, 105)	dimgrey
(173, 216, 230)	lightblue	(210, 105, 30)	chocolate
(165, 42, 42)	brown	(218, 165, 32)	goldenrod
(220, 220, 220)	gainsboro	(221, 160, 221)	plum
(95, 158, 160)	cadetblue		

The default colour palette (used for options '`-datacellcol`', '`-dataelt3dcol`', etc.) is defined from the above colour list, by excluding colours of brightness below 0.2 and above 0.8. The brightness is defined as the average of the channel values divided by 255. The resulting list of colours is: `red`, `green`, `blue`, `yellow`, `magenta`, `cyan`, `chartreuse`, `springgreen`, `olive`, `purple`, `teal`, `grey`, `deepskyblue`, `lawngreen`, `darkgrey`, `orangered`, `silver`, `darkorange`, `mediumblue`, `indigo`, `lightcoral`, `coral`, `salmon`, `aquamarine`, `gold`, `orange`, `darkmagenta`, `darkcyan`, `peru`, `steelblue`, `mediumspringgreen`, `darkslateblue`, `darkgoldenrod`, `lightsalmon`, `lightskyblue`, `tomato`, `slategrey`, `hotpink`, `darkkhaki`, `darkturquoise`, `mediumseagreen`, `mediumvioletred`, `violet`, `greenyellow`, `darkseagreen`, `rosybrown`, `deeppink`, `saddlebrown`, `darkviolet`, `dodgerblue`, `lightslategrey`, `burlywood`, `mediumslateblue`, `turquoise`, `skyblue`, `mediumturquoise`, `tan`, `limegreen`, `darksalmon`, `lightsteelblue`, `royalblue`, `palegreen`, `crimson`, `mediumorchid`, `khaki`, `lightgreen`, `darkslategrey`, `darkorchid`, `seagreen`, `yellowgreen`, `blueviolet`, `palevioletred`, `olivedrab`, `mediumpurple`, `sandybrown`, `darkolivegreen`, `mediumaquamarine`, `slateblue`,

forestgreen, midnightblue, lightseagreen, orchid, cornflowerblue, sienna, firebrick, indianred, dimgrey, chocolate, brown, goldenrod, plum and cadetblue.

Appendix B File Formats

B.1 Tessellation File ('.tess')

Here are details on the '.tess' file format version 2.0. Developers should note that read and write functions are available as 'neut_tess_fscanf' and 'neut_tess_fprintf', defined in directories 'neut/neut_tess/neut_tess_fscanf' and 'neut/neut_tess/neut_tess_fprintf'.

```
***tess
 **format
   format
 **general
   dim type
 **cell
   number_of_cells
 [*id
   cell1_id cell2_id ... ]
 [*germ
   germ_id germ_x germ_y germ_z germ_weight
   ... ]
 [*ori
   descriptor
   cellid_param1 cellid_param2 ...
   ... ]
 **vertex
   total_number_of_vertices
   ver_id  ver_x ver_y ver_z ver_state
   ...
 **edge
   total_number_of_edges
   edge_id  ver_1 ver_2 edge_state
   ...
 **face
   total_number_of_faces
   face_id number_of_vertices ver_1   ver_2   ...
           number_of_edges    edge_1* edge_2* ...
           face_eq_d face_eq_a face_eq_b face_eq_c
           face_state face_point face_point_x face_point_y face_point_z
   ...
 **polyhedron
   total_number_of_polyhedra
   poly_id number_of_faces face_1* face_2* ...
   ...
 **domain
  *general
   dom_type
  *vertex
   total_number_of_dom_vertices
   dom_ver_id  dom_ver_x dom_ver_y dom_ver_z dom_ver_label
               number_of_dom_tess_vertices ver_1
   ...
```

```
  *edge
   total_number_of_dom_edges
   dom_edge_id dom_ver_1 dom_ver_2 dom_edge_label
               number_of_dom_tess_edges edge_1 edge_2 ...
   ...
  *face
   total_number_of_dom_faces
   dom_face_id number_of_dom_vertices dom_ver_1  dom_ver_2  ...
               number_of_dom_edges    dom_edge_1 dom_edge_2 ...
               dom_face_eq_d dom_face_eq_a dom_face_eq_b dom_face_eq_c
               dom_face_label
               number_of_dom_tess_faces dom_tess_face_1 dom_tess_face_2 ...
   ...
***end
```

where (with identifiers being integer numbers),

- `***tess` denotes the beginning of a tessellation file.
- `**format` denotes the beginning of the format field.
- *`format`* is the file format, currently '`2.0`' (character string).
- `**general` denotes the beginning of the general information field.
- *`dim`* is the dimension of the tessellation (1, 2 or 3).
- *`type`* is the type of tessellation (always '`standard`').
- `**cell` denotes the beginning of the cell field.
- *`number_of_cells`* is the number of cells.
- `*id` denotes the beginning of an optional identifier field. It the field is not present, the cells are considered to be numbered contiguously from 1.
- *`cell1_id`*, *`cell2_id`*, ... are the actual identifiers of the cells.
- `*germ` denotes the beginning of a germ field.
- *`germ_id`* is the identifier of a germ and ranges from `1` to `number_of_cells`.
- *`germ_x`*, *`germ_y`* and `germ_z` are the three coordinates of a germ (real numbers).
- *`germ_weight`* is the weight of a germ (real number).
- `*ori` denotes the beginning of an optional crystal orientation field.
- `descriptor` is the descriptor used to parametrize the crystal orientations. See Section A.5 [Rotations and Orientations], page 44 for the list of available descriptors.
- *`cellid_param1`*, *`cellid_param2`*, ... are the values of the orientation descriptor of cell *id*.
- `**vertex` denotes the beginning of the vertex field.
- *`total_number_of_vertices`* is the total number of vertices.
- *`ver_id`* is the identifier of a vertex and ranges from `1` to `total_number_of_vertices`.
- *`ver_x`*, *`ver_y`* and `ver_z` are the three coordinates of a vertex (real numbers).
- *`ver_state`* is an integer indicating the state of a vertex. For a standard tessellation (no regularization), it equals `0`. For a regularized tessellation, it equals `0` if the vertex has not been modified by regularization and is higher than `0` otherwise.
- `**edge` denotes the beginning of the edge field.
- *`total_number_of_edges`* is the total number of edges.
- *`edge_id`* is the identifier of an edge and ranges from `1` to `total_number_of_edges`.
- *`ver_1`*, *`ver_2`*, ... are identifiers of vertices.

- *edge_state* is an integer indicating the state of an edge (always 0).
- **face denotes the beginning of the face field. It is present for a tessellation of dimension 2 or 3.
- *total_number_of_faces* is the total number of faces.
- *face_id* is the identifier of a face and ranges from 1 to total_number_of_faces.
- *number_of_vertices* is the number of vertices of a face.
- *number_of_edges* is the number of edges of a face.
- *edge_1**, *edge_2**, ... are identifiers of the edges of a face, signed according to their orientation in the face.
- *face_eq_a*, *face_eq_b*, *face_eq_c* and *face_eq_d* are the parameters of the equation of a face: $face\_eq\_a\,x + face\_eq\_b\,y + face\_eq\_c\,z = face\_eq\_d$. The parameters are scaled so that $face\_eq\_a^2 + face\_eq\_b^2 + face\_eq\_c^2 = 1$.
- *face_state* is an integer indicating the state of a face. For a standard tessellation (no regularization), it equals 0. For a regularized tessellation, it equals 0 if it has not been modified by regularization and 1 otherwise.
- *face_point* is an integer indicating the point used for the interpolation of a face. For a standard tessellation (no regularization), it equals 0. For a regularized tessellation: if the point is the face barycentre, it equals 0; if the point is one of the face vertices, it equals to the position of the vertex in the list of vertices of the face. It equals -1 if the point is undefined.
- *face_point_x*, *face_point_y* and *face_point_z* are the coordinates of the point used for the interpolation of a face (equal 0 if undefined).
- **polyhedron denotes the beginning of the polyhedron field. It is present for a tessellation of dimension 3.
- *total_number_of_polyhedra* is the total number of polyhedra.
- *poly_id* is the identifier of a polyhedron and ranges from 1 to total_number_of_polyhedra.
- *number_of_faces* is the number of faces of a polyhedron.
- *face_1**, *face_2**, ... are identifiers of the faces of a polyhedron, signed according to their orientations in the polyhedron (positive if the normal of the face is pointing outwards and negative if it is pointing inwards).
- **domain denotes the beginning of the domain field.
- *general denotes the beginning of the domain general information field.
- *dom_type* is the type of the domain (one of cube, cylinder, square, circle, poly and planes).
- *vertex denotes the beginning of the domain vertex field.
- *total_number_of_dom_vertices* is the total number of domain vertices.
- *dom_ver_id* is the identifier of a domain vertex and ranges between 1 to total_number_of_dom_vertices.
- *dom_ver_x*, *dom_ver_y* and dom_ver_z are the three coordinates of a domain vertex (real numbers).
- *dom_ver_label* is the label of a domain vertex.
- *number_of_dom_tess_vertices* is the number of tessellation vertices of a domain vertex (must be 1).
- *edge denotes the beginning of the domain edge field (for a tessellation of dimension 2 or 3).

- *total_number_of_dom_edges* is the total number of domain edges.
- *dom_edge_id* is the identifier of a domain edge and ranges between 1 to total_number_of_dom_edges.
- *dom_ver_1*, *dom_ver_2*, ... are identifiers of the domain vertices of a domain edge or face.
- *dom_edge_label* is the label of a domain edge.
- *number_of_dom_tess_edges* is the number of tessellation edges of a domain edge.
- *face denotes the beginning of the domain face field (for a tessellation of dimension 3).
- *total_number_of_dom_faces* is the total number of domain faces.
- *dom_face_id* is the identifier of a domain face and ranges from 1 to total_number_of_dom_faces.
- *number_of_dom_vertices* is the number of domain vertices of a domain face.
- *number_of_dom_edges* is the number of domain edges of a domain face.
- *dom_edge_1*, *dom_edge_2*, ... are identifiers of the domain edges of a domain face.
- *dom_face_eq_a*, *dom_face_eq_b*, *dom_face_eq_c* and *dom_face_eq_d* are the parameters of the equation of a domain face and are defined in the same way than *face_eq_a*, etc. (see above).
- *dom_face_label* is the label of a domain face. If dom_type is 'cube', it is one of 'x0', 'x1', 'y0', 'y1', 'z0' or 'z1'. If dom_type is 'cylinder', it is one of 'z0', 'z1', 'f1', 'f2', ... Otherwise, it is one of 'f1', 'f2', ...
- *number_of_dom_tess_faces* is the number of tessellation faces of a domain face.
- *dom_tess_face_1*, *dom_tess_face_2*, ... are the identifiers of the tessellation faces of a domain face.
- ***end denotes the end of a tessellation file.

B.2 Raster Tessellation File ('.tesr')

Here are details on the '.tesr' file format version 2.0. The developers should note that read and write functions are available as 'neut_tesr_fscanf' and 'neut_tesr_fprintf', defined in directories 'neut/neut_tesr/neut_tesr_fscanf' and 'neut/neut_tesr/neut_tesr_fprintf'.

```
***tesr
 **format
   format data_format
 **general
   dimension
   size1 [size2 size3]
   rpt_size1 [rpt_size2 rpt_size3]
[**cell
   number_of_cells
 [*id
   cell1_id cell2_id ...]]
 [*germ
   germ_id germ_x germ_y germ_z germ_weight
   ... ]
 [*ori
   descriptor
   cellid_param1 cellid_param2 ...]]
 **data
   rpt1_cell rpt2_cell ...
```

```
 or
  *file data_file_name
***end
```

where,

- `***tesr` denotes the beginning of a raster tessellation file.
- `**format` denotes the beginning of the format field.
- *`format`* is the file format, currently '`2.0`' (character string).
- *`data_format`* is the format of the data in field `**data`. It can be either `ascii`, `binary8` (8-bit binary), `binary16` (16-bit binary) or `binary32` (32-bit binary). The '`*`' suffix can be added to `binary16` and `binary32` to switch byte ordering (from LittleEndian to BigEndian, and vice versa).
- `**general` denotes the beginning of the general information field.
- *`dimension`* is the dimension of the raster tessellation.
- *`size1`*, *`size2`* and *`size3`* are the raster sizes along the 3 coordinate axes. The number of sizes must match *dimension*.
- *`rpt_size1`*, *`rpt_size2`* and *`rpt_size3`* are the point sizes along the 3 coordinate axes. The number of sizes must match *dimension*.
- `**cell` denotes the beginning of an optional cell field.
- *`number_of_cells`* is the number of cells.
- `*id` denotes the beginning of an optional identifier field. If the field is present, the cell identifiers listed under `**data` are supposed to be numbered contiguously from 1 (or 0 in case of void), and their actual identifiers are considered to be the ones provided in the list. The actual identifiers are used in output files.
- *`cell1_id`*, *`cell2_id`*, ... are the actual identifiers of the cells.
- `*germ` denotes the beginning of a germ field.
- *`germ_id`* is the identifier of a germ and ranges from `1` to `number_of_cells`.
- *`germ_x`*, *`germ_y`* and `germ_z` are the three coordinates of a germ (real numbers).
- *`germ_weight`* is the weight of a germ (real number).
- `*ori` denotes the beginning of an optional crystal orientation field. It requires field `*id`.
- `descriptor` is the descriptor used to parametrize the crystal orientations. See Section A.5 [Rotations and Orientations], page 44 for the list of available descriptors.
- *`cellid_param1`*, *`cellid_param2`*, ... are the values of the orientation descriptor of cell *id*.
- `**data` denotes the beginning of the data field. Data can be provided in the '`.tesr`' file or in a separate file, using '`*file`', see below.
- *`rptid_cell`* is the cell raster point *`id`* belongs to. The cell identifiers should start from 1. Use 0 for voids.
- `*file` denotes the beginning of a file field.
- `data_file_name` is the name of a file that contains the data. Typically, it is a '`.raw`' file.

B.3 Position File

A position file lists the coordinates of a given number of points. The file must contain 1 coordinate per point in 1D, 2 coordinates per point in 2D and 3 coordinates per point in 3D. A coordinate can be an integer or real number. A real number can have an arbitrary number of digits, but the decimal mark must be '`.`'. The coordinates can be separated from each other by spaces, tabulators or newlines (any number as well as arbitrary combinations of them are

supported). However, a good practice is to format the file with one line per point. An example of a position file containing 5 points in 3D is as follows,

```
2.1235 9.4544 5.2145
5.9564 3.6884 9.2145
2.2547 3.2658 8.2514
8.2515 9.4157 2.9454
0.5874 4.2848 2.4874
```

Appendix C Developer's Guide

This chapter provides information useful to anyone who plans to contribute to Neper or wishes to better understand how it works. The code structure is detailed and information are given on how to efficiently contribute to it. If you are missing information, complain!

C.1 Code Structure

The Neper root directory content is as follows (the slash character '`/`' denotes directories),

- `COPYING`: license terms
- `README`: information about other files and directories in the directory
- `VERSIONS`: information on the versions of Neper
- `src/`: source code directory
- `doc/`: documentation directory

Details on the '`src/`' and '`doc/`' directories are provided in the following.

C.1.1 Source Code

Neper's source code is located in directory '`src/`' and consists of roughly 85,000 lines shared between 200 directories and 800 text files. The '`src/`' directory contains the following files and directories,

- `neper_.h` and `neper.c`

  These are the main source code header file and source code file of Neper. '`neper.c`' contains the program '`main`' function. It reads the arguments passed at the command line and runs the corresponding functions, which can be one of the program module.

- `neper_t/`, `neper_m/`, `neper_v/` and `neper_d/`

  These are directories that contain the source code of each of the program modules. The modules aim to be independent from each other as much as possible, that is, a function of a given module will never calls a function of another module (with a few exceptions).

- `neut/`

  '`neut`' stands for *Neper utilities*. The directory contains utility functions specific to Neper and used by several modules.

- `contrib/`

  This directory contains utility functions not specific to Neper. The first one is '`ut`', which is a collection of general-purpose, low-level C functions (memory allocation, etc.). The second one is '`orilib`', which is a collection of routines for orientation manipulation (see `http://orilib.sourceforge.net`). The last one is ANN, a library for nearest neighbour searching (see `www.cs.umd.edu/~mount/ANN`). Although these libraries also are distributed alone (and might be already installed on your system), they are included into Neper instead of being considered as dependencies (contrary to the GSL, libmatheval, ...), to make Neper's installation easier.

- `CMakeLists.txt`, `neper_config.h.in` and `cmake/`

  These files and directories are specific to the building system, CMake. '`CMakeLists.txt`' is the CMake source file, which tells CMake where to find the program source files, how to manage dependencies, where to install Neper, etc. '`neper_config.h.in`' is a small configuration file that is useful to CMake for managing dependencies and program version numbers. '`cmake/`' contains '`.cmake`' files which help CMake locating the dependencies on the system (library and header files).

A module directory, `neper_X/`, where '`X`' stands for the module letter (one of '`t`', '`m`', '`v`' or '`d`'), is structured as follows,

- `neper_X.h`, `neper_X_.h` and `neper_X.c`

  These are the source code header files and source code file of the module. '`neper_X.c`' contains the module function, '`neper_X`'. '`neper_X_.h`' is the source code header file, which is `#include`'ed in '`neper_X.c`' and contains a bunch of `#include`s to all necessary library header files. '`neper_X.h`' contains the prototype of the module function and is `#include`'ed in '`neper_.h`'. Hence, files '`_.h`' are local header files while files '`.h`' are header files `#include`'ed into a upper-level source code header file. This is true anywhere in the source code. Moreover, any function specific to module `X` is prefixed '`neX_`'.

- `neX_input/` and `structIn_X.h`

  The '`neX_input/`' directory contains functions for reading the value of the arguments passed to module `X` from the command line. The information are recorded into an '`IN`' C structure, which is declared in file '`structIn_X.h`'.

- `neX_foo/`, `neX_bar/`, etc.

  Each of these directories is associated to a specific task of the module and contains a function of the same name ('`neX_foo`', etc.) which is called from function '`neper_X`'. Each directory contains a directory tree structure.

- `CMakeLists.txt`

  This file tells CMake where to find the source files and how to manage dependencies in the module. It is used by the upper-level '`CMakeLists.txt`' file (there is no lower-level '`CMakeLists.txt`' file).

The '`neut`' directory is roughly structured as follows,

- `CMakeLists.txt`

  This file tells CMake where to find the source files and how to manage dependencies in the module. It is used by the upper-level '`CMakeLists.txt`' file (there is no lower-level '`CMakeLists.txt`' file).

- `neut.h`, `neut_t.h`, `neut_m.h` and `neut_v.h`

  These files are source code header files that `#include` header files of `neut` (which contain function prototypes) and are `#include`d in the modules. '`neut.h`' `#include`s all header files while the three others `#include` header files only necessary to the corresponding module (this speeds up compilation at development stage).

- `neut_structs/`

  This directory contains header files which defines all C structures used in the program.

- `neut_foo/`, `neut_bar/`, etc.

  Each of these directories contain functions specific to a particular C structure. For example, '`neut_tess`' contains functions relative to the '`TESS`' structure, which describes a tessellation.

C.1.2 Documentation

Neper's documentation is located in directory '`doc/`'. It is written in Texinfo, the GNU software documentation system. The documentation consists in a collection of '`.texi`' files (text files). The documentation may be compiled in PDF, info or html format by running `make pdf`, `make info` or `make html`, respectively. In official releases, both the PDF and info documentation files are built and included in the archive.

C.2 Contributing to Neper

C.2.1 Coding Conventions

Neper is written following the GNU Coding Standards (http://www.gnu.org/prep/standards), with the exception that braces are not indented (because there is so often 3+ loop levels in Neper). Please follow this convention. Here are a few tips and other remarks,

- For Vim, put the following commands in file '`$HOME/.vimrc`':

```
:set sw=2
:set cindent
:syntax enable
:set textwidth=72
```

- Your can run `indent -bli0` *`source_files`* for automatic formatting.
- Break up the code into meaningful chunks using blank lines. Always use a single blank line to separate parts of the code.
- Neper admits no compilation warnings. Please fix all of them up.
- Please help us maintaining good documentation by documenting any capability you may add.

C.2.2 Adding a New Option

In modules -T, -M and -D, adding a new option can be done by following the successive steps,

- Add a variable to the '`IN`' structure to record the value of the option (file '`structIn_X.h`').
- Allocate / free the variable in the `neX_in_set_zero` and `neX_in_free` functions (file '`neX_input1.c`') Assign it a default value in `net_input_options_default` (file '`neX_input3.c`').
- Add the option to the option list in `net_input_options_set` (file '`neX_input3.c`'), taking as an example another option of the same type (integer, etc.).
- Where appropriate in the source code, add a new function for the new option (if necessary in a new file or directory). The function should be executed depending on the value of the option.
- If adding one or several files or directories, add the source file(s) to the source file list in the '`CMakeLists.txt`' file of the corresponding module.
- Make sure the whole thing compiles and runs properly for your purpose.
- Make sure your own changes did not break anything in the rest of the code by running full testing, using module -D as detailed in the following. You may also want to add a test specific to the new option.
- You may submit your code by email to rquey@users.sourceforge.net for inclusion into Neper's official distribution.

In module -V, options are processed differently. Instead of being recorded in a C structure, they are read one after the other and associated functions are executed along the way. To add a new option, take an existing option as an example.

C.2.3 Compilation Options

For development, several compilation options can be changed from their default values. This must be done at configuration stage, using commands '`ccmake ..`' or '`cmake-gui ..`'. The compilation options are,

- `HAVE_DEBUGGING`

  Setting the option to `ON` turns on the debugging compilation flag '`-g`', which is required for debugging with `gdb` and `valgrind`, turns on the compilation flag '`-Werror`', which makes

all compilation warnings into errors, and also runs internal tests during Neper execution at place where the code is otherwise considered as robust.

- `HAVE_OPTIMIZATION`

  Setting this option to `OFF` disables code optimization, which is useful for debugging with `gdb` and `valgrind`.

- `HAVE_PROFILING`

  Setting this option to `ON` turns on the code profiling compilation flag '`-pg`', which is required for profiling with `gprof`. This is a highy CPU-sensitive option, which should be used only when profiling is actually carried out.

C.3 Testing Module (-D)

Module -D is the module for testing Neper (which is helpful for development, hence the name). Module -D makes Neper run predefined commands on itself, for module -T (Chapter 2 [Tessellation Module (-T)], page 7), module -M (Chapter 3 [Meshing Module (-M)], page 17) and module -V (Chapter 4 [Visualization Module (-V)], page 29). The output of each command is checked and a report of how many tests have failed (if any) is provided. For any properly-installed, official release of Neper, all tests should succeed. The module runs in a temporary directory created into the working directory and named '`neper_testing`'. The directory is erased once testing completes.

A typical use is to run '`neper -D all`', which tests all modules. The tests can only be run on specific modules or on specific commands of a module (option '`-test`'), which is handy for debugging.

Here is what a typical run of module -D looks like,

```
$ neper -D all -run fast

=======================    N   e   p   e   r    =======================
Info   : A software package for polycrystal generation and meshing.
Info   : Version 2.0.0
Info   : Built with: gsl libmatheval libscotch
Info   : Loading initialization file '/foo/bar/.neperrc'...
Info   : ---------------------------------------------------------------
Info   : MODULE  -D loaded with arguments:
Info   : [ini file] (none)
Info   : [com line] all -run fast
Info   : ---------------------------------------------------------------
Info   : Reading input data...
Info   : Running in 'fast' mode...
Info   : Testing module -T... (80 tests)
Info   :   - Preparing...
Info   :   [ 1] -n 2 ........................................... passed
Info   :   [ 2] -n 2 -id 1 ..................................... passed
[...]
Info   :   [80] -n 2 -id 1 -statpoint id,x,y,z,poly ............ passed
Info   :   - 80 tests, passed: 80, skipped: 0, failed: 0.
Info   : Testing module -M... (64 tests)
Info   :   - Preparing...
Info   :   [ 1] n2-id1.tess ..................................... passed
Info   :   [ 2] n2-id1.tess -elt hex ............................ passed
```

```
[...]
Info   :   [64] n2-id1-dim2.tesr -rcl 0.7 -for msh,geof,inp .... passed
Info   :   - 64 tests, passed: 64, skipped: 0, failed: 0.
Info   : Testing module -V... (137 tests)
Info   :   - Preparing...
Info   :   [ 1] n2-id1.tess -print foo ......................... passed
Info   :   [ 2] n2-id1.tess -showcell none -showcell all   [...] passed
[...]
Info   :   [137] n2-id1.msh -loop F 1 0.1 1.2 -datanodecoo [...] passed
Info   :   - 137 tests, passed: 137, skipped: 0, failed: 0.
Info   : Elapsed time: 209.715 secs.
=====================================================================
```

C.3.1 Arguments

C.3.1.1 Prerequisites

`-binary` *path_name* [Prerequisite]
Specify the path of the current Neper binary (on which the tests are run). On most Unix systems, Neper will be able to find it by itself; use this option otherwise.
Possible values: `any`. Default value: `current_binary_path`.

C.3.1.2 Input Data

char_string [Input data]
Modules on which the tests are run. Provide '`all`' for all, '`none`' for none or a list of modules combined with '`,`', e.g. '`-T,-M`'.
Possible values: `any`. Default value: `none`.

C.3.1.3 Testing Options

`-test` *char_string* [Option]
Specify the list of tests to run. Provide '`all`' for all, '`none`' for none, '`last`' for the last one of the tests, or a list of test numbers combined with '`,`'.
Possible values: `any`. Default value: `all`.

`-runmode` *char_string* [Input data]
Specify the running mode. This changes the number of cells. Provide '`fast`', '`normal`', '`extensive`', '`paranoiac`' to use 2, 32, 512 and 8192-cell tessellations, respectively.
Possible values: `'fast', 'normal', 'extensive' or 'paranoiac'`. Default value: `'normal'`.

C.3.2 Examples

Below are some examples of use of neper -D.

1. Test all modules.
   ```
   $ neper -D all
   ```
2. Test module -T only.
   ```
   $ neper -D -T
   ```
3. Run specific tests of module -T.
   ```
   $ neper -D -T -test 10,12,45,57
   ```

Appendix D Versions

```
New in 2.0.0 (10 Jan 2014):
- General: Full restructuring and added many new features.  Neper now
  has 3 main modules: tessellation module (-T), meshing module (-M) and
  visualization module (-V); details are provided below. Added
  developer's guide and module (-D). Documentation has been much
  improved.
- module -T: added several tessellation algorithms (hardcore Voronoi
  and Laguerre Voronoi); added orientation generation (was previously in
  -O); significantly sped up tessellation; included and significantly
  sped up regularization (was previously in -FM); added 2-scale
  polycrystal generation; added 2D and 1D supports; improved statistics;
  enabled both scalar (tess) and raster (tesr) outputs; cleaned up tess
  file.
- module -M: module for free and mapped meshings (merging of -FM and
  -MM). Removed regularization (now in -T); added per-cell mesh size
  definition; sped up multimeshing; improved statistics.
- module -V: full restructuring; added support for 2D and 1D
  tessellations and meshes; the way all entities are shown (cells,
  polyhedra, faces, edges, vertices, germs, 3D/2D/1D/0D element sets and
  elements, nodes) can be set in great detail; added transparency.

New in 1.10.3 (26 Nov 2012):
- module -T: added 3dec geometry format, added option -checktess, minor
  improvements, added individual file extension support in -stattess,
  changed option -neigh 1 to -statp i,f,npl,fal,feql.
- module -FM: added 3dec geometry format; changed "top" and "bot" nset
  names for cylindrical domains to "z0" and "z1"; minor bug fixes;
  improved fev format support; added individual file extension support in
  options -stattess and -statmesh.
- module -O:  minor bug fixes.
- module -MM: sped up meshing; fixed -domain, -scale and -nset options,
  add .nper file for periodicity conditions; fixed msh output for meshes
  with different element dimensions; minor other bug fixes.
- module -VS: sped up meshing reconstruction and PNG file generation,
  added option '-camerasky', added option '-showeltedge', sped up mesh
  reconstruction, minor fixes
- documentation: minor fixes.
- General: minor fixes.

New in 1.10.2 (09 Aug 2012):
- module -T: fixed -centroid option.
- module -FM: fixed list of available meshing algorithms.  Added
  tests.
- module -MM: fixed nset syntax in inp (Abaqus) files.
- module -VS: added capability to plot mapped meshes.
- General: various minor improvements, code cleaning.

New in 1.10.1 (08 June 2012):
- Bug fix to get Neper working after invoquing 'make install'.
```

```
New in 1.10.0 (04 June 2012):
- General: New (hopefully simpler) installation procedure based on Cmake.
  Added support for domains of any convex polyhedral shape.
- module -VS: major code rewriting and option changes. New capabilities
  for defining the colours and sizes of the tessellation / mesh
  (including gradients).  Added options to show only specific parts of the
  tessellation / mesh and to view slices of a mesh. Other small
  enhancements.
- module -T : added option '-domain' to specify the shape of the domain
  (cuboidal, cylindrical or of any convex shape), small bug fixes, added
  centroid Voronoi tessellation generation (option -centroid), merged
  option -centrecoo into option -morpho, added polyhedron centroid
  coordinates in file .stt3, changed option -load to -loadtess, added
  output format '.ply' (thanks Ehsan!).
- module -FM: mesh partitionning needs libscotch version 5.1.12 or later,
  small bug fixes, changed default value of -faset to "" (i.e. no faset
  in output), fixed bug for Abaqus output, added polyhedron centroid
  coordinates in file .stt3, added output format '.ply' (geometry only).
- module -MM: new options -dsize and -scale, new option -loadmesh, new
  option -outdim, changed arguments of -ttype, changed default value of
  -faset to "" (i.e. no faset in output), fixed bug for Abaqus output,
  small bug fixes.

New in 1.9.2 (September 2011):
- module -T: added option -morpho for specifying the type of grain
  structure (equiaxed, columnar or bamboo), merged option -regular
  with -morpho, added post-processing -neighbour option for information
  on the polyhedron neighbours, added geo (Gmsh geometry) output format
  (mostly for visualization), fixed bugs.
- module -MM: proper processing of the input tess files, added msh
  (Gmsh) and inp (Abaqus) output formats, added options -morpho and
  -centrecoo (as in module -T), small bug fixes, code cleaning.
- module -FM: added geo (Gmsh geometry) output format (mostly for
  visualization), small bug fixes.
- documentation: small corrections.

New in 1.9.1 (May 2011):
- module -FM: fixed bug occurring when -mesh3dalgo is not set by the
  user.  Small other bug fixes.
- module -MM: small bug fixes.

New in 1.9.0 (Apr 2011):
This is a major release.  Neper now has its own paper:
"R.Quey, P.R. Dawson and F. Barbe.  Large-scale 3D random polycrystal
 for the finite element method: Generation, meshing and remeshing.
 Computer Methods in Applied Mechanics and Engineering,
 Vol. 200, pp. 1729--1745, 2011."
Please cite it in your works if you use Neper.
- General: added option --rcfile to disregard / change the
  initialization file; big distribution and source clean up; bug fixes.
- module -T: added capability to generate regular morphologies
  (truncated octahedra), tess file format bumped to 1.9; big clean up.
```

```
- module -FM: included multimeshing, remeshing and mesh partitioning
  capabilities; big clean up.  Neper now uses the *standard* Gmsh
  distribution for 2D and 3D meshings (versions >=2.4.2).  Strongly
  reduced memory usage.
- module -O: added capability to handle different orientation
  descriptors.
- module -VS: new visualization module to generate publication-quality
  images (PNG format) of the tessellations, meshes and more...

New in 1.8.1 (Aug 2009):
- upgraded website at http://neper.sourceforge.net
- module -T: new file format ***tess1.8, new option -restart to load an
  existing tessellation (not through std input any more), new option
  -printformat, bug fixes.
- module -MM: bug fixes.
- module -FM: new output format mae, new option -restart to restart from
  an existing geometry or mesh (options -mesh and -conv removed); new
  options -printformat and -maeextension; better mesh numbering (+ new
  options -elementfirstid and -nodefirstid), new way to choose the node
  sets to output (-nset 4), fixed option -estat, renamed -bwcy-clmin to
  -clmin, cleaned bunch of options, bug fixes.
- module -O: added option -eulerangleconvention (Bunge, Roe & Kocks);
  new output formats mae and geof (option -format).
- manual: some corrections.

New in 1.8.0 (Jul 2009):
- First GPL-distributed version of Neper.
```

Appendix E GNU General Public License

GNU General Public License

Version 3, 29 June 2007

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program—to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

   "This License" refers to version 3 of the GNU General Public License.

   "Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

   "The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

   To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

   A "covered work" means either the unmodified Program or a work based on the Program.

   To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a devel copy. Propagation includes copying, distribution (with or without modification), making available to the distrib, and in some countries other activities as well.

   To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

   An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

   The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

   A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

   The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the distrib in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

   The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms

that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

   All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

   You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

   Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

   No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

   When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

   You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

   You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

   You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

   a. The work must carry prominent notices stating that you modified it, and giving a relevant date.

   b. The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

c. You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d. If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c. Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d. Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e. Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general distrib at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is distribly documented (and with an implementation available to the distrib in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

   a. Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b. Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c. Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d. Limiting the use for distribity purposes of names of licensors or authors of the material; or

e. Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f. Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

   You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

   Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

   You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

    Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

    An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

    You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

    A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

    A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

    Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

    In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

    If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a distribly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

    If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

    If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

    Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

    The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

    Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

    If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's distrib statement of acceptance of a version permanently authorizes you to choose that version for the Program.

    Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

    THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRIT-

ING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

    IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

    If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the distrib, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
one line to give the program's name and a brief idea of what it does.
Copyright (C) year name of author

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or (at
your option) any later version.

This program is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see http://www.gnu.org/licenses/.
```

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
program Copyright (C) year name of author
```

```
This program comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’.
This is free software, and you are welcome to redistribute it
under certain conditions; type ‘show c’ for details.
```

The hypothetical commands ‘`show w`’ and ‘`show c`’ should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an “about box”.

You should also get your employer (if you work as a programmer) or school, if any, to sign a “copyright disclaimer” for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see http://www.gnu.org/licenses/.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read http://www.gnu.org/philosophy/why-not-lgpl.html.

Option Index