DSixOutputConverter - Manual

Andreas Nicolai

andreas.nicolai@tu-dresden.de

Version 2.2.0, April 2020

Table of Contents

1. Overview

The `DSixOutputConverter` tool assists in automatically processing output files in *DataIO*-format, that are generated from IBK simulation tools (DELPHIN and NANDRAD). The *DataIO*-format is primarily designed to hold simulation results for programs solving *partial differential equations (PDE)*.

1.1. Quick summary of DataIO-format

Details of the `DataIO`-file format can be found in the publication

Stefan Vogelsang and Andreas Nicolai, 2011, *Delphin 6 Output File Specification*, https://nbn-resolving.org/urn:nbn:de:bsz:14-qucosa-70337

Output files can be written either as ASCII-files or as binary files. The latter tend to be a bit smaller, yet cannot be compressed as good as ASCII-files. Accessing large data sets in binary format is usally much more efficient than parsing ASCII-files, so whenever large 2D or 3D data sets are to be processed, the binary format is recommended.

There are two different file types:

- **data files** - contain the actual time series data
- **geometry files** - contain the geometry information (only stored once per project)

The files have specific extensions:

- ASCII output files have the extension `d6o` (data files) and `g6a` (geometry files).
- binary output files have the extension `d6b` (data files) and `g6b` (geometry files).

Data files generally hold transient data (date for each output time point). Output time point frequency can vary among data files, even from the same project (e.g. longer intervals for large fields, smaller intervals for integral values).

Data files for 3D, 4D and 5D datasets require the geometry file to be present. 2D data files (i.e. time series of scalar values) do not require geometry files.

1.1.1. Geometry files

The geometry files store basically the element and side geometry to visualize the calculation grid. All elements and sides are numbered, so that actual values can reference the respective element/side via their indexes.

1.1.2. Data files

Data files contain output data of a specific physical quantity (e.g. temperature or relative humidity), thus each data file only has values of one physical unit (the value unit). The type of data, the time and value units and captions are stored in the header of the file.

For each calculated time point data is written, in ASCII-Format each row means one time point.

The values are associated with element/side indexes, so that they can be shown in a graphical representation, accordingly.

Reference data files

A special subset of 2D data files is the REFERENCE format, basically a table of scalar values, where the quantity string encodes the captions of the individual columns. The data in this file is meant to group individual 2D data sets (i.e. time-value pairs), so that the overhead of writing the time column in each file is remedied.

1.1.3. Analysing DataIO-Files

The PostProc2 tool is the primary means of analysing and visualizing DataIO data.

1.2. DSixOutputConverter Feature Set

The purpose of the `DSixOutputConverter` converter tool is to automate specific conversion operations, that require otherwise manually written scripts or very laborious manual copy+paste work.

Here are some of the things, you can do with the tool:

- inspect the contents of a data file (show header analysis), output can be parsed by scripts for error checking
- convert from ASCII to binary format
- convert to other analysis formats, mainly TECPLOT format
- convert 2D REFERENCE data files to tab-separated-value or comma-separate-value format
- compute difference between two data files (i.e. from a variant analysis)
- extract only the last time point (i.e. extract final results)
- extract construction lines and generate a file with theses lines to plot in a different program
- extract a subset of data, i.e. a given time range
- scale all values by a given factor
- change the time unit (and convert the time point values accordingly)
- extract a slide across the data set, for example cut a plane out of a 3D data set
- mirror the data and geometry to left, right, top or bottom, and create a new geometry and data file that holds original and mirrored construction side-by-side (can be used to reconstruct original geometry that was calculated as symmetric half-construction)

2. General Usage

There are some common argument to the command line tool:

`--verbosity-level=<0|1|2|3|4>`
: Controls the verbosity of information and debug messages, Use `--verbosity-level=0` to disable all output (except required output from *list* command.

`-h, --help`
: Shows a list of all command line arguments.

`-v, --version`
: Shows version information.

`-x, --close-on-exit`

Closes the command line window (Windows only).

`--cmd-line`

Prints the individual parts of the command line, that were understood by the command line parser (useful to debug problems with execution, e.g. when paths with spaces/special characters are used).

For many commands, the following options may be applied:

`--output=<output filename>`

Specifies the output file name and extension to be used (otherwise, output file names will be generated depending on the selected operation).

`--skip-geofile`

Even if a transformation operation results in changed geometry files, it will not be written. This is useful if several data files are transformed, but they all use the same (new) geometry file.

`--overwrite-geofile`

By default existing geometry files will not be overwritten and commands will abort, when they would overwrite a geometry file. Use this option to explicitely overwrite geometry files.

`-g=<geo file>, --geofile=<...>`

When reading a geometry file, the file path is normally taken from the data file's header section. If you want to use another geometry file, you can specify a path to this file with this option.

Other options are specific to individual operations and are explained below.

3. Operations

The general three-step process when using DSixOutputConverter looks like that:

1. read data file(s)
2. apply data transformation operation
3. write file(s) in target format

3.1. list

Shows a summary of the data format and content of a data file.

Syntax:

```
> DSixOutputConverter list <data file>
```

Example:

```
> DSixOutputConverter list field-temperature.d6o
```

gives:

```
Reading data file 'field-temperature.d6o'.
  1 ms  for reading input DataIO file.
  0 ms  for parsing 3 datasets.
Reading geometry file 'LShape_3706993807.g6a'.
  0 ms  for reading input geometry file.
```

```
--- field-temperature.d6o ---
Output type    : FIELD
Quantity       : Temperature
Space type     : SINGLE
Time type      : NONE
Value unit     : C
Time unit      : d
Start year     : 2000
Data format    : 4D [time, matrix]
Elements       : 28
Materials      : 1
Geometry       : Plane 2D structure, 7 columns, 6 rows
Time points    : 3
--------------------------------
```

With minimal verbosity level:

```
> DSixOutputConverter list field-temperature.d6o --verbosity-level=0
```

the output is:

```
Output type    : FIELD
Quantity       : Temperature
Space type     : SINGLE
Time type      : NONE
Value unit     : C
Time unit      : d
Start year     : 2000
Data format    : 4D [time, matrix]
Elements       : 28
Materials      : 1
Geometry       : Plane 2D structure, 7 columns, 6 rows
Time points    : 3
```

which can be easily parsed by a script or just help to check the contents of a file.

3.2. ascii

Reads a data file (either in binary or ASCII-format) and writes it in ASCII format (possibly applying data transformation operations). A typical use of the operation is the conversion from binary to ASCII format.

When writing the data file, also the associated geometry file is written in ASCII-format (see also options `--skip-geofile` or `--overwrite-geofile`).

Example command line:

```
> DSixOutputConverter ascii profile-temperature-x.d6b
```

with output:

```
Reading data file 'profile-temperature-x.d6b'.
  0 ms  for reading input DataIO file.
  0 ms  for parsing 3 datasets.
Reading geometry file 'LShape_3706993807.g6b'.
  0 ms  for reading input geometry file.
```

```
--- profile-temperature-x.d6b (to ASCII) ---
Writing 'profile-temperature-x.d6o'.
 *  0 ms  for writing output DataIO file.
Writing 'LShape_3706993807.g6a'.
 *  0 ms  for writing output geometry file.
--------------------------------
```

The operation has generated files `profile-temperature-x.d6o` and `LShape_3706993807.g6a`.

You can apply data transformation flags to this operation.

If you want to use data transformation operations, but keep the ASCII-Format, you can also use the `ascii` operation. This only means "write in ASCII" but does not require data to be read in binary format.

3.3. bin

Reads a data file (either in binary or ASCII-format) and writes it in binary format (possibly applying data transformation operations). A typical use of the operation is the conversion from ASCII to binary format.

When writing the data file, also the associated geometry file is written in binary-format (see also options `--skip-geofile` or `--overwrite-geofile`).

The operation is otherwise identical to the `ascii` command.

3.4. tecplot

Reads a data file (and its geometry file), and exports a data file in format suitable for the TECPLOT post-processing software. You may need to specify additional arguments depending on the type of data file you want to generate.

You can apply data transformation flags to this operation.

The conversion into a Tecplot format is depending on the data format of the file:

- 2D data: Tecplot X-Y-format with data- and layout-files.
- 3D data: Tecplot Surface data or Standard Texplot format including relevant Makro for loading the data

- 4D data: depending on the order flags 4 versions are possible:
  - Tecplot 2D Finite elements with attached Makro.
  - Tecplot Geometrie file with attached Makro.
  - Tecplot 2D Vektor file with attached Makro.
  - Standard Tecplot file with attached Makro.

3.5. generic

Reads data file and exports it in generic format. *generic* format means to write data in plain text format, so that it can be easily imported in other software.

The output format is further determined by the flags: `--csv`, `--tsv` and `--block`, as described below.

The generic format (especially the block format) is quite useful for getting a quick look at the actual numbers, especially when working on a remote system via command line, when there is no graphical Post-Processing available. Except for the csv and tsv variants, the data is formatted and aligned such, that it can be nicely viewed in text editors.

3.5.1. 2D Data Files

2D data files contain a time and value column. In case of reference-type data containers, also several data columns can be stored (see section 2D Reference Data Files).

For plain 2D data files, there are different output formats, all only slightly different.

Plain format

When running the converter without further arguments, for example:

```
> DSixOutputConverter generic integrals.d6o
```

the converter will write a simple table of value pairs, each line in the text file contains two values, separated by a single space and a single tabulator character, for example:

```
1        0.982
2.5      1.123
4        3.2221
```

Unless the option `--output` is specified, the output file name is the same as the input filename, with the extension `txt`.

The units of the time points and values are the same as used in the data file. If necessary, refer to the header information in the DataIO data file (see List operation to obtain these values.

This plain output format is particular useful when feeding the resulting file into other command line post-processing tools, for example GnuPlot.

Comma separated values and tabulator separated values

By specifying the flag `--csv` or `--tsv` you can select a slightly different output format. For example, the data file with content:

```
D6OARLZ! 007.000
TYPE          = FIELD
PROJECT_FILE  = Absorption_with_gravity_Kl.d6p
CREATED       = Mon Apr 27 15:57:18 2020
GEO_FILE      = Absorption_with_gravity_Kl_1491615742.g6a
GEO_FILE_HASH = 2959590413
QUANTITY      = Total mass density of liquid water, water vapor and ice
QUANTITY_KW   = MoistureMassDensity
SPACE_TYPE    = INTEGRAL
TIME_TYPE     = NONE
VALUE_UNIT    = kg
TIME_UNIT     = h
START_YEAR    = 2007
INDICES       = 0 1 2 3 4 5
```

```
0               0.973987
0.0166667       6.32897
0.0333333       8.40736
0.05            10.0081
0.0666667       11.3503
0.0833333       12.53
0.1             13.5949
```

will be converted with the following command line:

```
> DSixOutputConverter generic --csv moist_integral.d6o
```

to `moist_integral.csv`:

```
Time [h],"Total mass density of liquid water, water vapor and ice [kg]"
0,0.973987
0.0166667,6.32897
0.0333333,8.40736
0.05,10.0081
0.0666667,11.3503
0.0833333,12.53
0.1,13.5949
```

and with

```
> DSixOutputConverter generic --tsv moist_integral.d6o
```

to `moist_integral.tsv`:

```
Time [h]    Total mass density of liquid water, water vapor and ice [kg]
0   0.973987
0.0166667   6.32897
0.0333333   8.40736
0.05    10.0081
0.0666667   11.3503
0.0833333   12.53
0.1 13.5949
```

Both formats are also directly usable with *Postproc 2* and can be copied into/opened directly with spreadsheet programs.

2D Reference Data Files

2D reference type files are data files with several scalar variables stored in a single file. For each time point, several values are given. Such data can be converted to generic data table format, hereby using either comma or tab-separated values. This is controlled with the `--csv` or `--tsv` flags.

Reference data files require either `--csv` or `--tsv`!

Example command line:

```
> DSixOutputConverter generic --tsv Zones_AirTemperature.d6o
```

Will read the file `Zones_AirTemperature.d6o` in reference format:

```
D6OARLZ! 007.000
TYPE          = REFERENCE
PROJECT_FILE  = MileStone1_Passive
CREATED       = Fri Jan 17 09:16:06 2020
GEO_FILE      =
GEO_FILE_HASH = 0
QUANTITY      = 1 'Meeting room' | 2 'Office'
QUANTITY_KW   = AirTemperature
SPACE_TYPE    = SINGLE
TIME_TYPE     = NONE
VALUE_UNIT    = C
TIME_UNIT     = d
START_YEAR    = 2001
INDICES       = 1 2
```

```
0               12              20
0.04166667      9.106795        9.126668
0.08333333      9.000735        8.142881
0.125           8.925145        7.665662
0.1666667       8.881829        7.38817
0.2083333       8.894107        7.309933
0.25            8.85182         7.225131
0.2916667       8.780251        7.095621
...
```

and convert it to file `Zones_AirTemperature.tsv` in tsv-format:

```
Time [d]	1 'Meeting room' [C]	2 'Office' [C]
0	12	20
0.0416667	9.10679	9.12667
0.0833333	9.00074	8.14288
0.125	8.92515	7.66566
0.166667	8.88183	7.38817
0.208333	8.89411	7.30993
0.25	8.85182	7.22513
0.291667	8.78025	7.09562
...
```

When using `--csv` as argument, the resulting file will be named `Zones_AirTemperature.csv` and have the content:

```
Time [d],1 'Meeting room' [C],2 'Office' [C]
0,12,20
0.0416667,9.10679,9.12667
0.0833333,9.00074,8.14288
0.125,8.92515,7.66566
0.166667,8.88183,7.38817
0.208333,8.89411,7.30993
0.25,8.85182,7.22513
0.291667,8.78025,7.09562
...
```

Both files can be easily imported into spreadsheet software (like LibreOffice, Excel, Gnumeric etc.).

You can use `--output` to specify an alternative output file name.

3.5.2. 3D Data Files

3D data is obtained when time, value and coordinate tuple data is stored in a file.

Such data can be converted into two different formats: triplet format and block format.

By default, without passing any other option, the data is stored in a `txt` file as matrix, each line corresponds to a different time point. Per line all values at the different locations are printed, separated by spaces and tabs. The first value in each line stands for the time point. The first line in the file contains all x or y coordinates for the printed values:

```
---	0.05	0.15	0.25	0.35
1	0.982	0.990	1.01	1.1123
2.5	1.123	1.001	1.233	1.4322
4	3.2221	4.222	4.775	8.4233
```

In this example the x coordinates are defined as 0.05, 0.15, 0.25 and 0.35 m. For each time point, 1h, 2.5h, 4h the monitored value is given for each element. Again the units are the same as in the original output.

With the flag `--csv` or `--tsv` the data is exported in three column format:

```
"Time [d]","X","Temperature [C]"
0,0.0005,20
0,0.00165,20
0,0.003145,20
0,0.0050885,20
0,0.00761505,20
0,0.010899565,20
0,0.015169435,20
0.04166666667,0.0005,12.99030792
0.04166666667,0.00165,13.08689416
0.04166666667,0.003145,13.21114302
0.04166666667,0.0050885,13.37043312
0.04166666667,0.00761505,13.5737075
0.04166666667,0.010899565,13.83148257
0.04166666667,0.015169435,14.15553594
...
```

3.5.3. 4D Data Files

4D data is stored as series of tables separated by an empty line. Every table contains the 2D field data at a certain time point. The first row and column of the table contain the coordinates of the element centers. The value in cell 1, row 1 contains the time point.

For the elements in the grid which haven't been assigned materials (and are therefore not monitored) the placeholder value NaN is printed as defined constant for undefined values.

Consider this example for one table in a 4D output file:

```
5.12	0.05	0.15	0.25	0.35	0.45	0.55
0.05	0.982	0.990	1.01	NaN	NaN	NaN
0.15	1.123	1.001	1.233	NaN	NaN	NaN
0.25	3.2221	4.222	4.775	8.4233	9.2133	10.1122
0.35	4.341	6.12	7.122	9.2442	10.1412	10.2010
```

This data corresponds to a grid like:

```
x x x
x x x
x x x x x x
x x x x x x
```

and the 5.12 in first row and first column indicates time point 5.12 (in the time unit of the source data file).

3.6. isopleth

TODO

3.7. diff

With this command, the difference between two data files can be computed.

Example syntax:

```
> DSixOutputConverter diff "values 1.d6o" -a="values 2.d6o" --output="diff.d6o"
```

The values of the second file are subtracted from the first. Data format and type of the files stays the same.

When subtracting temperatures, the value unit is changed to K (Kelvin).

4. Data transformations

The conversion operations `bin`, `ascii`, `tecplot`, `generic` allow use of data transformation flags.

4.1. `--scale` - Scaling data values by given factor

Very simple transformation. Multiplies all values by the given factor. Useful for inverting the sign of flux outputs.

Example command line:

```
# multiply all values in profile-temperature-x.d6o with -1 and write result
# to file 'profile-temperature-x.d6o'
> DSixOutputConverter ascii --scale=-1 profile-temperature-x.d6b
```

4.2. `--mirror` - Mirroring of geometry and data

Especially with 2D and 3D simulations it is frequently possible to use symmetry properties of the model and thus reduce the size of the calculation domain. For example, the following simulation model

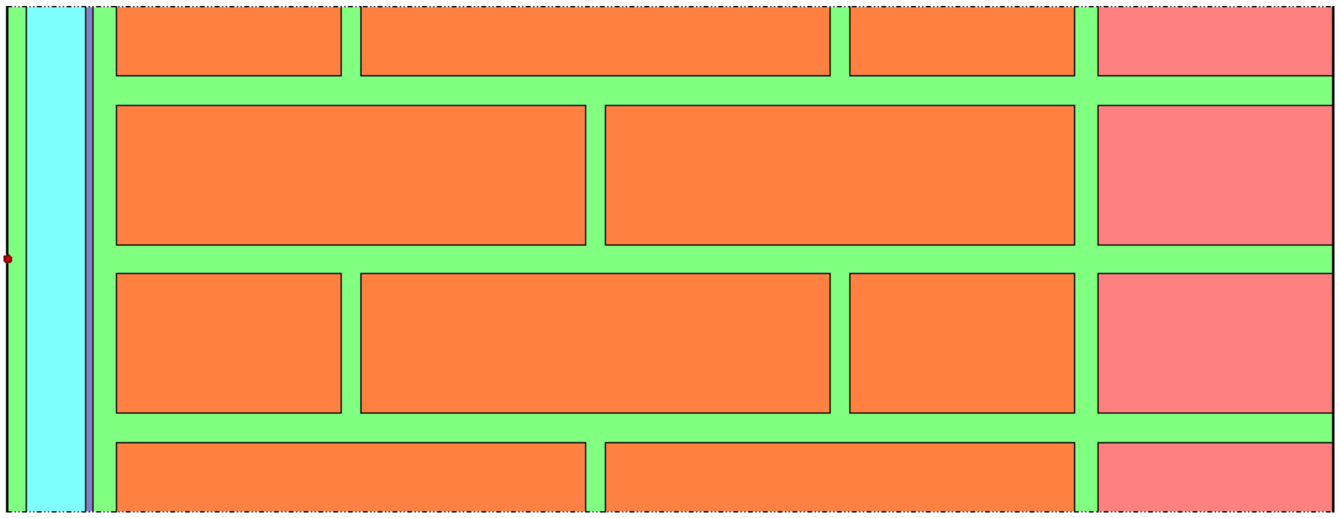
Figure 1. Complete model

can be replaced (exploiting symmetry) by:

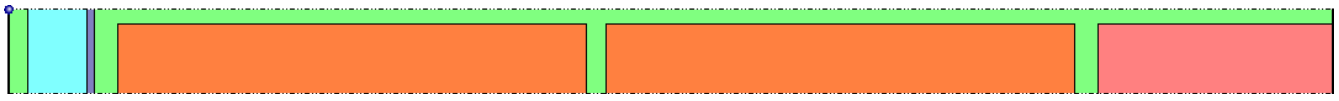
Figure 2. Reduced model (using symmetry)

The difference in simulation time (26 s for 2108 elements compared to 4:30 min for 14964 elements, for 30 d simulation time, each) is remarkable.

For the presentation of the calculation results in a report, however, a full view of the construction and results is often helpful. In the shown example, the brick/plaster geoemtry pattern would otherwise not be apparent.

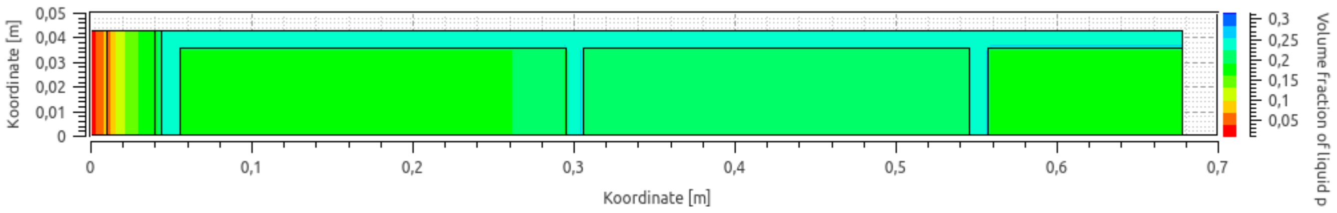

Figure 3. Simulation results for reduced model (using symmetry)

With the `--mirror` data transformation operation it is possible to mirror the construction geometry to the left (`xl`), to the right (`xr`), to the top (`yt`) and to the bottom (`yb`). The transformation is applied to the geometry and to the complete data set (all time points).

Example for the command line:

```
> DSixOutputConverter bin Field_SaturationDegree.d6b --mirror=yt --output=Field2x.d6b
```

As explained above, the operation `bin` means "write in binary format". The argument `--mirror=yt` tells the converter to place the mirror axis to the top of the construction. And `--output` specifies the resulting file name.

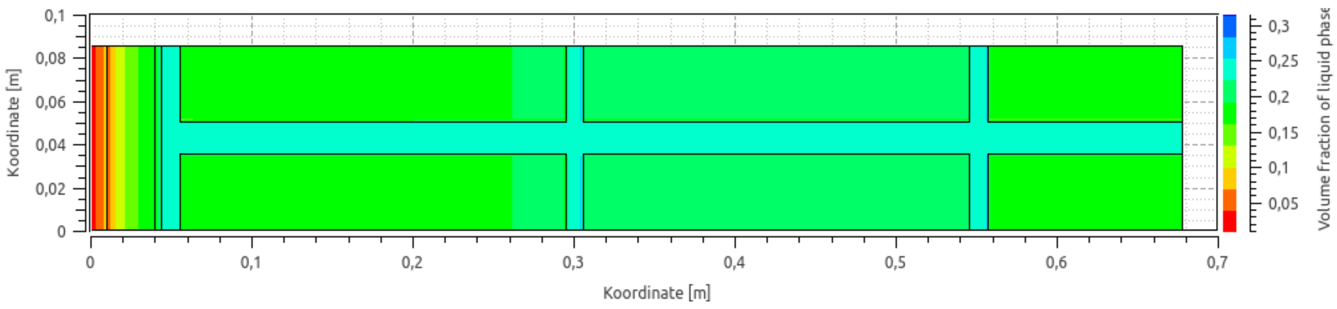

Figure 4. Simulation results mirrored once (to the top)

Another data transformation can be used to mirror again at the top axis:

```
> DSixOutputConverter bin Field2x.d6b --mirror=yt --output=Field4x.d6b
```

resulting in the final output:

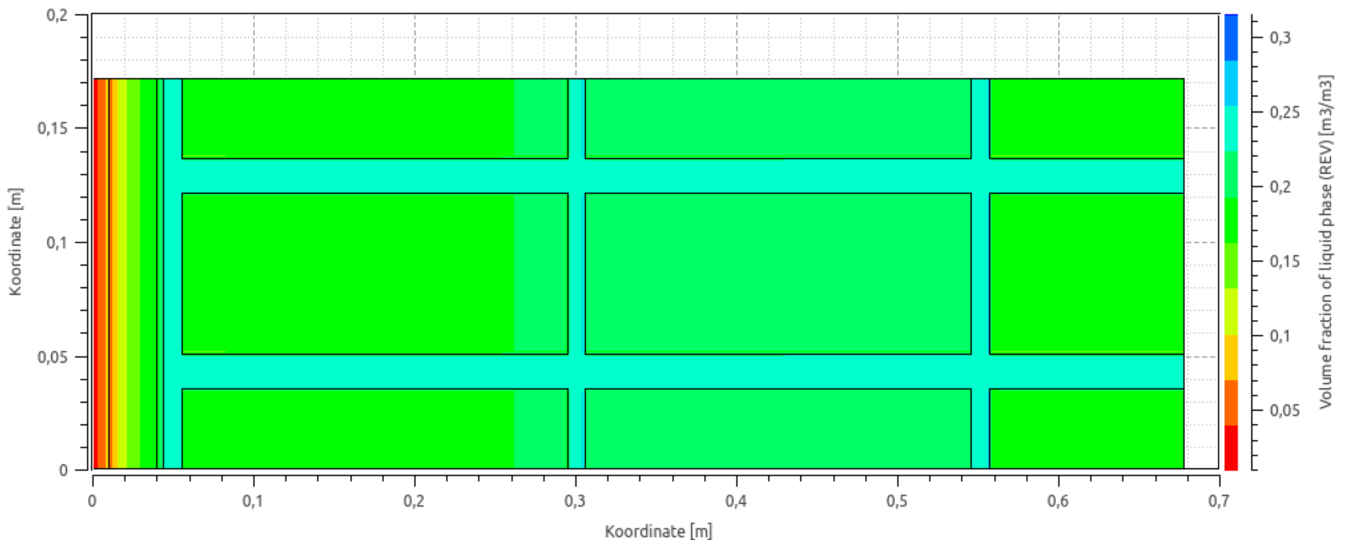

Figure 5. Simulation results mirrored twice

The operation can be made *in-place*, when `--output` points to the input file.

The mirror operation will generate a new geometry file. When analyzing the results with the PostProcessing, you need the new geometry file. See also the options `--overwrite-geofile` and `--skip-geofile`.

4.3. `--timeindex=<first, last>` - extract several time points between given time indexes

DataIO files contain data sets for different time points. These time points are numbered, starting with index 0 for the first time point.

Suppose an output file contains the following data lines:

Example DataIO data file

```
0           20          20          20          20
0.2         11.1297     6.85728     11.1297     6.85728
0.4         10.9863     6.76522     10.9863     6.76522
0.6         11.3529     7.44026     11.3529     7.44026
0.8         10.6092     5.9981      10.6092     5.9981
1           9.49113     3.99976     9.49113     3.99976
1.2         8.9502      4.11519     8.9502      4.11519
1.4         10.0403     5.73425     10.0403     5.73425
1.6         11.344      7.8672      11.344      7.8672
1.8         12.7176     9.57009     12.7176     9.57009
2           13.1497     10.2886     13.1497     10.2886
```

The first column holds the time point of the data set, in the unit specified in the header. In this example this is `d` (days). The simulation output time range spans 0 to 2 days, with a total of 11 time points.

You can now extract a subset of the data using the option `--timeindex` (and the other options described hereafter).

`--timeindex` takes either one or two arguments:

- if only one argument is given, only the time point with the index is kept,
- if two arguments are given (comma separated), the first is the index of the first time point to keep (index of the first data line), and the seconds is the index of the last line to keep.

Example syntax:

```
> DSixOutputConverter ascii temperature_field.d6o --timeindex="5,7" --output="temperature_field_subset.d6o" --skip
-geofile
```

If the command above is applied to the Example DataIO data file the following output is generated:

```
Reading data file 'temperature_field.d6o'.
  0 ms  for reading input DataIO file.
  0 ms  for parsing 11 datasets.
Reading geometry file 'xtime_plot_nogaps_1602485813.g6a'.
  0 ms  for reading input geometry file.
Keeping data sets 5..7, [ 1.000 d, 1.400 d]
Deleting data sets 0..4
Deleting data sets 8..10
```

```
--- temperature_field.d6o (to ASCII) ---
Writing 'temperature_field_subset.d6o'.
```

The lines with index 5, 6 and 7 are kept (corresponding to time points 1, 1.2 and 1.4 d).

If you use the `ascii` conversion option to create an ASCII output and also use ASCII input files, you need to specify both the target output data file name (with option `--output`) *and* that you do not wish to write the geometry file again (option `--skip-geofile`). After the operation, the original data file and the new data file will both reference the same geometry file.

Also, mind to use quotes around the time index range.

4.4. `-l, --last` - extract last time point

Using the `-l` or `--last` flag, only the last data set for the last time point is kept.

This is a convenience function and equivalent to specifying `--timeindex=11` in the example Example DataIO data file (passing the index of the last time point).

4.5. `--time=<time point>` - extract values at a specific time points (with interpolation)

This function is similar to the variant with `--timeindex` and giving only one index argument. It extracts a single data line.

The function determines the dataset and time point closest to the given time and, if there is no exact match, interpolated linearly between values. If the following command line is used on Example DataIO data file:

```
> DSixOutputConverter ascii temperature_field.d6o --time="1.1" --output="temperature_field_subset.d6o" --skip-geofile
```

The following output is created:

```
1.1             9.22067         4.05748         9.22067         4.05748
```

which is the interpolated value between the original data lines:

```
1               9.49113         3.99976         9.49113         3.99976
1.2             8.9502          4.11519         8.9502          4.11519
```

If the time point is larger than the last time point, an error is generated, for example when passing `--time=4` as argument:

```
Interpolation time point 4 (argument to --time) out of time point range [2,2] in DataIO.
```

4.6. `--timeslice=<startTime, endTime>` - extract several time points between given time span (no interpolation)

This time slicing data transformation extracts a subset of data points between the given time points. Here, the range of datasets is delimited by time points.

Example syntax:

```
> DSixOutputConverter ascii temperature_field.d6o --timeslice="1.1,4" --output="temperature_field_subset.d6o" --skip
-geofile
```

All data sets with time points within the range (limits including) are kept. Applied to Example DataIO data file this gives:

```
1               9.49113         3.99976         9.49113         3.99976
1.2             8.9502          4.11519         8.9502          4.11519
1.4             10.0403         5.73425         10.0403         5.73425
1.6             11.344          7.8672          11.344          7.8672
1.8             12.7176         9.57009         12.7176         9.57009
2               13.1497         10.2886         13.1497         10.2886
```

The `--timeslice` transformation does not require time point ranges to be within the available time slices of the output files. As in the example above, the upper limit may be well outside the time point range in the data file.

If time points are rounded and specifying the exact number isn't easily possible, the `--timeslice` transformation can also be used to bracket time points with some offset around it, like using `--timeslice="0.32,0.34"` to get all time points of format `0.33` or `0.33333`, etc.