

The Therion Book

Stacho Mudrák

Martin Budaj



Therion is copyrighted software. Distributed under the GNU General Public License.
Copyright © 1999–2004 Stacho Mudrák, Martin Budaj

We owe thanks to
Martin Sluka for his support concerning not only Therion,
Martin Heller for ideas
and *Wookey* for improving the documentation.

The cover picture shows survey sketch of the *Hrozný kameňolom* part of the Dead Bats Cave in Slovakia and the map of it produced by Therion.

Contents

Introduction	5
Why Therion?	5
Features	6
Software requirements	7
Installation	7
Setting-up environment	7
How does it work?	8
First run	9
Creating data files	10
Basics	10
Data types	11
Data format	11
‘encoding’	12
‘input’	12
‘survey’	12
‘centreline’	13
‘scrap’	16
‘point’	17
‘line’	20
‘area’	22
‘map’	23
‘join’	24
‘grade’	24
‘revise’	25
XTherion	25
XTherion—text editor	26
XTherion—map editor	26
Keyboard and mouse shortcuts in the Map editor	29
Thinking in Therion	31
How to enter centreline?	31
How to draw maps?	31
How to create models?	32
Processing data	33
Configuration file	33
‘encoding’	33
‘input’	33
‘source’	33
‘select’	33

‘unselect’	34
‘layout’	34
‘export’	38
Running Therion	40
XTherion—compiler	41
What we get?	42
Information files	42
Log file	42
XTherion	42
SQL export	42
2D maps	43
3D models	44
XTherion—model viewer	44
Changing layout	45
Page layout in the atlas mode	45
Page layout in the map mode	50
Customizing text labels	50
New map symbols	51
Point symbols	52
Line symbols	52
Area symbols	53
Special symbols	54
Appendix	55
Compilation	55
Quick start	55
Hacker’s guide	56
Environment variables	57
Initialization files	57
Therion	57
XTherion	59
Example data	59
History	61
Future	62
General	62
2D maps	62
3D models	62

Introduction

Therion is a tool for cave surveying. Its purpose is to help

- archive survey data on computer in a form as close to the original notes and sketches as possible and retrieve them in a flexible and efficient way;
- draw a nice up-to-date plan or elevation map;
- create a realistic 3D model of the cave.

It runs on Unix, Linux, MacOS X and Win32 operating systems. Source code and Windows installer are available on the Therion web page (<http://therion.speleo.sk>).

Therion is distributed under the GNU General Public License.

Why Therion?

In the 1990s we've done a lot of caving and cave surveying. Some computer programs existed which displayed survey shots and stations after loop closure and error elimination. These were a great help, especially for large and complicated cave systems. We used the output of one of them—TJIKPR—as a background layer with survey stations for hand-drawn maps. After finishing a huge 166 page Atlas of Dead Bats Cave, we soon had a problem: we found new passages connecting between known passages and surveyed them. After processing in TJIKPR, the new loops influenced the position of the old surveys; most survey stations now had a slightly different position from before due to the changed error distribution. So we could either draw the whole Atlas again, or accept that the location of some places was not accurate—in the case of loops with a length of approximately 1 km there were sometimes errors of about 10 m—and try to distort the new passages to fit to old ones.

These problems remained when we tried to draw maps using some CAD programs. It was always hard to add new surveys without adapting the old ones to the newly calculated positions of survey stations in the whole cave. We found no program that was able to draw an up-to-date complex map (i.e. not just survey shots with LRUD envelope), in which the old parts are modified according to the most recent known coordinates of survey stations.

In 1999 we begun to think about creating own program for map drawing. We knew about programs which were perfectly suited for particular sub-tasks. There was **METAPOST**, a high level programming language for vector graphics description, Survox for excellent processing of survey shots, and T_EX for typesetting the results. We had only to glue them together. By Xmas 1999 we had a minimalistic version of Therion working for the

first time. This consisted only of about 32 kB of Perl scripts and **METAPOST** macros but served the purpose of showing that our ideas were implementable.

During 2000–2001 we searched for the optimal format of the input data, programming language, concept of interactive map editor and internal algorithms with the help of Martin Sluka (Prague) and Martin Heller (Zürich). In 2002 we were able to introduce the first really usable version of Therion, which met our requirements.

Features

Therion is a command-line application. It processes input files, which are—including 2D maps—in text format, and creates files with 2D maps or 3D model as the output.

The syntax of input files is described in detail in later chapters. You may create these files in an arbitrary plain text editor like *ed* or *vi*. They contain instructions for Therion like

```
point 1303 1004 pillar
```

where **point** is a keyword for point symbol followed by its coordinates and a symbol type specification.

Hand-editing of such files is not easy—especially when you draw maps, you need to think in spatial (Cartesian coordinate) terms. Thus there is a special GUI for Therion called XTherion. XTherion works as an advanced text editor, map editor (where maps are drawn fully interactively), compiler (which runs Therion on the data) and 3D model viewer.

It may look quite complicated, but this approach has a lot of advantages:

- There is strict separation of data and visualization. The data files specify only what is where, not what it looks like. The visual representation is added by **METAPOST** in later phases of data processing. (It's very similar concept to XML data representation.)

This makes it possible to change map symbols used without changing the input data, or merge more maps created by different persons in different styles into one map with unified map symbols set.

2D maps are adapted for particular output scale (level of abstraction, non-linear scaling of symbols and texts)

- All data are relative to survey station positions. If the coordinates of survey stations are changed in the process of loop closure, then all relevant data is moved correspondingly, so the map is always up-to-date.
- Therion is not dependent on particular operating system, character encoding or input files editor; input files will remain human readable
- It's possible to add new output formats

- 3D model is generated from 2D maps to get a realistic 3D model without entering too much data
- although the support for WYSIWYG is limited, you get what you want

Software requirements

“A program should do one thing, and do it well.” (Ken Thompson) Therefore we use some valuable external programs, which are related to the problems of typesetting and data visualization. Therion can then do its task much better than if it was a standalone application in which you could calibrate your printer or scanner and with one click send e-mail with your data.

Therion needs:

- T_EX distribution. Necessary only if you want to create 2D maps.
- Tcl/Tk with *BWidget*, *Tom* and optionally *tkImg* extension. It is only required for XTherion.

Windows installer includes T_EX and Tcl/Tk distributions with all required extensions. Read the *Appendix* if you want to compile Therion yourself.

For displaying of maps and models you may use any of the following programs:

- any PDF viewer like Acrobat Reader, ghostscript or xpdf for displaying 2D maps;
- appropriate 3D viewer for models exported in other than default format;
- any SQL database client to process exported database.

Installation

Installation from sources (therion-0.3.*.tar.gz package):

The source code is a primary Therion distribution. It needs be compiled and installed according to instructions in the *Appendix*.

Installation on Windows:

Run the setup program and follow instructions. It installs all the required stuff and creates shortcuts to XTherion and Therion Book.

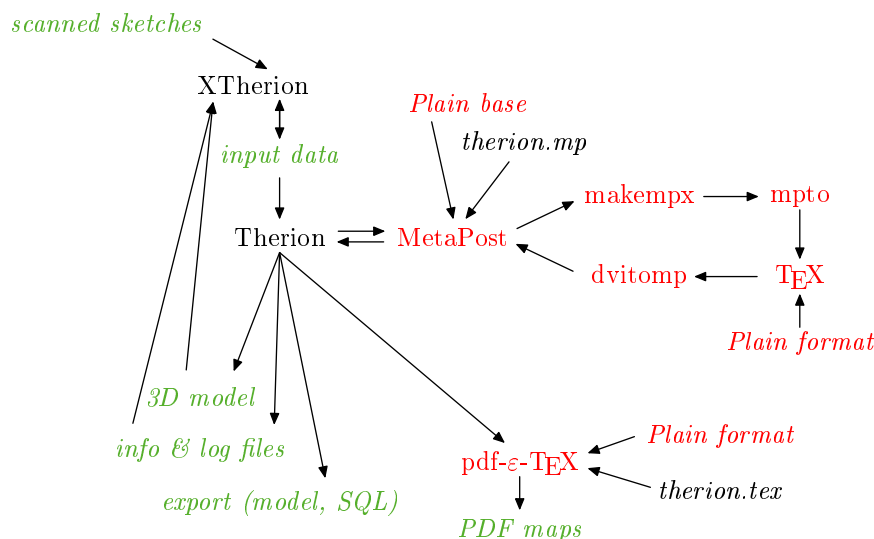
Setting-up environment

Therion reads settigs from the initialization file. Default settings should work fine for users using only ASCII (non-accented latin) characters, standard T_EX and METAPOST.

If you want to use accented latin or non-latin characters, edit initialization file. Instructions on how to do this are in the *Appendix*.

How does it work?

So, now it's clear what Therion needs, let's have a look at the way it interacts with all these programs:



DON'T PANIC! When your system is set-up right the majority of this is hidden from the user and all necessary programs are run automatically by Therion.

For working with Therion it is enough to know that you have to create input data (best done with XTherion), run Therion, and display output files (3D model, PDF map, log file) in the appropriate program.

For those who want to understand more about it, here is a brief explanation of the above flowchart. Program names are in roman font, data files in italics. Arrows show data flow between programs. Temporary data files are not shown. Meaning of colors:

- black—Therion programs and macros (XTherion is written in Tcl/Tk, so it needs this interpreter to run)
- red—**TEX** package
- green—input files created by the user and output files created by Therion

Therion itself does the main task. It reads the input files, interpretes them, finds closed loops and distributes errors. Next it transforms all other data (e.g. 2D maps) according to new stations position. Therion exports data for 2D maps in **METAPOST** format. **METAPOST** gives the actual shape to abstract map symbols according to map symbol definitions; it creates a lot of PostScript files with small fragments of the cave. These are read back and converted to a PDF-like format, which forms input data for pdf**TEX**. Pdf**TEX** does all the typesetting and creates a PDF file of the cave map.

Therion also exports 3D model (full or centreline) in various formats. The model in the `therion` format may be loaded and displayed in the XTherion's model viewer.

Centreline may be exported for further processing in any SQL database.

First run

After explaining the basic principles of Therion it's a good idea to try it on the example data.

- Download the sample data from Therion web page and unpack it somewhere on your computer's hard drive.
- Run XTherion (under Unix and MacOS X by typing 'xtherion' in the command line, under Windows there is a shortcut in the *Start* menu).
- Open the file 'thconfig' from the sample data directory in the 'Compiler' window of XTherion
- Press 'F9' or 'compile' in the menu to run Therion on the data—you'll get some messages from Therion, **METAPOST** and **T_EX**.
- PDF maps and 3D model are created in the data directory.

Additionally, you may open survey data files (*.th) in the 'Text editor' window and map data files (*.th2) in the 'Map editor' window of XTherion. Although the data format may look confusing for the first time, it will be explained in the following chapters.

Creating data files

Basics

The input files for Therion are in text format. There are a few rules about how such a file should look:

- There are two kinds of commands. One-line commands and multi-line commands.
- A one-line command is terminated by an end of line character. The syntax of these is

```
command arg1 ... argN [-option1 value1 -option2 value2 ...]
```

where *arg1 ... argN* are obligatory arguments, and pairs *-option value* are options, which you may freely omit. Which arguments and options are available depends on the particular command. An example may be

```
point 643.5 505.0 gradient -orientation 144.7
```

with three obligatory arguments and one optional option/value pair. Sometimes options have no or multiple values.

- Multi-line commands begin similarly to one line commands, but continue on subsequent lines until explicit command termination. These lines may contain either data or options, which are applied to subsequent data. If a data line starts with a word reserved for an option, you have to insert ‘!’ in front of it. The syntax is

```
command arg1 ... argN [-option1 value1 -option2 value2 ...]
```

```
...
optionX valueX
data
...
endcommand
```

Again, for better illustration, a real example follows:

```
line wall -id walltobereferenced
  1174.0 744.5
  1194.0 756.5 1192.5 757.5 1176.0 791.0
smooth off
  1205.5 788.0 1195.5 832.5 1173.5 879.0
endline
```

This command `line` has one obligatory argument, a line type (passage wall in this case), followed by one option. The next two lines contain data (coordinates of Bézier curves to be drawn). The next line (“`smooth off`”) specifies an option which applies

to subsequent data (i.e. not for the whole line, unlike the option `-id` in the first line) and the last line contains some more data.

- if the value of an option or argument contains spaces, you should enclose this value in " " or []. If you want to put a double-quote " into text in " " you need to insert it twice. Quotes are used for strings; brackets for numerical values and keywords.
- each line ending with a backslash (\) is considered to continue on the next line, as if there was neither line-break nor backslash.
- everything that follows #, until the end of line—even inside a command—is considered to be a comment, and is ignored.

Data types

Therion uses following data types:

- *keyword* ▷ a sequence of A-Z, a-z, 0-9 and _- characters (not starting with '-').
- *ext_keyword* ▷ keyword that can also contain +*/. , ' characters, but not on the first position.
- *date* ▷ a date (or a time interval) specification in a format
YYYY.MM.DD@HH:MM:SS.SS - YYYY.MM.DD@HH:MM:SS.SS or - to leave a date unspecified.
- *person* ▷ a person's first name and surname separated by whitespace characters. Use '/' to separate first name and surname where there are more names.
- *string* ▷ a sequence of any characters.
- *units* ▷ length units supported: meter[s], centimeter[s], inch[es], feet[s], yard[s] (also m, cm, in, ft, yd). Angle units supported: degree[s], minute[s] (also deg, min), grad[s], mil[s]. A degree value may be entered in decimal notation (*x.y*) or in a special notation for degrees, minutes and seconds (*deg[:min[:sec]]*).

Data format

The syntax of input files is explained in the description of individual commands. Studying the example files distributed with Therion will help you understand. See also an example in the *Appendix*.

Each of the following sections describes one Therion command using the following structure:

Description: notes concerning this command.

Syntax: schematic syntax description.

Context: specifies the context in which is this command allowed. The *survey* context means that the command must be enclosed by `survey ... endsurvey` pair. The *scrap* context means that the command must be enclosed within `scrap ... endscrap` pair. Context *all* means that the command may be used anywhere.

Arguments: a list of the obligatory arguments with explanations.

Options: a list of the available options.

Command-like options: options for multi-line commands, which can be specified among the data lines.

‘encoding’

Description: sets the encoding of input file. This allows the use of non-ASCII characters in input files.

Syntax: `encoding <encoding-name>`

Context: It should be the very first command in the file.

Arguments:

- `<encoding-name>` ▷ to see a list of all the supported encoding names, run Therion with `--print-encodings` option. ‘UTF-8’ (Unicode) and ‘ASCII’ (7 bit) encodings are always supported.

‘input’

Description: inserts the contents of a file in place of the command. Default extension is ‘.th’ and may be omitted. For greatest portability use relative paths and Unix slashes ‘/’, not Windows backslashes ‘\’, as directory separators.

Syntax: `input <file-name>`

Context: all

Arguments:

- `<file-name>`

‘survey’

Description: Survey is the main data structure. Each data object must belong to a survey. Surveys may be nested—this allows a hierarchical structure to be built.

Each survey has its own namespace specified by its `<id>` argument. Objects (like survey stations or scraps; see below) which belong to a subsurvey of the current survey are referenced as

<object-id>@<subsurvey-id>,

or, if there are more nesting levels

<object-id>@<subsubsurvey-id>.<subsurvey-id>.

This means, that object identifiers must be unique only in the scope of one survey. For instance, survey stations names can be the same if they are in different surveys. This allows stations to be numbered from 0 in each survey or the joining of two caves into one cave system without renaming survey stations.

Syntax: survey <id> [OPTIONS]
... other therion objects ...
endsurvey [<id>]

Context: none, survey

Arguments:

- <id> ▷ survey identifier

Options:

- **declination** <specification> ▷ set the default declination for all data objects in this survey (which can be overridden by declination definitions in subsurveys). The <specification> has three forms:
 1. [] an empty string. This will reset the declination definition.
 2. [<value> <units>] will set a single value (also for undated surveys).
 3. [<date1> <value1> [<date2> <value2> ...] <units>] will set declination for several dates. Then the declination of each shot will be set according to the date specification of the data object. If you want to explicitly set the declination for undated survey data, use ‘-’ instead of date.
- **person-rename** <old name> <new name> ▷ rename a person whose name has been changed
- **title** <string> ▷ description of the object

‘centreline’

Description: Survey data (centreline) specification. The syntax is borrowed from Survex with minor modifications; the Survex manual may be useful as an additional reference for the user. A synonym term ‘centerline’ may be used.

Syntax: centreline [OPTIONS]
date <date>
team <person> [<roles>]
explo-date <date>
explo-team <person>

```

instrument <quantity list> <description>
calibrate <quantity list> <zero error> [<scale>]
units <quantity list> [<factor>] <units>
sd <quantity list> <value> <units>
grade <grade list>
declination <value> <units>
infer <what> <on/off>
mark <type>
flags <shot flags>
station <station> <comment> [<flags>]
fix <station> [<x> <y> <z> [<std x> <std y> <std z>]]
equate <station list>
data <style> <readings order>
break
...
[SURVEY DATA]
...
endcentreline

```

Context: survey

Options:

- `id <ext_keyword>` ▷ id of the object
- `author <date> <person>` ▷ author of the data and it's creation date
- `copyright <date> <string>` ▷ copyright date and name
- `title <string>` ▷ description of the object

Command-like options:

- `date <date>` ▷ survey date. If multiple dates are specified, a time interval is created.
- `explo-date <date>` ▷ discovery date. If multiple dates are specified, a time interval is created.
- `team <person> [<roles>]` ▷ a survey team member. The first argument is his/her name, the others describe the roles of the person in the team (optional—currently not used). The following role keywords are supported: station, length, tape, compass, bearing, clino, gradient, counter, depth, station, position, notes, pictures, instruments (insts), assistant (dog).
- `explo-team <person>` ▷ a discovery team member.
- `instrument <quantity list> <description>` ▷ description of the instrument that was used to survey the given quantities (same keywords as team person's role)
- `infer <what> <on/off>` ▷ 'infer plumbs on' tells the program to interpret gradients $\pm 90^\circ$ as UP/DOWN (this means no clino corrections are applied). 'infer equates

on' will case program to interpret shots with 0 length as equate commands (which means that no tape corrections are applied)

- **declination** <value> <units> ▷ sets the declination for subsequent shots

$$\text{true bearing} = \text{measured bearing} + \text{declination}.$$

If no declination is specified, or the declination is reset (-), then a valid declination specification is searched for in all surveys the data object is in. See declination option of survey command.

- **sd** <quantity list> <value> <units> ▷ sets the standard deviation for the given measurements. The Quantity list can contain the following keywords: length, tape, bearing, compass, gradient, clino, counter, depth, x, y, z, position, easting, dx, northing, dy, altitude, dz.
- **grade** <grade list> ▷ sets standard deviations according to the survey grade specification (see grade command). All previously specified standard deviations or grades are lost. If you want to change an SD, use the sd option after this command. If multiple grades are specified, only the last one applies. You can specify grades only for position or only for surveys. If you want to combine them, you must use them in one grade line.
- **units** <quantity list> [<factor>] <units> ▷ set the units for given measurements (same quantities as for sd).
- **calibrate** <quantity list> <zero error> [<scale>] ▷ set the instrument calibration. The measured value is calculated using the following formula: $\text{measured value} = (\text{read value} - \text{zero error}) \times \text{scale}$. The supported quantities are the same as sd.
- **break** ▷ can be used with interleaved data to separate two traverses
- **mark** <type> ▷ set the type of the station. <type> is one of: fixed, painted and temporary (default).
- **flags** <shot flags> ▷ set flags for following shots. The supported flags are: surface (for surface measurements), duplicate (for duplicate surveys). Both are excluded from length calculations. Also "not" is allowed before a flag.
- **station** <station> <comment> [<flags>] ▷ set the station comment and flags: entrance or continuation. If "" is specified as a comment, it is ignored.
- **fix** <station> [<x> <y> <z> [<std x> <std y> <std z>]] ▷ fix station coordinates (with specified errors—only the units transformation, not calibration, is applied to them).
- **equate** <station list> ▷ set points that are equivalent
- **data** <style> <readings order> ▷ set data style (normal, topofil, diving, cartesian, cypolar, nosurvey) and readings order. Reading is one of the following keywords: station, from, to, tape/length, [back]compass/[back]bearing, [back]clino/[back]gradient,

depth, fromdepth, todepth, depthchange, counter, fromcount, tocount, northing, easting, altitude, up/ceiling, down/floor, left, right, ignore. For interleaved data both newline and direction keywords are supported. If backward and forward compass or clino reading are given, the average of them is computed. See Survox manual for details.

'scrap'

Description: Scrap is a piece of 2D map, which doesn't contain overlapping passages (i.e. all the passages may be drawn on the paper without overlapping). For small and simple caves, the whole cave may belong to one scrap. In complicated systems, a scrap is usually one chamber or one passage. Ideally, a scrap contains about 100 m of the cave. Each scrap is processed separately by **METAPOST**; scraps which are too large may exceed **METAPOST**'s memory and cause errors.

Each scrap has its own local cartesian coordinate system, which usually corresponds with the millimeter paper (if you measure the coordinates of map symbols by hand) or pixels of the scanned image (if you use XTherion). Therion does the transformation from this local coordinate system to the real coordinates using the positions of survey stations, which are specified both in the scrap as point map symbols and in centreline data. If the scrap doesn't contain at least two survey stations, you have to use the **-scale** option for calibrating the scrap. (This is usual for cross sections.)

The transformation consists of the following steps:

- Linear transformation (shifting, scaling and rotation) which 'best' fits stations drawn in the scrap to real ones. 'Best' means that the sum of squared distances between corresponding stations before and after transformation is minimal. The result is displayed red if **debug** option of the **layout** command is set **on**.
- Non-linear transformation of the scrap which (1) moves survey stations to their correct position, (2) is continuous. Displayed blue in the **debug** mode.
- Non-linear transformation of the scrap which (1) moves joined points together, (2) doesn't move survey stations, (3) is continuous. Finally the position of curves' control points is adjusted to preserve smoothness. The result is final map.

Syntax: **scrap** <id> [OPTIONS]
... point, line and area commands ...
endscrap [<id>]

Context: survey

Arguments:

- <id> ▷ scrap identifier

Options:

- **projection** `<specification>` ▷ specifies the drawing projection. Each projection is identified by a type and optionally by an index in the form `type[:index]`. The index can be any keyword. The following projection types are supported:
 1. **none** ▷ no projection, used for cross sections or maps that are independent of survey data (e.g. digitization of old maps where no centreline data are available). No index is allowed for this projection.
 2. **plan** ▷ basic plan projection (default).
 3. **elevation** ▷ orthogonal projection which optionally takes a view direction as an argument (e.g. `[elevation 10]` or `[elevation 10 deg]`).
 4. **extended** ▷ extended elevation.
- **scale** `<specification>` ▷ is used to pre-scale (convert coordinates from pixels to meters) the scrap data. If scrap projection is none, this is the only transformation that is done with coordinates. The `<specification>` has four forms:
 1. `<number>` ▷ `<number>` meters per drawing unit.
 2. `[<number> <length units>]` ▷ `<number>` `<length units>` per drawing unit.
 3. `[<num1> <num2> <length units>]` ▷ `<num1>` drawing units corresponds to `<num2>` `<length units>` in reality.
 4. `[<num1> ... <num8> [<length units>]]` ▷ this is the most general format, where you specify, in order, the x and y coordinates of two points in the scrap and two points in reality. Optionally, you can also specify units for the coordinates of the ‘points in reality’.
- **stations** `<list of station names>` ▷ stations you want to plot to the scrap, but which are not used for scrap transformation. You don’t have to specify (draw) them with the `point station` command.
- **author** `<date>` `<person>` ▷ author of the data and it’s creation date
- **copyright** `<date>` `<string>` ▷ copyright date and name
- **title** `<string>` ▷ description of the object

‘point’

Description: Point is a command for drawing a point map symbol.

Syntax: `point <x> <y> <type> [OPTIONS]`

Context: scrap

Arguments:

- **<x>** and **<y>** are the drawing coordinates of an object.
- **<type>** determines the type of an object. The following types are supported:
 - special objects:* station, section, water-flow, spring, sink, air-draught, entrance, gradient;
 - labels:* label, remark, altitude, height, passage-height, station-name, date;
 - symbolic passage fills:* bedrock, sand, raft, clay, pebbles, debris, blocks, water, ice, guano;
 - speleothems:* flowstone, moonmilk, stalactite, stalagmite, pillar, curtain, helictite, soda-straw, crystal, wall-calcite, popcorn, disk, gypsum, gypsum-flower, aragonite, cave-pearl, rimstone-pool, rimstone-dam, anastomosis, karren, scallop, flute, raft-cone;
 - equipment:* anchor, rope, fixed-ladder, rope-ladder, steps, bridge, traverse, camp, no-equipement;
 - passage ends:* continuation, narrow-end, low-end, flowstone-choke, breakdown-choke;
 - others:* archeo-material, paleo-material, vegetable-debris, root.

Options:

- **subtype <keyword>** ▷ determines the object's subtype. The following subtypes for given types are supported:
 - station:* temporary (default), painted, natural, fixed;
 - water-flow:* permanent (default), intermittent, paleo
- **orientation/orient <number>** ▷ defines the orientation of the symbol. If not specified, it's oriented to north. $0 \leq \text{number} < 360$.
- **align** ▷ alignment of the symbol or text. The following values are accepted: center, c, top, t, bottom, b, left, l, right, r, top-left, tl, top-right, tr, bottom-left, bl, bottom-right, br.
- **scale** ▷ symbol scale, can be: tiny (xs), small (s), normal (m), large (l), huge (xl). Normal is default.
- **place <bottom/default/top>** ▷ where to place the symbol relatively to other objects.
- **clip <on/off>** ▷ specify whether a symbol is clipped by the scrap border. You cannot specify this option in the following symbols: station, station-name, label, remark, date, altitude, height, passage-height.
- **visibility <on/off>** ▷ displays/hides an object
- **id <ext_keyword>** ▷ id of the object

Type-specific options:

- **name** <reference> ▷ if the point type is station, this option gives the reference to the real survey station.
- **extend** <specification> ▷ if the point type is station and scrap projection is extended elevation, you can adjust the extension of the centreline using this option. The <specification> is a list of one or more following keywords:
 1. **left** ▷ indicates extension to the left
 2. **right** ▷ the opposite of above. If no extension direction is given, therion uses the direction of the previous station, or right, if no such station exists.
 3. **root** ▷ determines the starting node for extension.
 4. **sticky** <on/off> ▷ two keywords that identify whether other scraps can be attached to this station. The default is true for end-station.
 5. **previous/prev** <reference> ▷ reference to previous station (or point of type station)
- **scrap** <reference> ▷ if the point type is section, this is a reference to a cross-section scrap.
- **text** ▷ text of the label or remark. It may contain following formatting keywords:
 -
 ▷ line break
 - <center>/<centre>, <left>, <right> ▷ line alignment for multi-line labels. Ignored if there is no
 tag.
 - <thsp> ▷ thin space
 - <rm>, <it>, <bf>, <ss>, <si> ▷ font switches
- **value** ▷ value of height, passage-height or altitude label

Special notes: The following point types have specific behaviours:

altitude: the value specified is the altitude difference from the nearest station. If the altitude value is prefixed by “fix” (e.g. -value [fix 1300]), this value is used as an absolute altitude. The value can optionally be followed by length units.

height: according to the sign of the value (positive, negative or unsigned), this type of symbol represents chimney height, pit depth or step height in general. The numeric value can be optionally followed by ‘?’, if the value is presumed and units can be added (e.g. -value [40? ft]).

passage-height: the following four forms of value are supported: +<number> (the height of the ceiling), -<number> (the depth of the floor or water depth), <number> (the distance between floor and ceiling) and [<number> <number>] (the distance to ceiling and distance to floor).

station: in any projection (with the exception of ‘none’ projection), at least one station with station reference (`-name` option) has to be specified for each scrap.

station-name: if no text is specified, the name of the nearest station is used.

section: place the section at this point. The section scrap must be in ‘none’ projection. You can specify it through the `-scrap` option. This symbol has no visual representation.

spring, sink: always use these two symbols with a water-flow arrow.

symbolic passage fills: unlike other symbols, these are clipped by the scrap border.

air-draught: number of ticks is set according to `-scale` option

‘line’

Description: Line is a command for drawing a line symbol on the map. Each line symbol is oriented and its visualization may depend on its orientation (e.g. pitch edge ticks). The general rule is that the free space is on the left, rock on the right. Examples: the lower side of a pitch, higher side of a chimney and interior of a passage are on the left side of pitch, chimney or wall symbols, respectively.

Syntax: `line <type> [OPTIONS]`

```
    altitude <value>
    border <on/off>
    clip <on/off>
    close <on/off/auto>
    direction <begin/end/both/none/point>
    gradient <none/center/point>
    head <begin/end/both/none>
    mark <keyword>
    orientation/orient <number>
    outline <in/out/none>
    place <bottom/default/top>
    reverse <on/off>
    size <number>
    r-size <number>
    l-size <number>
    smooth <on/off/auto>
    subtype <keyword>
    text <string>
    ...
    [LINE DATA]
    ...
endline
```

Context: scrap

Arguments:

- **<type>** is a keyword that determines the type of line. The following types are supported:

passages: wall, contour, slope, floor-step, pit, ceiling-step, chimney, overhang

passage fills: flowstone, rock-border, rock-edge, water-flow

labels: label

special: border, arrow, section, survey

Command-like options:

- **subtype <keyword>** ▸ determines line subtype. The following subtypes are supported for given types:

wall: invisible, bedrock (default), sand, clay, pebbles, debris, blocks, ice, underlying, unsurveyed, presumed

border: visible (default), invisible, temporary

water-flow: permanent (default), conjectural, intermittent

- **[LINE DATA]** specify either the coordinates of a line segment **<x> <y>**, or coordinates of a Bézier curve arc **<c1x> <c1y> <c2x> <c2y> <x> <y>**, where **c** indicates the control point.
- **close <on/off/auto>** ▸ determines whether a line is closed or not
- **mark <keyword>** ▸ is used to mark the station on the line (see join command).
- **orientation/orient <number>** ▸ orientation of the symbols on the line. If not specified, it's perpendicular to the line on its left side. $0 \leq \text{number} < 360$.
- **outline <in/out/none>** ▸ determines whether the line serves as a border line for a scrap. Default value is 'out' for walls, 'none' for all other lines.
- **reverse <on/off>** ▸ whether points are given in reverse order.
- **size <number>** ▸ line width (left and right sizes are set to one half of this value)
- **r-size <number>** ▸ size of the line to the right
- **l-size <number>** ▸ same to the left. Required for **slope** type.
- **smooth <on/off/auto>** ▸ whether the line is smooth at the given point. Auto is default.
- **place <bottom/default/top>** ▸ where to place the symbol relative to other objects.
- **clip <on/off>** ▸ specify whether a symbol is clipped by the scrap border.
- **visibility <on/off>** ▸ displays/hides an object

Type-specific options:

- **altitude** <value> ▷ can be specified only with the wall type. This option creates an altitude label on the wall. The value gives the altitude difference of the point on the wall relative to the nearest station. The value can be prefixed by a keyword “fix”, then no nearest station is taken into consideration; the absolute given value is used instead. Units can follow the value. Examples: +4, [+4 m], [fix 1510 m].
- **border** <on/off> ▷ this option can be specified only with the ‘slope’ symbol type. It switches on/off the border line of the slope.
- **direction** <begin/end/both/none/point> ▷ can be used only with the section type. It indicates where to put a direction arrow on the section line. None is default.
- **gradient** <none/center/point> ▷ can be used only with the contour type and indicates where to put a gradient mark on the contour line. If there is no gradient specification, behaviour is symbol-set dependent (e.g. no tick in UIS, tick in the middle in SKBB).
- **head** <begin/end/both/none> ▷ can be used only with the arrow type and indicates where to put an arrow head. End is default.
- **text** <string> ▷ valid only for label lines.

Options:

- **id** <ext_keyword> ▷ id of the object

Special notes: The following line types have specific behaviour:

section: if both control points of a Bézier curve are given then the line is drawn up to the perpendicular projection of the first control point and from the projection of the section control point. No section curve is allowed.

survey: survey line is automatically drawn by Therion.

slope: slope line marks upper border of the slopy area. It’s necessary to specify **l-size** in at least one point. Gradient lines length and orientation is an average of specified **l-sizes** and **orientations** in the nearest points. If there is no orientation specification, gradient marks are perpendicular to the slope line.

rock-bord: if the line is closed, it is filled with the background colour.

‘area’

Description: Area is specified by surrounding border lines. They may be of any type, but must be listed in order and each pair of consecutive lines must intersect. In order to be sure that lines intersect even after scrap transformation you may e.g. continue a lake border 1 cm behind a passage wall—these overlaps will be automatically clipped by scrap border. You may use invisible border to achieve this inside of the passage.

Syntax: area <type>
 place <bottom/default/top>
 clip <on/off>
 visibility <on/off>
 ... border line references ...
 endarea

Context: scrap

Arguments:

- <type> is one of following: water, sump, sand, debris.

Command-like options:

- the data lines consist of border line references (IDs)
- place <bottom/default/top> ▷ where to place the symbol relative to other objects (bottom by default).
- clip <on/off> ▷ specify whether a symbol is clipped by the scrap border.
- visibility <on/off> ▷ displays/hides an object

Options:

- id <ext_keyword> ▷ id of the object

'map'

Description: A map is a collection of either scraps or other maps of the same projection type. It simplifies the data management when selecting data for output.

Syntax: map <id>
 ... scrap or other map references ...
 break
 ... next level scrap or other map references ...
 preview <above/below> <other map id>
 endmap

Context: survey

Arguments:

- <id> ▷ scrap identifier

Command-like options:

- the data lines consist of scrap or map references. Note that you can not mix them together.
- scraps following the break will be placed on another level

- **preview** <above/below> <other map id> will put the outline of the other map in the specified preview position relative to the current map.

Preview is displayed only if the map is in the **map-level** level as specified by the **select** command.

Use the **revise** command if you want to add maps from higher levels to the preview.

Options:

- **title** <string> ▷ description of the object

'join'

Description: Join works in two modes: it joins either two scraps or two or more points in a map together. When joining scraps, only passage walls are joined. It's a good idea to place a scrap join in the passage which is as simple as possible, otherwise you have to specify join for each pair of objects which should be joined.

If you want some object which is clipped by a scrap boundary to continue to a neighbouring scrap, use **-clip off** option for that object.

Syntax: join <point1> <point2> ... <pointN> [OPTIONS]

Context: scrap, survey

Arguments:

- <pointX> can be an ID of a point or line symbol, optionally followed by a line point mark <id>:<mark> (e.g. podangl_131@podangl:mark1). <mark> can be also 'end' (end of the line) or line point index (where 0 is the first point).

A special case is when <point1> and <point2> are scrap IDs—than the closest scrap ends are joined together.

Options:

- **smooth** <on/off> indicates whether two lines are to be connected smoothly.
- **count** <N> (when used with scraps) ▷ Therion will try to find more connections of given two scraps

'grade'

Description: This command is used to store predefined precisions of centreline data. See **sd** option description for **centreline** command.

Syntax: : grade <id>

```
...
[<quantity list> <value> <units>]
...
```


endgrade

Context: all

‘revise’

Description: This command is used to set or change properties of an already existing object.

Syntax: The syntax of this command for object created with “single line” command is

```
revise id [-option1 value1 -option2 value2 ...]
```

For objects created with “multi line” commands is syntax following

```
revise id [-option1 value1 -option2 value2 ...]
```

```
...
optionX valueX
data
...
endrevise
```

Context: all

Arguments:

The id stands for object identifier (the id of an object you want to revise must always be specified).

XTherion

XTherion is a GUI (Graphical User Interface) for Therion. It helps a lot with creating input data files. Currently it works in four main modes: text editor, map editor, compiler and model viewer. (Here we’re concerned with creating data, so only the two first modes are described in this section. For compiler features see the chapter *Processing data*, for model viewer features chapter *What we get?*)

It’s not necessary for Therion itself—you may edit input files in your favourite text editor and run Therion from the command line. XTherion is also not the only GUI which may be used with Therion. It’s possible to write a better one, which would be more user friendly, more WYSIWYG, faster, more robust and easier to use. Any volunteers?

This manual does not describe such familiar things as ‘if you want to save a file, go to menu File and select Save, or press Ctrl-s’. Browse the top menu for a minute to get feeling of XTherion.

For each mode of operation, there is an additional right or left menu. The submenus may be packed; you may unpack them by clicking on the menu button. For most of the menus and buttons, there is a short description in the status line, so it’s not hard to guess

the meaning of each one. The order of submenus on the side may be customized by the user. Right-click on the menu button and select in the menu which of the other menus it should be swapped with.

XTherion—text editor

XTherion's text editor offers some interesting features which may help with creating text input files: support for Unicode encoding and ability to open multiple files.

To make entering data easy, it supports table formatting of centreline data. There is a menu *Data table* for typing the data. It may be customized to user's data order by pressing a *Scan data format* button when the cursor is below the data order specification ('data' option in the 'centreline' command).

XTherion—map editor

Map editor allows you to draw and edit map fully interactively. But don't expect too much. XTherion is not a truly WYSIWYG editor. It displays only the position, not the actual shape, of drawn point or line symbols. Visually there is no difference between a helictite and a text label—both are rendered as simple dots. The type and other attributes of any object are specified only in the *Point control* and *Line control* menus.

Exercise: Find two substantial reasons, why the map drawn in XTherion can't be identical with Therion output. (If you answer this, you'll know, why XTherion will never be true WYSIWYG editor. Authors' laziness is not the correct answer.)

Let's begin by describing typical use of the map editor. First, you have to decide which part of the cave (which scrap) you'll draw. (It's possible to draw more than one scrap in one file, in which case all inactive scraps are rendered yellow.)



After creating a new file in the map editor, you may load one or more **images**—scanned survey sketches from the cave—as a background for the drawing. Click on the *Insert* button in *Background images* menu. Unfortunately, as a limitation of Tcl/Tk language, only GIF, PNM and PPM (plus PNG and JPEG if you installed tkImg extension) images are supported. All opened images are placed in the upper-left corner of the working area. Move them by double clicking and dragging with the right mouse button or through a menu. For better performance on slower computers, it's possible to temporarily unload a currently unused image from memory by unchecking its *Visibility* check-box. It's possible to open an existing file without loading of background images using *Open XP* menu. (*Note:* Therion doesn't use background images in any way.)

The size and zoom setting of the **drawing area** is adjusted in the corresponding menu. *Auto adjust* calculates optimal size of the working area according to the sizes and positions of loaded background images.

After these preparation steps, you're ready for drawing, or, more precisely, for **creating a map data file**. It's important to remember, that you're actually creating a text file which should conform to the syntax described in the chapter *Data format*. Actually, only a subset of the Therion commands are used in the Map editor: multi-line **scrap** ... **endscrap** commands which may contain **point**, **line** and **area** commands. (Cf. chapter *Data format*). This corresponds with a section of hand-drawn map, which is built up from points, lines and filled areas.

So, the first step is defining the **scrap** by a **scrap** ... **endscrap** multi-line command. In the *File commands* menu click on the *Action* submenu and select *Insert scrap*. This changes the *Action button* to *Insert scrap* if it had any other value. After pressing this button a new scrap will be inserted. You should see lines

```
scrap - scrap1
endscrap
end of file
```

in the preview window above the *Insert scrap* button. This window is a simplified outline of the text file, which will be saved by XTherion. Only the command (**scrap**, **point**, **line**, **text**—why text see below) and its type (for **point** and **line**) or ID (for **scrap**) are shown.

The full contents of any command is displayed in the *Command preview* menu.

For modifying previously-created commands, there are additional menus—e.g. *Scrap control* for the **scrap** command. Here you can change the ID (very important!) and other options. For details see chapter *Data format*.

Now it's possible to insert some **point symbols**. As with scrap insertion, go to the *File commands* menu, click on the *Action* submenu and select *Insert point*; then press newly renamed *Insert point* button. A shortcut for all this is Ctrl-p. Then click on the desired spot in the working area and you'll see a blue dot representing a point symbol. Its attributes can be adjusted in the *Point control* menu. You'll stay in 'insert' mode—each click on the working area adds a new point symbol. Take care not to click twice on the same place—you would insert two point symbols in the same place! To escape from 'insert' to 'select' mode, press *Esc* key on the keyboard or *Select* button in the *File commands* menu.

What order will the commands be in the output file? Exactly as in the outline in the *File commands* menu. Newly created point, line and text objects are added before the currently marked line in the outline. It's possible to change the order by selecting a line and pressing *Move down*, *Move up* or *Move to* buttons in the *File commands* menu.

Drawing lines is similar to drawing in other vector editing programs, which work with Bézier curves. (Guess how to enter the line insertion mode, other than using the shortcut Ctrl-l.) Click where the first point should be, then drag the mouse with pressed left button and release it where the first control point should be. Then click somewhere else (this point will be the second point of the curve) and drag the mouse (adjusting the second

control point of the previous arc and the first control point of the next one simultaneously.) If this explanation sounds too obscure, you can get some practise working in some of the standard vector editors with comprehensive documentation. The line will be finished after escaping from the insertion mode. Beginning and orientation of the line is marked by a small orange tick to the left at the first point.

For line symbols, there are two control menus: *Line control* and *Line point control*. First one sets attributes for the whole curve, like type or name. The check-box *reverse* is important: Therion requires oriented curves and it is not unusual that you begin to draw from the wrong end. The *Line point control* menu enables you to adjust the attributes of any selected point on the line, such as the curve being smooth at this point (which is on by default), or the presence of neighbouring control points ('<<' and '>>' check-boxes).

Areas are specified by their surrounding lines. Click on *Insert area* and then click on the lines surrounding the desired area. They are automatically inserted in the *Area control* and named (if not already named). An alternate way is to insert them as a **text** command which contents (entered in the *Text editor* menu of the Map editor) is usual **area ... endarea** multi-line command (see the chapter *Data format*.)

CAUTION! The command **text** is not a Therion command! It's only a nickname for a block of an arbitrary text in XTherion. In the file saved by XTherion, there'll only be whatever you type into the *Text editor* or see in the *Command preview*. It may be an area definition or whatever you want, such as a comment beginning with a '#' character.

If you draw some scraps with **none** projection, it's necessary to **calibrate** the drawing area. The scale can be defined only one way in XTherion—using coordinates of two points. After selecting a scrap (click on its header in the *File commands* menu) two small red squares will appear (by default, they'll be in the lower corners of drawing area). You have to drag them to points with known coordinates—usually intersections of mm grid lines on the scanned drawing. If you can not see these points, you can move pointer to desired position, read pointer coordinates from the status bar and enter these coordinates into *picture scale points* boxes in the *Scrap control*. You have to fill X1,Y1 and X2,Y2 coordinate pairs. Then you have to enter real coordinates of these points.

In the **selection mode** you can select existing line or point objects and set their attributes in the corresponding menus, move them, or delete them (Ctrl-d or *Action button* in *File commands menu* after setting *Action* to *Delete*).

There is a *Search and select* menu, which makes it easy to switch between objects and visualize things, you can't see at the first look at the picture. For example, if you enter expression 'station' and press *Show All*, all stations on the picture will become red.

XTherion doesn't do any syntax checking; it only writes drawn objects with their attributes to a text file. Any errors are detected only when you process these files with Therion.

TIP: Entering symbols of the same type at once saves you a lot of time because you need not change symbol type and fill options for each new symbol. (*Options* box preserves

the old value and it's enough to change a few characters (e.g. if you're entering stations, `-name 34@pajan` option inherited from previous station may be easily changed to `-name 35@pajan` by retyping one character). It is a good idea to start with drawing all survey stations (don't forget to give them names according to real names in the centreline command), then all passage walls followed by all other point symbols, lines and areas. Finally draw cross-sections.

Keyboard and mouse shortcuts in the Map editor

General

- Ctrl+Z ▷ undo
- Ctrl+Y ▷ redo
- F9 ▷ compile current project
- to select object in the listbox using keyboard: switch using 'Tab' into desired listbox; move with underlined cursor to desired object; press 'Space'

Drawing area and background images

- RightClick ▷ scroll drawing area
- Double RightClick on the image ▷ move the image

Inserting line

- Ctrl+L ▷ insert new line and enter an 'insert line point' mode
- LeftClick ▷ insert line point (without control points)
- Ctrl+LeftClick ▷ insert line point very close to existing point (normally it's inserted right above closest existing point)
- LeftClick + drag ▷ insert line point (with control points)
- hold Ctrl while dragging ▷ fix the distance of previous control point
- LeftClick + drag on the control point ▷ move its position
- RightClick on one of the previous points ▷ selects the previous point while in insert mode (useful if you want to change also the direction of previous control point)
- Esc or LeftClick on the last point ▷ end the line insertion
- LeftClick on the first line point ▷ close the line and end line insertion

Editing line

- LeftClick + drag ▷ move line point
- Ctrl+LeftClick + drag ▷ move line point close to the existing point (normally it is moved right above closest existing point)
- LeftClick on control point + drag ▷ move control point

Adding line point

- select the point before which you want to insert points; insert required points; press Esc or left-click on the point you selected at the beginning

Deleting line point

- select the point you want to delete; press *Edit line* → *Delete point* in the *Line control* panel

Splitting line

- select the point at which you want to split the line; press *Edit line* → *Split line* in the *Line control* panel

Inserting point

- Ctrl+P ▷ switch to ‘insert point’ mode
- LeftClick ▷ insert point at given position
- Ctrl+LeftClick ▷ insert point very close to existing point (normally it will be inserted right above the closest point)
- Esc ▷ escape from the ‘inset point’ mode

Editing point

- LeftClick + drag ▷ move point
- Ctrl+LeftClick + drag ▷ move point close to the existing point (normally it is moved right above closest existing point)
- LeftClick + drag on point arrows ▷ change point orientation or sizes (according to given switches in Point cotrol panel)

Inserting area

- press *File commands* → *Insert area* to switch to the ‘insert area border’ mode
- RightClick on the lines, that surround desired area
- Esc to finish area border lines insertion

Editing area

- select area you want to edit
- pres ‘Insert’ in the *Area control* to insert other border lines at current cursor position
- pres ‘Insert ID’ to insert border with given ID at current cursor position
- pres ‘Delete’ to remove selected area border line

Selecting an existing object

- LeftClick ▷ select object on the top
- RightClick ▷ select object right below the top object (useful when several points lie above each other)

Thinking in Therion

Although everything about Therion input files has been explained, this chapter offers some additional tips and hints.

How to enter centreline?

If the cave is larger than a few meters it's a good idea to split data in more files and separate centreline data from map data.

We usually use one `*.th` file containing centreline per survey trip. It's handy to start with an empty template file as shown below, where dots will be replaced with appropriate texts.

```
encoding IS08859-1
survey ... -title "..."
  centreline
    team "..."
    team "..."
    date ...
    units clino compass grad
    data normal from to compass clino length
    ... ..
  endcentreline
endsurvey
```

It's useful when the survey has the same name as the file which contains it. (E.g. `survey entrance` in the file `entrance.th`.)

For really large caves it's possible to build a hierarchical structure of directories. In such a case we create one special file called `INDEX.th` which includes all other `*.th` files from given directory and contains `equate` commands to define connections between surveys.

How to draw maps?

The most important thing is to devise division of the cave into scraps. It's almost always a *bad* idea to try to fit each scrap to corresponding `*.th` file with centreline from one survey trip. The reason is that connections between scraps should be as simple as possible. Scraps in general are independent on centreline hierarchy so try to forget the survey hierarchy when drawing maps and choose best scrap joins.

We usually insert maps in the last-but-one level in survey hierarchy. Each scrap may then contain arbitrary part of any survey in the last level of hierarchy. For example, there is a survey `main` which contains surveys `a`, `b`, `c` and `d`. Surveys `a` – `d` contain centreline data

from four survey trips and each of them is in a separate file. There is a map `main_map` which contains scraps `s1` and `s2`. If the `main_map` is located in the `main` survey, scrap `s1` may cover part of the centreline from survey `a`, complete survey `b` and part of `c`; `s2` will cover part of the `a` and `c` surveys and a complete `d` survey. The survey stations names will be referenced using `@` symbol (e.g. `1@a`) in the scraps.

Scraps are usually stored in `*.th2` files. Each file may contain more scraps. To keep data well organized, we have some naming conventions: in the file `foo.th2` all scraps are named `foo_si`, where `i` is 1, 2 and so on. Cross-sections are named `foo_ci`, lines `foo_li` etc. This helps a lot with large cave systems: if some scrap is referenced, you immediately know in which file it had been defined.

Similar to `*.th` files, there may be one file `INDEX.th2` per directory which includes all `*.th2` files, defines scrap joins and maps.

How to create models?

The model is created from scrap outlines. The height and depth of the passage are computed from `passage-height` point map symbols. In the future, there will be `point dimensions` map symbol which will allow more precise floor and ceiling specification.

Processing data

Besides data files, which contain survey data, Therion uses a configuration file, which contains instructions on how the data should be presented.

Configuration file

The configuration filename can be given as an argument to `therion`. By default Therion searches for file named `thconfig` in the current working directory. It is read like any other therion file (i.e. one command per line; empty lines or lines starting with `#` are ignored; lines ended with a backslash continue on the next line.) A list of currently supported commands follow.

'encoding'

Works like the `encoding` command in data files—specifies character sets.

'input'

Works like `input` command in data files—includes other files.

'source'

Description: Specifies which source (data) files Therion should read. You can specify several files here; one per line. You can also specify them using the `-s` command line option (see below).

Syntax: `source <file-name>`

Arguments:

- `<file-name>`

'select'

Description: selects objects (surveys and maps) for export. By default, all survey objects are selected. If there is no map selected, all maps belonging to selected surveys are selected by default for map export.

Syntax: `select <object> [OPTIONS]`

Arguments:

- **<object>** ▷ any survey or map, identified by its ID.

Options:

- **recursive <on/off>** ▷ valid only when a survey is selected. If set on (by default) all subsurveys of the given survey are recursively selected/unselected.
- **map-level <number>** ▷ valid only when a map is selected. Determines the level at which map expansion for atlas export is stopped. By default 0 is used; if “basic” is specified, expansion is done up to the basic maps. *Note:* Map previews are displayed only as specified in maps in the current **map-level**.
- **chapter-level <number>** ▷ valid only when a map is selected. Determines the level at which chapter expansion for atlas export is stopped. By default 0 is used, if “-” or “.” is specified, no chapter is exported for this map. If **title-pages** option in **layout** is on, each chapter starts with a title page.

‘unselect’

Description: Unselects objects from export.

Syntax: **unselect <object> [OPTIONS]**

Arguments:

The same as the **select** command.

Options:

The same as the **select** command.

‘layout’

Description: Specifies layout for 2D maps. Settings which apply to atlas mode are marked ‘A’; map mode ‘M’.

Syntax: **layout <id> [OPTIONS]**

```
copy <source layout id>
scale <picture length> <real length>
base-scale <picture length> <real length>
symbol-set <symbol-set>
symbol-assign <point/line/area/group/special> <symbol-type> \
                                     <symbol-set>
symbol-hide <point/line/area/group/special> <symbol-type>
symbol-show <point/line/area/group/special> <symbol-type>
size <width> <height> <units>
overlap <value> <units>
```

```

page-setup <dimensions> <units>
page-numbers <on/off>
exclude-pages <on/off> <list>
title-pages <on/off>
nav-factor <factor>
nav-size <x-size> <y-size>
transparency <on/off>
opacity <value>
layers <on/off>
grid-origin <x> <y> <x> <units>
grid-size <width> <height> <units>
origin <x> <y> <x> <units>
origin-label <x-label> <y-label>
own-pages <number>
page-grid <on/off>
legend <on/off/all>
map-comment <string>
map-header <x> <y> <off/n/s/e/w/ne/nw/se/sw/center>
statistics <explo/topo/carto/copyright all/off/number>
           <explo/topo-length on/off>
scale-bar <length> <units>
language <xx[_YY]>
colour/color <item> <colour>
debug <on/all/first/second/off>
doc-author <string>
doc-keywords <string>
doc-subject <string>
doc-title <string>
code <metapost/tex-map/tex-atlas>
endlayout

```

Arguments:

<id> ▷ layout identifier (to be used in the `export` command)

Command-like options:

- `copy <source layout id>` ▷ set properties here that are not modified based on the given source layout.

map presentation-related:

- `scale <picture length> <real length>` ▷ set scale of output map or map atlas (M, A; default: 1 200)

- **base-scale** <picture length> <real length> ▷ if set, Therion will optically scale the map by a (scale/base-scale) factor. This has the same effect as if the map printed in base-scale would be photoreduced to the scale. (M, A)

- **symbol-set** <symbol-set> ▷ use symbol-set for all map symbols, if available (M, A)

Therion uses following predefined symbol sets:

UIS (International Union of Speleology)

ASF (Australian Speleological Federation)

CCNP (Carlsbad Caverns National Park)

SKBB (Speleoklub Banská Bystrica)

- **symbol-assign** <point/line/area/group/special> <symbol-type> <symbol-set> ▷ display a particular symbol in the given symbol-set. This option overrides symbol-set option.

If the symbol has a subtype, <symbol-type> argument may have one of the following forms: **type:subtype** or simply **type**, which assigns new symbol set to all subtypes of a given symbol.

Following symbols may not be used with this option: point *section* (which isn't rendered at all) and all point and line labels (*label*, *remark*, *altitude*, *height*, *passage-height*, *station-name*, *date*). See the chapter *Changing layout/Customizing text labels* for details how to change labels' appearance. (M, A)

Group may be one of the following: all, centerline, sections.

There are two special symbols: north-arrow, scale-bar.

- **symbol-hide** <point/line/area/group/special> <symbol-type> ▷ don't display particular symbol or group of symbols. May be combined with **symbol-show**. (M, A)
- **symbol-show** <point/line/area/group/special> <symbol-type> ▷ display particular symbol or group of symbols. May be combined with **symbol-hide**. (M, A)

page layout related:

- **size** <width> <height> <units> ▷ set map size in the atlas mode. In map mode applies iff **page-grid** is on (M, A; default: 18 22.2 cm)
- **overlap** <value> <units> ▷ set overlap size in paper units in the atlas mode or map margin in the map mode (M, A; default: 1 cm)
- **page-setup** <dimensions> <units> ▷ set page dimensions in this order: paper-width, paper-height, page-width, page-height, left-margin and top-margin (A; default: 21 29.7 20 28.7 0.5 0.5 cm)
- **page-numbers** <on/off> ▷ turn on/off page numbering (A; default: true)

- **exclude-pages** <on/off> <list> ▷ exclude specified pages from cave atlas. The list may contain page numbers separated by a comma or dash (for intervals) e.g. 2,4-7,9,23 means, that pages 2, 4, 5, 6, 7, 9 and 23 should be omitted. Only the map pages should be counted. (Set **own-pages** 0 and **title-pages** off to get the correct page numbers to be excluded.) Changes of **own-pages** or **title-pages** options don't affect page excluding. (A)
- **title-pages** <on/off> ▷ turn on/off title pages before each atlas chapter (A; default: off)
- **nav-factor** <factor> ▷ set atlas navigator zoom factor (A; default: 30)
- **nav-size** <x-size> <y-size> ▷ set number of atlas pages in both directions of navigator (A; default: 2 2)
- **transparency** <on/off> ▷ set transparency for the passages (underlying passages are also visible) (M, A; default: on)
- **opacity** <value> ▷ set opacity value (used if **transparency** is on). Value range is 0–100. (M, A; default: 70)
- **layers** <on/off> ▷ enable/disable PDF 1.5 layers (M, A; default: on)
- **grid-origin** <x> <y> <x> <units> ▷ set coordinates of grid origin (M, A) [Grid is currently not supported.]
- **grid-size** <width> <height> <units> ▷ set grid size in real units (M, A; default 10 m)
- **origin** <x> <y> <x> <units> ▷ set origin of atlas pages (M, A)
- **origin-label** <x-label> <y-label> ▷ set label for atlas page which has the lower left corner at the given origin coordinates (M, A; default: 0 0)
- **own-pages** <number> ▷ set number of own pages added before the first page of automatically generated pages in atlas mode (currently required for correct page numbering) (A; default: 0)
- **page-grid** <on/off> ▷ show pages key plan (M; default: off)

map legend related:

- **map-header** <x> <y> <off/n/s/e/w/ne/nw/se/sw/center> ▷ print map header at location specified by <x> <y>. Predefined map header contains some basic informations about cave: name, scale, north arrow, list of surveyors etc. It is fully customizable (see the chapter *Changing layout* for details). Ranges for <x> and <y> are 0–100. Lower-left corner of the map is 0 0, upper-right corner is 100 100. The header is aligned with the specified corner or side to this anchor point. (M; default: 0 100 nw)
- **legend** <on/off/all> ▷ display list of used map symbols in the map header. If set to all, all symbols from the current symbol set are displayed. (M, A; default: off)
- **map-comment** <string> ▷ optional comment displayed at the map header (M)

- `statistics <explo/topo/carto/copyright all/off/number>` or
- `statistics <explo/topo-length on/off>` ▷ display some basic statistics (M, A; default: off)
- `scale-bar <length> <units>` ▷ set the length of the scale-bar (M, A)
- `language <xx[_YY]>` ▷ set output language. Currently supported languages are `sk` (Slovak) and `en` (English). See the *Appendix* if you want to add or customize translations. (M, A)
- `colour/color <item> <colour>` ▷ customize colour for special map items (map-fg, map-bg, preview-above, preview-below). Colour range is 0–100 for grayscale, [0–100 0–100 0–100] triplet for RGB colours.
- `debug <on/all/first/second/off>` ▷ draw scrap in different stages of transformation in different colours to see how Therion distorts map data. See the description of `scrap` command for details. The points with distance changed most during transformation are displayed orange.

PDF related:

- `doc-author <string>` ▷ set document author (M, A)
- `doc-keywords <string>` ▷ set document keywords (M, A)
- `doc-subject <string>` ▷ set document subject (M, A)
- `doc-title <string>` ▷ set document title (M, A)

customization:

- `code <metapost/tex-map/tex-atlas>` ▷ Add/redefine \TeX and **METAPOST** macros here. This allows user to configure various things (like user defined symbols, map and atlas layout at one place &c.) See the chapter *Changing layout* for details.

'export'

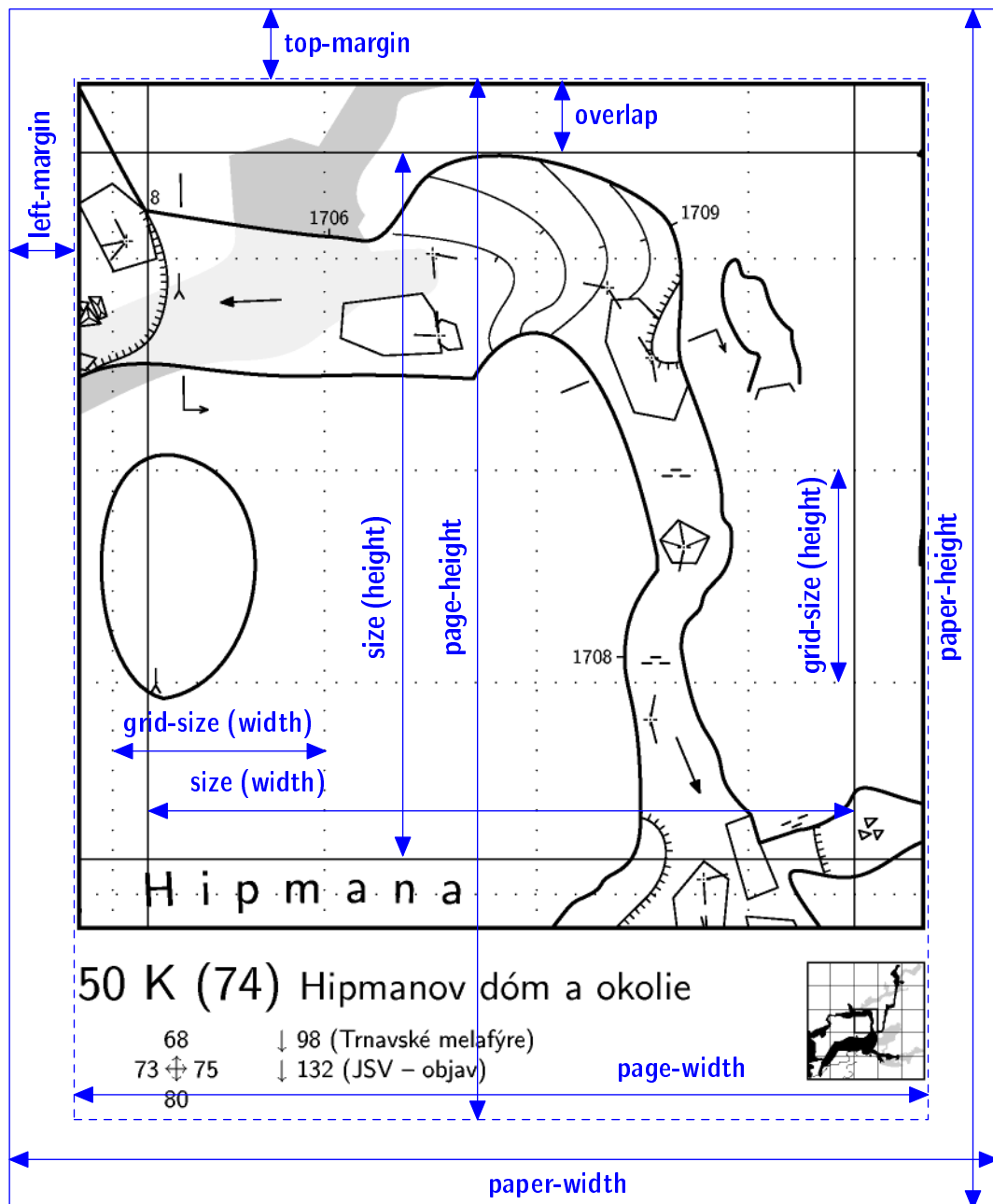
Description: Exports selected surveys or maps.

Syntax:

- `export <type> [OPTIONS]`

Arguments:

- `<type>` ▷ The following export types are supported:
 - `model` ▷ 3D model of the cave
 - `map` ▷ one page 2D map
 - `atlas` ▷ 2D atlas in more pages



database ▷ SQL database with centreline

Options:

common:

- **output/o** <file> ▷ set output file name. If no file name is given the prefix “cave.” is used with an extension corresponding to output format.

model:

- **format/fmt** <format> ▷ set model output format. Currently the following output formats are supported: **therion** (native format; default), **compass** (plt file), **survex** (3d file), **vrml** and **3dmf** (walls only).

map/atlas:

- **projection** <id> ▷ unique identifier that specifies the map projection type. (See the **scrap** command for details.) If there are no scraps with the specified projection then Therion will give an error.
- **layout** <id> ▷ map or atlas layout—see layout command for details.
- **layout-xxx** ▷ where **xxx** stands for other layout options. Using this you can change some layout properties directly within the export command.
- **format/fmt** <format> ▷ set map format. Currently only PDF format is supported.

database:

- **format/fmt** <format> ▷ currently only **sql**
- **encoding/enc** <encoding> ▷ set output encoding

Running Therion

Now, after mastering data and configuration files, we're ready to run Therion. Usually this is done from the command line in the data directory by typing

therion

The full syntax is

```
therion [-q] [-L] [-l <log-file>]
        [-s <source-file>] [-p <search-path>]
        [-g/-u] [-i] [-d] [-x] [--use-extern-lib] [<cfg-file>]
```

or

```
therion [-h/--help]
        [-v/--version]
        [--print-encodings]
        [--print-tex-encodings]
        [--print-init-file]
        [--print-environment]
```

Arguments:

<cfg-file> Therion takes only one optional argument: the name of a configuration file. If no name is specified **thconfig** in the current directory is used.

Options:

- **-d** ▷ Turn on debugging mode. The current implementation creates a temporary directory named **thTMPDIR** (in your system temporary directory) and does not delete any temporary files.

- **-g** ▷ Generate a new configuration file. This will be the given **<cfg-file>** if specified, or **thconfig** in the current directory if not. If the file already exists, it will be overwritten.
- **-h, --help** ▷ Display short help.
- **-i** ▷ Ignore comments when writing **-g** or **-u** configuration file.
- **-L** ▷ Do not create a log-file. Normally therion writes all the messages into a **therion.log** file in the current directory.
- **-l <log-file>** ▷ Change the name of the log file.
- **-p <search-path>** ▷ This option is used to set the search path (or list of colon-separated paths) which therion uses to find its source files (if it doesn't find them in the working directory).
- **-q** ▷ Run therion in quiet mode. It will print only warning and error messages to **STDERR**.
- **--print-encodings** ▷ Print a list of all supported input encodings.
- **--print-tex-encodings** ▷ Print a list of all supported encodings for PDF output.
- **--print-init-file** ▷ Print a default initialization file. For more details see the *Initialization* section in the *Appendix*.
- **-s <source-file>** ▷ Set the name of the source file.
- **-u** ▷ Upgrade the configuration file.
- **--use-extern-libs** ▷ Don't copy **T_EX** and **METAPOST** macros to working directory. **T_EX** and **METAPOST** should search for them on their own. Use with caution.
- **-v, --version** ▷ Display version information.
- **-x** ▷ Generate file **'xth-thconfig'** with additional informations for XTherion.

XTherion—compiler

XTherion makes it easier to run Therion especially on systems without a command line prompt. Compiler window is the default window of XTherion. To run Therion it's enough to open a configuration file and press 'F9' or 'Compile' button.

After a first run there are activated additional menus *Survey structure* and *Map structure*. User may comfortably select a survey or map for export by double clicking on some of the items in the tree. Simple click in the *Survey structure* tree displays some basic informations about the survey in the *Survey info* menu.

What we get?

Information files

Log file

Besides the messages from Therion and other programs used, the log file contains information about loop errors and scrap distortions.

XTherion

Therion provides some basic facts about each survey (length, vertical range, N–S range, E–W range, number of shots and stations) if `-x` option is given. This information is displayed in XTherion, *Compiler* window, *Survey info* menu, when some survey from the *Survey structure* menu is selected.

SQL export

SQL export makes it easy to get very detailed and subtle informations about centreline. It is a text file starting with tables declaration (where ‘?’ stands in the following listing for a minimal value required by the column data)

```
create table SURVEY (ID integer, PARENT_ID integer,
  NAME varchar(?), FULL_NAME varchar(?), TITLE varchar(?));
create table CENTRELINE (ID integer, SURVEY_ID integer,
  TITLE varchar(?), TOPO_DATE date, EXPLOR_DATE date,
  LENGTH real, SURFACE_LENGTH real, DUPLICATE_LENGTH real);
create table PERSON (ID integer, NAME varchar(?), SURNAME varchar(?));
create table EXPLOR (PERSON_ID integer, CENTRELINE_ID integer);
create table TOPO (PERSON_ID integer, CENTRELINE_ID integer);
create table STATION (ID integer, NAME varchar(?),
  SURVEY_ID integer, X real, Y real, Z real);
create table STATION_FLAG (STATION_ID integer, FLAG char(3));
create table SHOT (ID integer, FROM_ID integer, TO_ID integer,
  CENTRELINE_ID integer, LENGTH real, BEARING real, GRADIENT real,
  ADJ_LENGTH real, ADJ_BEARING real, ADJ_GRADIENT real,
  ERR_LENGTH real, ERR_BEARING real, ERR_GRADIENT real);
create table SHOT_FLAG (SHOT_ID integer, FLAG char(3));
```

which is followed by a mass of SQL insert commands. This file may be loaded into any SQL database (after some database-dependent initialization, which may include running a SQL server and connecting to it, creating a database and connecting to it. A good idea is to start a transaction before loading this file, if database doesn't start a transaction automatically.) It's important to set-up database encoding to match the one specified in Therion `export database` command.

Table and column names are self-explaining; for undefined or non-existing values `NULL` is used. Examples of simple queries follow:

List of survey team members with an information how much has each of them surveyed:

```
select sum(LENGTH), sum(SURFACE_LENGTH), NAME, SURNAME
  from CENTRELINE, TOPO, PERSON
 where CENTRELINE.ID = TOPO.CENTRELINE_ID and PERSON.ID = PERSON_ID
 group by NAME, SURNAME order by 1 desc, 4 asc;
```

Which parts of the cave were surveyed in the year 1998?

```
select TITLE from survey where ID in
  (select SURVEY_ID from CENTRELINE
   where TOPO_DATE between '1998-01-01' and '1998-12-31');
```

How long are passages lying between 1500 and 1550 m a.s.l.?

```
select sum(LENGTH) from SHOT, STATION S1, STATION S2
 where (S1.Z+S2.Z)/2 between 1500 and 1550 and
  SHOT.FROM_ID = S1.ID and SHOT.TO_ID = S2.ID;
```

2D maps

Maps are produced in PDF format, which may be viewed or printed in a wide variety of viewers. Be sure to uncheck *Fit page to paper* or similar option if you want to print in the exact scale.

In atlas mode some additional information is put on each page: page number, map name, and page label.

Especially useful are the numbers of neighbouring pages in N, S, E and W directions, as well as in upper and lower levels. There are also hyperlinks at the border of the map if the cave continues on the next page and on the appropriate cells of the Navigator.

PDF files are highly optimized—scraps are stored in XObject forms only once in the document and then referenced on appropriate pages. Therion uses most advanced PDF features like transparency and layers.

Created PDF files may be optionally post-processed in applications like pdfTEX or Adobe Acrobat—it's possible to extract or change some pages, add comments or encryption, etc.

3D models

Therion may export 3D model in various formats besides its native format. These may be loaded in appropriate viewing, editing or raytracing programs to be printed or further processed. If the format doesn't support arbitrary passage shape definition, only the centreline is included.

XTherion—model viewer

XTherion includes OpenGL model viewer which displays models in the native (**therion**) format. It's tested under Linux and Windows 2000 and XP. If you encounter problems compiling or running it on other systems, you may use any other format exported from Therion.

Changing layout

This chapter is extremely useful if you're not satisfied with the predefined layout of map symbols and maps provided, and want to adapt them to your needs. However, you need to know how to write plain \TeX and **METAPOST** macros to do this.

Page layout in the atlas mode

The `layout` command allows basic page setup in the atlas mode. This is done through its options such as `page-setup` or `overlap`. But there are no options which would specify the position of map, navigator and other elements inside the area defined by `page-width` and `page-height` dimensions; e.g., why is the navigator below the map and not on its right or left side?

There are many possible arrangements for a page. Rather than offer even more options for the `layout` command, Therion uses the \TeX language to describe other page layouts. This approach has the advantage that the user has direct access to the advanced typesetting engine without making the language of Therion overcomplex.

Therion uses pdf \TeX with the *plain* format for typesetting. So you should be familiar with the plain \TeX if you wish to define new layouts.

The ultimate reference for plain \TeX is

Knuth, D. E.: *The \TeX book*, Reading, Massachusetts, Addison-Wesley ¹1984

For pdf \TeX 's extensions there is a short manual

Thành, H. T.—Rahtz, S.—Hagen, H.: *The pdf \TeX user manual*, available at <http://www.pdfTeX.org>

The \TeX macros are used inside of code `tex-atlas` part of the `layout` command (see the chapter *Processing data* for details). The basic one predefined by Therion is the

`\dopage`

macro. The idea is simple: for each page Therion defines \TeX variables (count, token, and box registers) which contain the page elements (map, navigator, page name etc.). At the end of each page macro `\dopage` is invoked. This defines the position of each element on the page. By redefining this macro you'll get desired page layout. Without this redefinition you'll get the standard layout.

Here is the list of variables defined for each page:

Boxes:

- `\mapbox` > The box containing the map. Its width (height) is set according to the `size` and `overlap` options of the `layout` command to

`size_width + 2*overlap` or
`size_height + 2*overlap`, respectively

- `\navbox` ▷ The box containing the navigator, with dimensions
`size_width * (2*nav_size_x+1) / nav_factor` or
`size_height * (2*nav_size_y+1) / nav_factor`, respectively

Both `\mapbox` and `\navbox` also contain hyperlinks.

Count registers:

- `\pointerE`, `\pointerW`, `\pointerN`, `\pointerS` contain the page number of the neighbouring pages in the E, W, N and S directions. If there is no such a page its page number is set to 0.
- `\pagenum` current page number

Token registers:

- `\pointerU`, `\pointerD` contain information about pages above and below the current page. It consists of one or more concatenated records. Each record has a special format
`page-name|page-number|destination||`

If there are no such pages, the value is set to `notdef`.

See the description of the `\processpointeritem` macro below for how to extract and use this information.

- `\pagename` ▷ name of the current map according to options of the `map` command.
- `\pagelabel` ▷ the page label as specified by `origin` and `origin-label` options of the `layout` command.

The following variables are set at the beginning of the document:

- `\hsize`, `\vsize` ▷ \TeX page dimensions, set according to `page-width` and `page-height` parameters of the `page-setup` option of the `layout` command. They determine our playground when defining page layout using the `\dopage` macro.
- `\ifpagenumbering` ▷ This conditional is set true or false according to the `page-numbers` option of the `layout` command.

There are also some predefined macros which help with the processing of `\pointer*` variables:

- `\showpointer` with one of the `\pointerE`, `\pointerW`, `\pointerN` or `\pointerS` as an argument displays the value of the argument. If the value is 0 it doesn't display anything. This is useful because the zero value (no neighbouring page) shouldn't be displayed.
- `\showpointerlist` with one of the `\pointerU` or `\pointerD` as an argument presents the content of this argument. (Which contains `\pointerU` or `\pointerD`, see above.)

For each record it calls the macro `\processpointeritem`, which is responsible for data formatting.

Macro `\showpointerlist` should be used without redefinition in the place where you want to display the content of its argument; for custom data formatting redefine `\processpointeritem` macro.

- `\processpointeritem` has three arguments (page-name, page-number, destination) and visualizes these data. The arguments are delimited as follows

```
\def\processpointeritem#1|#2|#3\endarg{...}
```

An example definition may be

```
\def\processpointeritem#1|#2|#3\endarg{%
  \hbox{\pdfstartlink attr {/Border [0 0 0]}}%
    goto name {#3} #2 (#1)\pdfendlink}%
}
```

(note how to use the *destination* argument), or much simpler (if we don't need hyperlink features):

```
\def\processpointeritem#1|#2|#3\endarg{%
  \hbox{#2 (#1)}%
}
```

For font management there are macros

- `\size[#1]` for size changes, and
- `\rm`, `\it`, `\bf`, `\ss`, `\si` for type face switching.

See below for a list of predefined texts which may be used in the atlas.

Now we're ready to define the `\dopage` macro. You may choose which of the predefined elements to use. A very simple example would be

```
layout my_layout
  scale 1 200
  page-setup 29.7 21 27.7 19 1 1 cm
  size 26.7 18 cm
  overlap 0.5 cm
  code tex-atlas
    \def\dopage{\box\mapbox}
    \insertmaps
endlayout
```

which defines the landscape A4 layout without the navigator nor any texts. There is only a map on the page.

Note the `\insertmaps` macro. Map pages are inserted at its position. This is not done automatically because you may wish to insert some other pages before the first map page.

More advanced is the default definition of the `\dopage` macro:

```

\def\dopage{%
\ vbox{\centerline{\framed{\mapbox}}
\bigskip
\line{%
\ vbox to \ht\ navbox{
\ hbox{\size[20]\the\pagelabel
\ifpagenumbering\space(\the\pagenum)\fi
\space\size[16]\the\pagename}
\ifpagenumbering
\medskip
\ hbox{\qqquad\qqquad
\ vtop{%
\ hbox to 0pt{\hss\showpointer\pointerN\hss}
\ hbox to 0pt{\llap{\showpointer\pointerW\hskip0.7em}%
\raise1pt\hbox to 0pt{\hss$\updownarrow$\hss}%
\raise1pt\hbox to 0pt{\hss$\leftrightharpoonup$\hss}%
\rlap{\hskip0.7em\showpointer\pointerE}}
\ hbox to 0pt{\hss\showpointer\pointerS\hss}
}\qqquad\qqquad
\ vtop{
\def\arr{$\uparrow$}
\showpointerlist\pointerU
\def\arr{$\downarrow$}
\showpointerlist\pointerD
}
}
\fi
\vss
\scalebar
}\hss
\box\ navbox
}
}
}

```

Using other plain T_EX macros or T_EX primitives it's possible to add other features, e.g. a different layout for odd and even pages; headers and footers; or adding a logo to each page.

In addition to map pages contains atlas additional items: title page, basic facts about the cave, legend with used map symbols etc.

Therion automatically generates list of used map symbols and lists of persons who have discovered, surveyed and drawn selected part of the cave. Following token registers may be used (according to user's requirements before or after the `\insertmaps` macro):

- `\explotitle`, `\topotitle`, `\cartotitle` ▷ translated titles
- `\exploteam`, `\topoteam`, `\cartoteam` ▷ participating members (according to `team`, `explo-team` options for `centrelines` and `author` option of `scraps`)
- `\explodate`, `\topodate`, `\cartodate` ▷ corresponding dates
- `\comment` ▷ is set according to `map-comment` option of the `layout` command
- `\copyrights` ▷ is set according to copyright options for surveys and other objects
- `\cavename` ▷ name of the exported map; set according to `-title` option of exported map
- `\cavelength`, `\cavedepth` ▷ approximate length and depth of displayed map
- `\cavelengthtitle`, `\cavedepthtitle` ▷ translated labels

There is a macro `\atlastitlepages` which combines all token registers mentioned above to get simple preformatted atlas introductory pages.

For legend displaying there are

- `\iflegend` ▷ conditional; true iff `legend` option of the `layout` command was set to `on` or `all` values
- `\legendtitle` ▷ token register containing translated legend title
- `\insertlegend` ▷ macro for inserting legend symbols pictures with translated descriptions
- `\begmulti <i>`, `\endmulti` ▷ text between these macros is typeset in `<i>` columns
- `\formattedlegend` ▷ combines all four above commands to get preformatted legend with header and symbols typeset in two (default; adjust the count register `\legend-columns` to get them more or less) columns if `legend` option is set `on`

North arrow and scale bar may be displayed using

- `\ifnortharrow` ▷ conditional; true if map projection is plan and symbol north-arrow is not hidden in `layout`
- `\ifscalebar` ▷ conditional; true if scalebar is not hidden
- `\northarrow` ▷ PDF form with the north arrow
- `\scalebar` ▷ PDF form with the scale bar

Example how to create atlas with lists of surveyors etc. followed by map pages and with legend at the end:

```
code tex-atlas
  \atlastitlepages
  \insertmaps
  \formattedlegend
```

Page layout in the map mode

In the map mode it's possible to use a lot of predefined variables which are described in the previous chapter: `\cavename`, `\comment`, `\copyrights`, `\exploitle`, `\topotitle`, `\cartotitle`, `\exploteam`, `\topoteam`, `\cartoteam`, `\explodate`, `\topodate`, `\cartodate`, `\cavelength`, `\cavedepth`, `\cavelengthtitle`, `\cavedepthtitle`, `\ifnortharrow`, `\ifscalebar`, `\northarrow`, `\scalebar`, `\iflegend`, `\legendtitle`, `\insertlegend`, `\begmulti` *<i>*, `\endmulti`, `\formattedlegend`, `\legendcolumns`.

In order to place them somewhere on the map page, you have to define `\maplayout` macro in the `code tex-map` section of the `layout` command. It should contain one or more `\legendbox` invocations. The `\legendbox` macro has two parameters: specification of the position on the page (N, E, S, W, NE, SE, SW or NW) and the content to be displayed.

A simple example is

```
\def\maplayout{
  \legendbox{NW}{\northarrow}
}
```

which displays north arrow in the upper-left corner of the map sheet.

For user's convenience, there is `\legendcontent` token register. It contains preformatted cave name, north arrow, scale bar, explo/topo/carto teams, comment, copyrights and legend. Width of the above text may be adjusted by `\legendwidth` dimen register. The `\legendcontent` is also used in the default map layout definition:

```
\def\maplayout{\legendbox{NW}{\the\legendcontent}}
```

Customizing text labels

There is a preliminary interface to changing font sizes for labels via the **METAPOST** macro

```
fonts_setup(<tinysize>,<smallsize>,<normalsize>,<largesize>,<hugesize>);
```

which may be used inside of the `code metapost` section of the `layout` command. `<normalsize>` applies to point label, `<smallsize>` applies to remark and all other point labels. Each of them may apply to line label according to its `-size` option.

Example:

```
code metapost
  fonts_setup(6,8,10,14,20);
```

New map symbols

Therion's layout command makes it easy to switch among various predefined map symbol sets. If there is no such symbol or symbol set you want, it's possible to design new map symbols.

However, this requires knowledge of the **METAPOST** language, which is used for map visualization. It's described in

Hobby, J. D.: *A User's Manual for MetaPost*, available at <http://cm.bell-labs.com/cm/cs/cstr/162.ps.gz>

User may also benefit from comprehensive reference to the **METAFONT** language, which is quite similar to **METAPOST**:

Knuth, D. E.: *The METAFONTbook*, Reading, Massachusetts, Addison-Wesley ¹1986

New symbols may be defined in the `code metapost` section of the `layout` command. This makes it easy to add new symbols at the run-time. It's also possible to add symbols permanently by compiling into Therion executable (see the *Appendix* for instructions how to do this).

Each symbol has to have a unique name, which consists of following items:

- one of the letters 'p', 'l', 'a', 's' for point, line, area or special symbols, respectively;
- underscore character;
- symbol type as listed in the chapter *Data format* with all dashes removed;
- if the symbol has a subtype, add underscore character and subtype;
- underscore character;
- symbol set identifier in uppercase

Example: standard name for a point 'water-flow' symbol with a 'permanent' subtype in the 'MY' set is `p_waterflow_permanent_MY`.

Each new symbol has to be registered by a macro call

```
initsymbol("<standard-name>");
```

unless it's compiled into Therion executable.

There are four predefined pens *PenA* (thickest) ... *PenD* (thinnest), which should be used for all drawings. For drawing and filling use `thdraw` and `thfill` commands instead of **METAPOST**'s `draw` and `fill`.

Point symbols

Point symbols are defined as macros using `def ... enddef` commands. Majority of point symbol definitions has four arguments: position (pair), rotation (numeric), scale (numeric) and alignment (pair). Exceptions are *section* which has no visual representation; all *labels*, which require special treatment as described in the previous chapter, and *station* which takes only one argument: position (pair).

All point symbols are drawn in local coordinates with the length unit u . Recommended ranges are $\langle -0.5u, 0.5u \rangle$ in both axes. The symbol should be centered at the coordinates' origin. For the final map, all drawings are transformed as specified in the T transformation variable, so it's necessary to set this variable before drawing.

This is usually done in two steps (assume that four arguments are P, R, S, A):

- set the U pair variable to $\left(\frac{width}{2}, \frac{height}{2}\right)$ of the symbol for correct alignment. The alignment argument A is a pair representing ratios $\left(\frac{shift_x}{U_x}\right)$ and $\left(\frac{shift_y}{U_y}\right)$.
(Hence aligned A means shifted (xpart $A * \text{xpart } U$, ypart $A * \text{ypart } U$.)
- set the T transformation variable

`T:=identity aligned A rotated R scaled S shifted P;`

For drawing and filling use `thdraw` and `thfill` commands instead of METAPOST's `draw` and `fill`. These take automatically care of T transformation.

An example definition may be

```
def p_entrance_UIS (expr P,R,S,A)=
  U:=(.2u,.5u);
  T:=identity aligned A rotated R scaled S shifted P;
  thfill (-.2u,-.5u)--(0,.5u)--(.2u,-.5u)--cycle;
enddef;
initsymbol("p_entrance_UIS");
```

Line symbols

Line symbols differ from point symbols in respect that there is no local coordinate system. Each line symbol gets the *path* in absolute coordinates as the first argument. Therefore it's necessary to set T variable to `identity` before drawing.

Following symbols take additional arguments:

- arrow ▷ numeric: 0 is no arrows, 1 arrow at the end, 2 begin, 3 both ends
- contour ▷ text: list of points which get the tick or one of -1 , -2 or -3 to mark undefined tick, tick in the middle or no tick, respectively

- section ▷ text: list of points which get the orientation arrow or -1 to indicate no arrows
- slope ▷ numeric: 0 no border, 1 border; text: list of (point,direction,length) triplets

Usage example:

```
def l_wall_bedrock_UIS (expr P) =
  T:=identity;
  pickup PenA;
  thdraw P;
enddef;
initsymbol("l_wall_bedrock_UIS");
```

Area symbols

Areas are similar to lines: they take only one argument – *path* in absolute coordinates.

You may fill them in three ways:

- fill an uniform or randomised grid in a temporary picture (having dimensions `bbox path`) with some point symbols; clip it according to `path` and add to the `currentpicture`
- fill `path` with a solid colour
- fill `path` with a predefined pattern using a `withpattern` keyword.

Patterns are defined using the same user interface (without the `patterncolor` macro) as described in the article

Bolek, P.: “**METAPOST** and patterns,” *TUGboat*, 3, XIX (1998), pp. 276–283, available online at <http://www.tug.org/TUGboat/Articles/tb19-3/tb60bolek.pdf>

You may use standard **METAPOST** `draw` and similar macros without setting of *T* variable in pattern definitions.

Example on how to define and use patterns:

```
beginpattern(pattern_water_UIS);
  draw origin--10up withpen pensquare scaled (0.02u);
  patternxstep(.18u);
  patterntransform(identity rotated 45);
endpattern;
```

```
def a_water_UIS (expr p) =
  T:=identity;
  thclean p;
  thfill p withpattern pattern_water_UIS;
enddef;
initsymbol("a_water_UIS");
```

Special symbols

There are currently two special symbols: scale bar and north arrow. Both are experimental and subject to change.

Appendix

Compilation

If you want to compile Therion from source code and run it, you need (first three are required only during compilation):

- GNU C/C++ compiler
- GNU make
- Perl
- Tcl/Tk 8.4.3 and newer (<http://www.tcl.tk>) with *BWidget* widget set (<http://sourceforge.net/projects/tcllib>), *Tom* OpenGL extension (improved version is included in Therion source distribution) and optionally *tkImg* extension (<http://sourceforge.net/projects/tking>).
- T_EX distribution with at least T_EX with Plain format, recent pdfT_EX, and METAPOST (<http://www.tug.org>).

All programs (with the exception of BWidget, Tom and tkImg package) are usually included in Linux, Unix or MacOS X distributions. For Windows consider using MinGW and MSYS (<http://www.mingw.org>). It's a distribution of GNU utilities with GNU make and GCC. (BTW, why not to use precompiled Windows version?)

Quick start

- `unpack source distribution therion-0.3.*.tar.gz`
- `cd therion`
- `make config-macosx` or `make config-win32`, if you use MacOS X or Windows, respectively
- `make`
- `su`
- `make install`

Installing Tom:

- if you use Windows, download a Tcl/Tk source distribution, `make` and `make install` it under MSYS
- `cd therion/thtom/linux` or `cd therion/thtom/win`

- `make`
- copy `Tom0.2` directory (which should contain `pkgIndex.tcl` and one of `libtom.so` or `libtom.dll`) to the `lib` subdirectory of your Tcl/Tk distribution.

Hacker's guide

Make parameters

Therion's *makefile* may take some optional parameters.

- `config-linux`, `config-macosx`, `config-win32` ▷ configure Therion for a specific platform. Linux is a default.
- `config-release`, `config-oxygen`, `config-ozone` ▷ set optimization level for C++ compiler (none, `-O2` and `-O3`)
- `config-debug` ▷ useful before debugging the program
- `install` ▷ install Therion
- `clean` ▷ delete all temporary files

Adding new translations

Therion supports translation of map labels. Suppose you want to add a new language `xx`.

- run `'perl process.pl export xx'` in the `'thlang'` Therion source subdirectory. This creates a file `texts_xx.txt`. This file is UTF-8 encoded.
- edit the `texts_xx.txt` file. Add your translations at lines beginning with `'xx:'`.
- run `make update`
- compile Therion

Adding new encodings

Although UTF-8 Unicode encoding covers all characters which Therion is able to process, it may be inconvenient to use it. In that case it's possible to add support for any 8-bit encoding for text input files. Copy a translation file to the `thchencdata` directory; add its name to 'ifiles' hash in the beginning of the Perl script `generate.pl`; run it and recompile Therion.

The translation file should contain two hexadecimal values of a character (first one in the 8-bit encoding, second one in Unicode) in each line. Possible comments follow the `'#'` character.

Adding new $T_E X$ encodings

It's easy to add new encodings for 2D map output. Copy an appropriate encoding mapping file with an `*.enc` extension to the `texenc/encodings`, run the Perl script `mktextenc.pl` located in the `texenc` directory and compile Therion.

Therion uses the same encoding files as `afm2tfm` program from the `TEX` distribution, which has the same format as an encoding vector in a PostScript font. You may find more details in the chapter *6.3.1.5 Encoding file format* in the documentation to Dvips program.

Generating new T_EX and METAPOST headers

Therion uses `TEX` and `METAPOST` for 2D map visualization and typesetting. Predefined macros are compiled into the Therion executable and are copied to the working directory just before running `METAPOST` and `TEX` (unless the `--use-extern-libs` option is used). Layout command makes it possible to modify some macros in the configuration file at the run-time.

However, it's possible to make permanent changes to the macro files. After modifying the files in the `mpost` and `tex` directories it's necessary to run Perl scripts `genmpost.pl` and `gentex.pl`, which generate C++ header files, and compile Therion executable again.

Environment variables

Therion reads following environment variables:

- `THERION` ▷ [not required] search path for (x)therion.ini file(s)
- `HOME` (`HOMEDRIVE` + `HOMEPATH` on WinXP) ▷ [not required, but usually present on your system] search path for (x)therion.ini file(s)
- `TEMP`, `TMP` ▷ system temporary directory, where Therion stores temporary files (in a directory named `thPID`, where `PID` is a process ID), unless `tmp-path` is specified in the initialization file.

Consult the documentation of your OS how to set them.

Initialization files

Therion's and XTherion's system dependent settings are specified in the file `therion.ini` or `xtherion.ini`, respectively. They are searched for in the following directories:

- on UNIX: `.`, `$THERION`, `$HOME/.therion`, `/etc`, `/usr/etc`, `/usr/local/etc`
- on Windows: `.`, `$THERION`, `$HOME\therion`, `<Therion-installation-directory>`, `C:\WINDOWS`, `C:\WINNT`, `C:\Program Files\Therion`

Therion

If no file is found Therion uses its default settings. If you want to list them, use `--print-init-file` option. The initialization file is read like any other therion file. (Empty lines

or lines starting with ‘#’ are ignored; lines ending with a backslash continue on next line.)
Currently supported initialization commands follow.

- `encoding-default <encoding-name>`

Set the default output encoding (currently unused).

- `encoding-sql <encoding-name>`

Set the default output encoding for SQL export.

- `language <xx[_YY]>`

Default output language (used for maps). Currently supported languages are `sk` (Slovak) and `en` (English).

- `mpost-path <file-path>`

Set the full path to a **METAPOST** executable if Therion can’t find it (“`mpost`” is the default).

- `pdftex-path <file-path>`

Set the full path to a pdf \TeX executable if Therion can’t find it (“`pdfetex`” is the default).

- `source-path <directory>`

Path to data and configuration files. Used mostly for system-wide grades and layout definitions.

- `tmp-path <directory>`

Path where temporary directory should be created.

- `tmp-remove <OS command>`

System command to delete files from the temporary directory.

- `tex-fonts <encoding> <rm> <it> <bf> <ss> <si>`

Set-up fonts used for given encoding. The list of currently supported encodings gives the `--print-tex-encodings` command line option. The same encoding must be used while generating \TeX metrics (*.tfm files) for those fonts (e.g. with the `afm2tfm` program) and this encoding must be explicitly given also in the pdf \TeX ’s map file. The only exception is the base set of Computern Modern fonts, which use ‘raw’ encoding. This encoding doesn’t need to be specified in the pdf \TeX ’s map file.

Encoding has to be followed by five font specifications for regular, italic, bold, sans-serif and sans-serif oblique styles. Default setting is `tex-fonts raw cmr10 cmti10 cmbx10 cmss10 cmssi10`

Example how to use other fonts (e.g. TrueType Palatino in `xl2` (an encoding derived from ISO8859-2) encoding). Run:

```
ttf2afm -e xl2.enc -o palatino.afm palatino.ttf
```

```
afm2tfm palatino.afm -u -v vpalatino -T xl2.enc
```

```
vptovf vpalatino.vpl vpalatino.vf vpalatino.tfm
```

You get files `vpalatino.vf`, `vpalatino.tfm` and `palatino.tfm`. Add the line

```
palatino <xl2.enc <palatino.ttf
```

to the pdfTEX's map file. The same should be done for the italic and bold faces and corresponding sansserif and sansserifoblique fonts. If you're lazy try

```
tex-fonts xl2 palatino palatino palatino palatino palatino
```

(We should use actually virtual font `vpalatino` instead of `palatino`, which contains no kerning or ligatures, but pdfTEX doesn't support `\pdfincludechars` command on virtual fonts. To be improved.)

If you want to add some unsupported encodings, read the chapter *Compilation / Hacker's guide*.

XTherion

Initialization file for XTherion is actually a Tcl script evaluated when XTherion starts. The file is commented; see the comments for details.

Example data

Following simple example illustrates basic usage of Therion commands:

```
encoding utf-8
```

```
survey main -title "Test cave"
```

```
survey first
```

```
  centreline
```

```
    units compass grad
```

```
    data normal from to compass clino length
```

```
              1    2  100    -5    10
```

```
  endcentreline
```

```
endsurvey
```

```
survey second -declination [3 deg]
```

```
  centreline
```

```
    calibrate length 0 0.96
```

```
    data normal from to compass length clino
```

```
              1    2   0     10    +10
```

```
  endcentreline
```

```
endsurvey
```

```
centreline
```

```
    equate 2@first 1@second
```

```
endcentreline
```

```
# scraps are usually in separate *.th2 files
```

```
scrap s1 -author 2004 "Therion team"
```

```
    point 763 746 station -name 2@second
```

```
    point 702 430 station -name 2@first
```

```
    point 352 469 station -name 1@first
```

```
    point 675 585 air-draught -orientation 240 -scale large
```

```
    line wall -close on
```

```
        287 475
```

```
        281 354 687 331 755 367
```

```
        981 486 846 879 683 739
```

```
        476 561 293 611 287 475
```

```
    endline
```

```
endscrap
```

```
map m1 -title "Test map"
```

```
    s1
```

```
endmap
```

```
endsurvey
```

Corrresponding configuration file could be:

```
encoding utf-8
```

```
source test
```

```
layout l1
```

```
    scale 1 100
```

```
    layers off
```

```
endlayout
```

```
select m1@main
```

```
export model -fmt survex
```

```
export map -layout l1
```

If you save data file as ‘test.th’ and configuration file as ‘thconfig’ you may process them with Therion.

History

• 1999

Oct: first concrete ideas

Nov: start of programming (Perl scripts and **METAPOST** macros)

Dec 27: Therion compiles simple map for the first time (32 kB of Perl and **METAPOST** source code). This first release had some interesting features such as *transformation functions*, which allowed user-specification of the input format for survey data.

• 2000

Jan: xthedit (Tcl/Tk), a graphical front-end for Therion

Feb 18: start of (first?) reprogramming (Perl)

Apr 1: first hyperlinked PDF cave map

Aug: experiments with PDF, pdfT_EX and **METAPOST**

• 2001

Nov: start of reimplementation from scratch: Therion (C++ with some Perl scripts inherited from the previous version); interactive 2D map editor ThEdit as a replacement of xthedit (Delphi)

Dec: ThEdit exports simple map for the first time

• 2002

Mar: Therion 0.1 — Therion is able to process survey data (centreline) of Dead Bats Cave. XTherion, text editor designed for Therion (Tcl/Tk).

Jul 27: Therion 0.2 — Therion compiles simple map (consisting of two scraps) for the first time (800 kB of source code)

Aug: XTherion extended to 2D map editor (as a replacement of ThEdit)

Sep: Therion compiles first real and complex map of a cave. XTherion extended to compiler.

• 2003

Mar: First version of The Therion Book finished

Apr: Therion included in Debian GNU/Linux

Jun: all Perl scripts rewritten in C++, Therion is one executable program now (although using Survox and T_EX)

• 2004

Mar: Therion 0.3 — Therion exports 3D model created from 2D maps. Loop closure algorithm included into Therion.

Future

Although Therion is already used for map production, there are a lot of new features to be implemented:

General

- loop closure informations in SQL

2D maps

- map rotation
- map grid
- complete map symbol sets
- area blocks
- SVG support

3D models

- improve passage walls modelling
- more formats for export