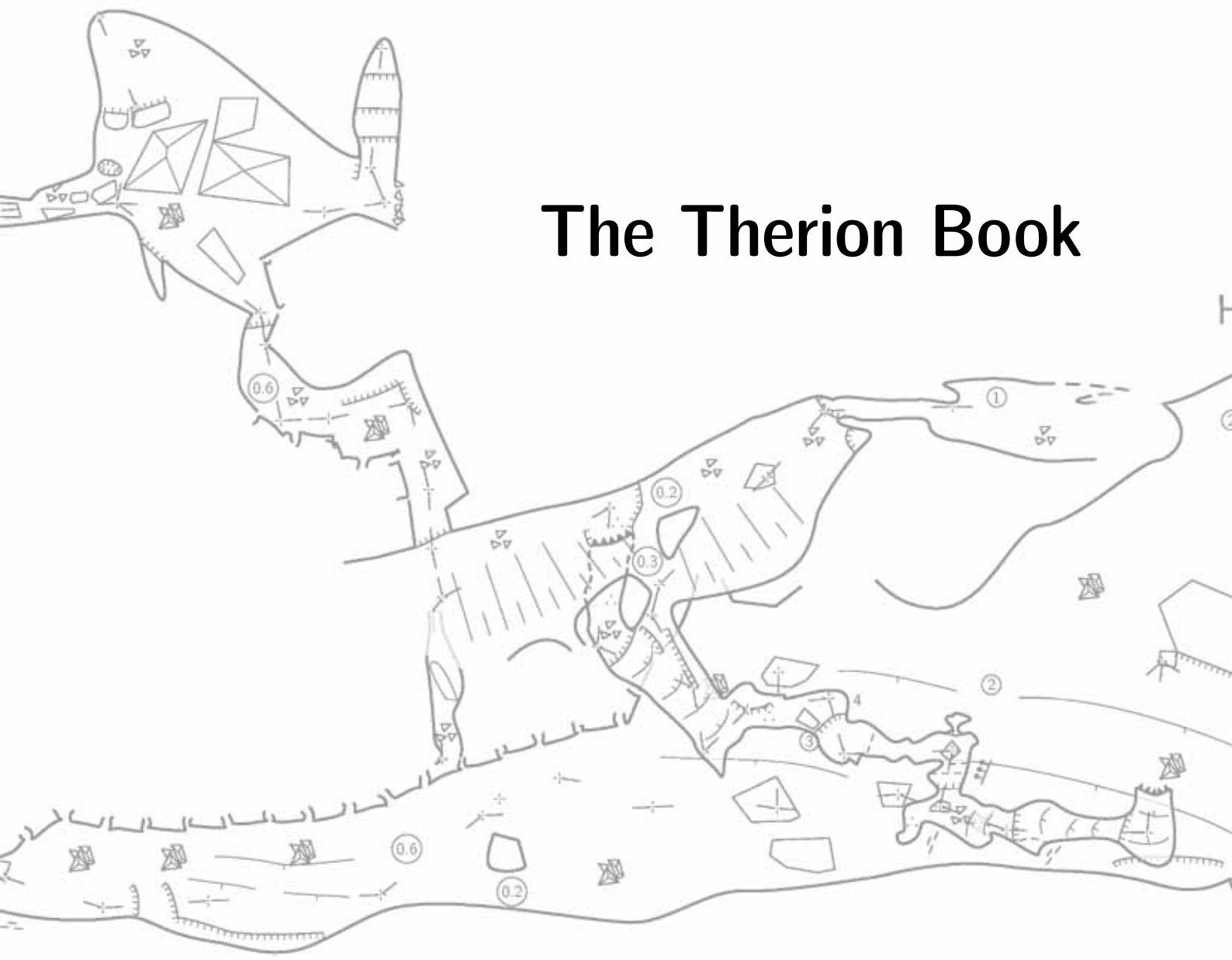
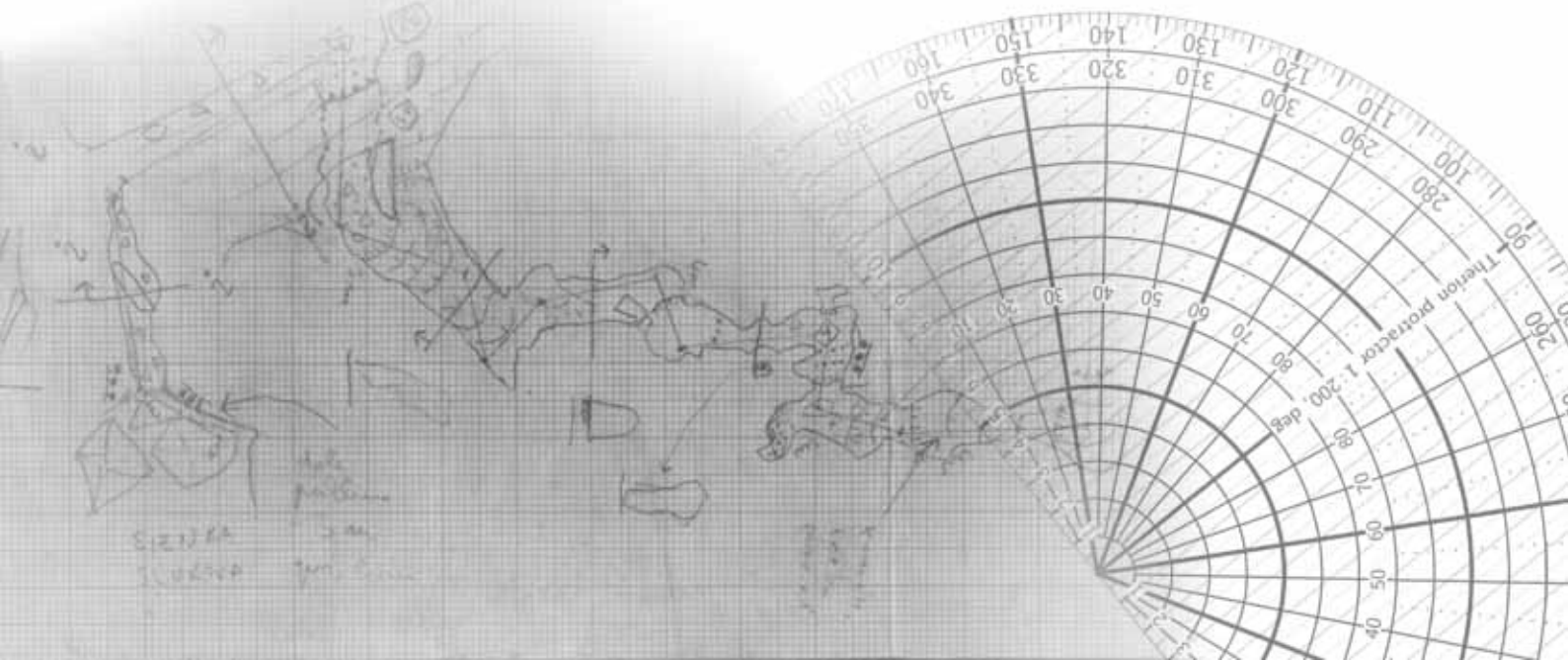
The Therion Book



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 language of this manual.

Contents

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;
- 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 precompiled binaries for Linux and Win32 systems 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, Survex for excellent processing of survey shots, and TEX 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 processess 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), and compiler (which runs Therion on the data).

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 adopted 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 will be generated from 2D plan, elevation and cross-sections data to get a realistic 3D model without entering too much data
- although the support for WYSIWYG is limited, you get what you want (WYWIWYG)

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 cave surveying, 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:

- recent Survex (necessary if you want to process any survey data) `http://www.survex.com`
- TEX distribution with at least TEX with Plain format, recent pdf-ε-TEX, and METAPOST with accompanying programs. (TEX and friends is necessary only if you want to create 2D maps.) Plain should be the default format for TEX; Therion doesn't use LaTEX or other available formats. If you have any doubts, install any full distribution of TEX. (TeXlive for any platform, teTeX for Unix, fpTeX or MikTeX for Windows.) If you're new to TEX, visit the homepage of *TEX Users Group* at `http://www.tug.org`
- Tcl/Tk 8.4.3 and newer (`http://www.tcl.tk`) with BWidget widget set (`http://sourceforge.net/projects/tcllib`). Tcl/Tk is only required for XTherion. For Windows try ActiveTcl from `http://www.activestate.com`.

Get all this working before installing Therion. We don't provide any support concerning this. All these programs are well documented and accessible on the net.

Installation

Primary Therion distribution is the source code. This needs be compiled according to instructions in the *Appendix*.

For users' convenience there are also precompiled binaries for Linux and Windows.

Installation on Linux (*.tar.gz package):

Unpack the archive and run the installation script 'install.'

Installation on Windows:

Run the setup program and follow instructions.

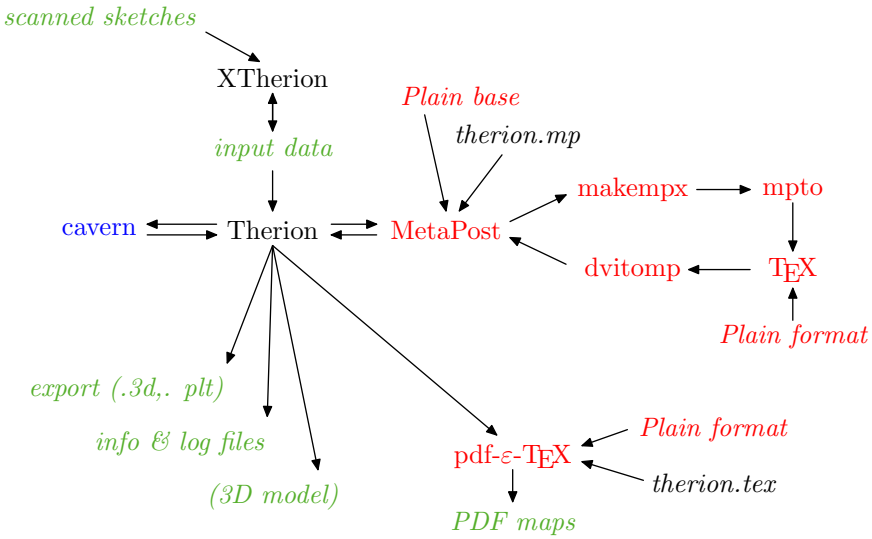
Setting-up environment

Therion reads settigs from the initialization file. Default settings should work fine for users using only ASCII (non-accented latin) characters, standard TEX 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)
- blue—Survex program
- 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, exports survey data to Survex, which does the loop closure. Then it reads back coordinates of survey stations and transforms all other data (e.g. 2D maps) according to them. 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 pdfTEX. PdfTEX does all the typesetting and creates

a PDF file of the cave map. Therion may also export simple 3D model (survey shots only) in Survex *.3d and Compass *.plt formats.

For displaying the results you need:

- Aven or xcaverot from Survex package for viewing the simple 3D model
- any PDF viewer like Acrobat Reader, ghostscript or xpdf for displaying 2D maps

[Complex 3D in the future]

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, Survex, METAPOST and TEX.
- 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 backlash.
- 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:

- *boolean* ▷ logical value (`on`/`off`)
- *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`
- *person* ▷ a person's first name and surname separated by whitespace characters. Use '`/`' where there are more names.
- *string* ▷ a sequence of any charaters.
- *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.] 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 commmand 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.

Example: a short illustration of this command.

'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.

Example: `encoding ISO8859-2`

'input'

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

Syntax: `input <file-name>`

Context: all

Arguments:

- `<file-name>`

Example: `input data/entrance.th2`

'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 explicitely set the declination for undated survey data, use '`-`' instead of date.
- `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

Example:

```
survey main -declination [3 deg]

  survey a
    centreline
      data normal from to compass clino length
      1 2 100 -10 10
    endcentreline
  endsurvey
```

```
  survey b
    centreline
      data normal from to compass length clino
      1 2 0 10 10
    endcentreline
  endsurvey

  centreline
    equate 2@a 1@b
  endcentreline

endsurvey
```

'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>]
          discovery-date <date>
          discovery-team <person>
          instrument <quantity list> <description>
          calibrate <quantity list> <zero error> [<scale>]
          units <quantity list> [<factor>] <units>
          declination <value> <units>
          infer <what> <on/off>
          sd <quantity list> <value> <units>
          grade <grade list>
          data <style> <readings order>
          mark <type>
          flags <shot flags>
          station <station> <comment> [<flags>]
          fix <station> [<x> <y> <z> [<std x> <std y> <std z> \
            [<covxy> <covyz> <covxz>]]]
          equate <station list>
          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.
- `discovery-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).
- `discovery-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

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

  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: $measured\ value = (read\ value - zero\ error) \times scale$. The supported quantities are the same as sd.
- `data <style> <readings order>` ▷ set data style (normal, topofil, diving, cartesian, cylpolar, 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 Survex manual for details.
- `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> [<covxy> <covyz> \ <covxz>]]]` ▷ fix station coordinates (with errors and covariances—only the units transformation, not calibration, is applied to them).
- `equate <station list>` ▷ set points that are equivalent

'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.)

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. No index is allowed for this projection.

  2. `plan` ▷ basic plan projection (default).

  3. `elevation` ▷ this projection optionally takes a view direction as an argument (e.g. `[elevation 10]` or `[elevation 10 deg]`).

  4. `extended` ▷ extended elevation.

- `scale <specification>` ▷ specifies the drawing units scale. 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;

  equipement: 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. $0 \leq$ `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/none/top>` ▷ where to place the symbol relatively to other objects (none by default).

- `clip <bool>` ▷ 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
- `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

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 followig 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:

  `<br>` ▷ line break

  `<center>`, `<left>`, `<right>` ▷ line alignment for multi-line labels. Ignored if there is no `<br>` tag.

- `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. `-altitude [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. 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.

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.

'line'

Description: Line is a command for drawing a line symbol on the map.

Syntax:
```
line <type> [OPTIONS]
    altitude <value>
    border <on/off>
    clip <bool>
    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/none/top>
    reverse <bool>
    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: arrow, border, chimney, contour, flowstone, label, overhang, pit, rock-border, rock-edge, section, slope, survey and wall.

Command-like 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 absolue 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.
- `close <on/off/auto>` ▷ determines whether a line is closed or not
- `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. Center is default.
- `head <begin/end/both/none>` ▷ can be used only with the arrow type and indicates where to put an arrow head. End is default.
- `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. $0 \leq$ `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 <bool>` ▷ 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
- `smooth <on/off/auto>` ▷ whether the line is smooth at the given point. Auto is default.
- `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, invisible, temporary

- `text <string>` ▷ valid only for label lines.
- `[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.
- `place <bottom/none/top>` ▷ where to place the symbol relative to other objects (none by default).
- `clip <bool>` ▷ 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
- `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

Special notes: The following line type has this 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.

'area'

Description: Area is a command for drawing an area on the map.

Syntax:
```
area <type>
  place <bottom/none/top>
  clip <bool>
... 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, which must be in order and each pair of consecutive lines must intersect (i.e. have a point in common).
- `place <bottom/none/top>` ▷ where to place the symbol relative to other objects (none by default).

- `clip <bool>` ▷ 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
- `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

'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:

- `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

'join'

Description: Joins two or more points in a map together

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

Context: scrap, survey

Arguments:

- `<pointX>` can be an ID of a point symbol or an ID of a line, optionally followed by a line point mark `[<id>:<mark>]`. `<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 <bool>` 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> [OPTIONS]`

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

Context: all

Arguments:

Options:

'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 an GUI (Graphical User Interface) for Therion. It helps a lot with creating input data files. Currently it works in three main modes: text editor, map editor and compiler. (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*.)

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

Although the benefits of the special text editor for Therion are disputable, without a map editor it is really too hard to draw any map. Most of the drawing commands (point, line) require specification of position in Cartesian coordinates (x-axis to the right, y-axis up). Which is better: to measure coordinates on the paper and type them by hand, or to display a scanned map on the background of the XTherion's working area and draw the map accordingly as in other vector graphics editors?

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 helectite 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 backround for the drawing. Click on the *Insert* button in *Background images* menu. Unfortunatelly, as a limitation of Tcl/Tk language, only GIF, PNM and PPM images are supported. All opened images are placed in the upper-left corner of the working area. Move it 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. (*Note:* Therion doesn't use these 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* above from *Insert text* to *Insert scrap*. 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 name (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 name 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*; than press newly renamed *Insert point* button. A shortcut for all this is Ctrl-p. Than 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. 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* or *Move up* 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. Than 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.

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 don't have any visual representation. They are inserted 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.

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*).

[adding or deleting a point on the curve]

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.

Keyboard and mouse shortcuts in the Map editor

Drawing area and background images

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

Inserting line

- Crtl+L ▷ insert new line and enter an 'insert line point' mode
- LeftClick ▷ insert line point (without control points)
- Shift+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 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
- Ctrl+Z ▷ undo
- Ctrl+Y ▷ redo

Editing line

- LeftClick + drag ▷ move line point
- Shift + 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 begining

Deleting line point

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

Inserting point

- Ctrl+P ▷ switch to 'insert point' mode
- LeftClick ▷ insert point at given position
- Shift+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
- Shift + 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)

Selecting an existing object

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

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 for export. By default, all survey objects are selected.

Syntax: `select <object> [OPTIONS]`

Arguments:

- `<object>` ▷ any object from the database, such as a 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.

'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]
        code <metapost/tex-map/tex-atlas>
        copy <source layout id>
        doc-author <string>
        doc-keywords <string>
        doc-subject <string>
        doc-title <string>
        exclude-pages <on/off> <list>
        grid-origin <x> <y> <x> <units>
        grid-size <width> <height> <units>
        layers <on/off>
        nav-factor <factor>
        nav-size <x-size> <y-size>
        opacity <on/off> <value>
        origin <x> <y> <x> <units>
```

```
    origin-label <x-label> <y-label>
    overlap <value> <units>
    own-pages <number>
    page-grid <on/off>
    page-numbers <on/off>
    page-setup <dimensions> <units>
    scale <picture length> <real length>
    base-scale <picture length> <real length>
    size <width> <height> <units>
    symbol-set <symbol-set>
    symbol-assign <point/line/area> <symbol-type> <symbol-set>
    symbol-hide <point/line/area> <symbol-type>
    title-pages <on/off>
  endlayout
```

Arguments:

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

Options:

Command-like options:

- `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.
- `copy <source layout id>` ▷ set properties here that are not modified based on the given source layout.
- `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)
- `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 omited. 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)
- `grid-origin <x> <y> <x> <units>` ▷ set coordinates of grid origin (M, A)
- `grid-size <width> <height> <units>` ▷ set grid size in real units (M, A; default `10 m`)
- `layers <on/off>` ▷ enable/disable PDF 1.5 layers (M, A; default: `on`)

- `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`)
- `opacity <value>` ▷ set opacity value (used if `transparency` is `on`). Value range is 0–100. (M, A; default: `70`)
- `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`)
- `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`)
- `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`)
- `page-numbers <on/off>` ▷ turn on/off page numbering (A; default: `true`)
- `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`)
- `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)
- `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`)
- `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)

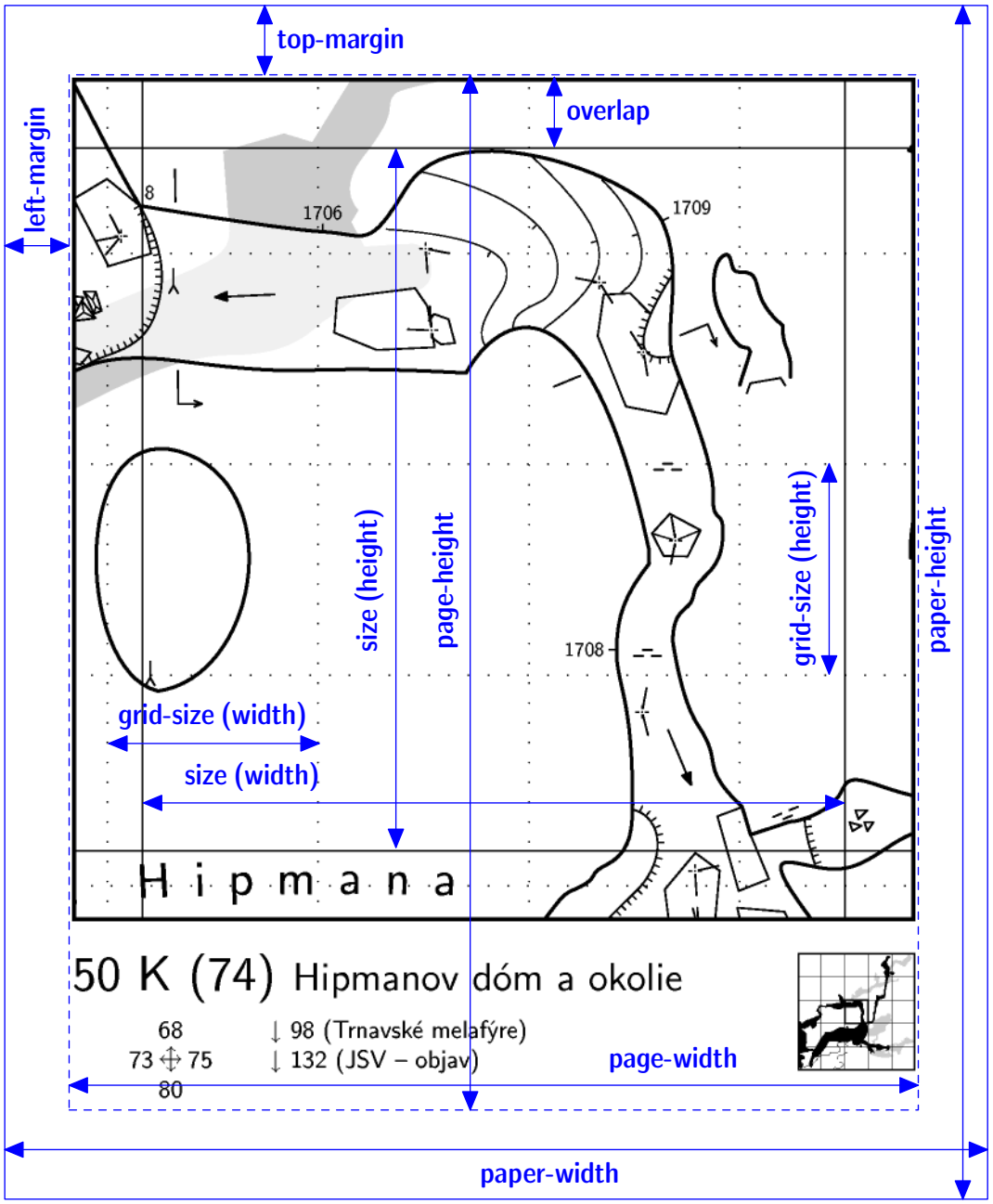
  SKBB (Speleoklub Banská Bystrica)

  JSSJ (map symbols from the book *Jaskynný systém Stratenskej jaskyne*)
- `symbol-assign <point/line/area> <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 has form `type:subtype`.

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)

- `symbol-hide <point/line/area> <symbol-type>` ▷ don't display particular symbol (M, A)
- `title-pages <on/off>` ▷ turn on/off atlas title pages (A; default: `off`)
- `transparency <on/off>` ▷ set transparency for the passages (underlying passages are also visible) (M, A; default: `on`)



'export'

Description: Exports specified data from database.

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

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), compass (plt file) and survex (3d file).

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.
- `title` ▷ title to go on the output survey

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 as follows

```
therion
```

The full syntax is

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

therion [-h/--help]
        [-v/--version]
        [--print-encodings]
        [--print-tex-encodings]
        [--print-init-file]
        [--use-extern-libs]
```

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 TEX and METAPOST macros to working directory. TEX 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

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 dispayed in XTherion, *Compiler* window, *Survey info* menu, when some survey from the *Survey structure* menu is selected.

More advanced statistics should be implemented in the near future.

2D maps

Maps are produced in PDF format, which may be viewed or printed in a wide variety of viewers.

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 than 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 exports centreline 3D models in Survex and Compass formats, which may be viewed in these programs. (Aven, xcaverot, or caverot for *.3d Survex files.) You may also use `print*` programs from the Survex distribution to print the 3D model.

In the future there should be full passage modelling in Therion. In order to display these models, XTherion will be extended to include an OpenGL 3D viewer.

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 pdfTEX 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 TEXbook*, Reading, Massachusetts, Addison-Wesley [1]1984

For pdfTEX's extensions there is a short manual

Thành, H. T.—Rahtz, S.—Hagen, H.: *The pdfTEX user manual*, available at `http://www.pdftex.org`

The TEX macros are used inside of `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 formating.

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

If you use own texts in TEX macros (like a label for the scale bar), these font switches work only in restricted manner: it's supposed that all characters are present in the first encoding specified in the initialization file.

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

  \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. [Similar macro `\insertlegend` will insert the map legend when Therion supports it.] This is not done automatically because you may wish to insert some other pages before the first map page. (In this case something like `\input myfile.tex` should preceed the `\insertmaps` macro.) Without the `\insertmaps` invocation no map pages will be included!

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

```
\def\dopage{%
  \vbox{\line{\hfil\box\mapbox\hfil}
    \bigskip
    \line{%
     \vbox to \ht\navbox{
      \hbox{\size[20]\the\pagelabel
        \ifpagenumbering\space(\the\pagenum)\fi
        \space\size[16]\the\pagename}
      \ifpagenumbering
        \medskip
        \hbox{\qquad\qquad
          \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$\leftrightarrow$\hss}%
           \rlap{\hskip0.7em\showpointer\pointerE}}
          \hbox to 0pt{\hss\showpointer\pointerS\hss}
      }\qquad\qquad
          \vtop{
        \def\arr{$\uparrow$}
        \showpointerlist\pointerU
        \def\arr{$\downarrow$}
        \showpointerlist\pointerD
      }
        }
      \fi
      \vss
      \scalebar{10}m
      }\hss
      \box\navbox
    }
  }
```

```
}
```

Using other plain TEX macros or TEX 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.

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 sybols.

Howewer, 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 compehensive reference to the METAFONT language, which is quite similar to METAPOST:

> Knuth, D. E.: *The METAFONTbook*, Reading, Massachusetts, Addison-Wesley [1]1986

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' for point, line or area symols, 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* (thinest), 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 represantation; 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 * xpart U, ypart A * 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 -1 to mark tick in the middle
- 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 defined as patterns with 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.

Example:

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

Appendix

Compilation

If you want to compile the Therion source code, you need

- GNU C/C++ compiler
- GNU make
- Tcl
- Perl

For Windows consider using MinGW (`http://www.mingw.org`). It's a distribution of GNU utilities with GNU make and GCC.

Quick start

- unpack source distribution `therion-0.2.*.tar.gz`
- `cd therion`
- `make config-macosx` or `make config-win32`, if you use MacOS X or Windows, respectively
- `make`
- `make install`

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
- `depend` ▷ find dependencies for include files
- `install` ▷ install Therion
- `clean` ▷ delete all temporary files

Adding new encodings

Although UTF-8 Unicode encoding covers all characters which Therion is able to process, it may be inconveniet 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 TEX 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 `mktexenc.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 TEX 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 `th$PID$`, 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 specifyed in the file therion.ini or xtherion.ini, respectively. They are searched 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.

- `mpost-path <file-path>`

  Set the full path to a METAPOST executable ("`mpost`" is the default).

- `pdftex-path <file-path>`

  Set the full path to a pdfTEX executable ("`pdfetex`" is the default).

- `cavern-path <file-path>`

  Path to `cavern` program. In most cases not necessary.

- `source-path <directory>`

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

- `tmp-path <directory>`

  Path to temporary directory.

- `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 pdfTEX'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 pdfTEX'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.

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 alowed 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, pdfTEX 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 Survex and TeX)

Future

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

General

- comprehensive information on centreline processing and loop closure

2D maps

- map legend
- debugging mode
- SVG support

3D models

- passage walls modelling
- OpenGL 3D viewer as a part of XTherion