PasTEX3.1

AMIGA Implementation

Documentation

of
27. July 1995

by
Georg Heßmann
Bernd Raichle
Translation by Thomas Tavoly

Contents:

1. Copyright

As TEX itself does not originate from me, and the adjustments/changes in it are minimal, this version is Public Domain. Anyone can copy this version and redistribute it.

Take note however, that single parts of the distributed programs (especially the drivers ShowDVI and DVIprint) e.g. Macro-packages do have a Copyright!

2. About TEX

TEX was invented, in order to be able to create high quality printouts with the computer. The goal was to be able to process books and texts of all kinds by the computer, in a way, that they look pretty. This however, does not per se mean the resolution of the printer. TEX typesets texts in a way that they are aesthetically pleasant.

This craftsmanship was only known by typesetters in the past. Normally you need a good education and a sizeable proportion of experience, to fabricate good looking texts. E.g. a text is pleasant to read, when not too many kinds of typefaces are used on a single page, as otherwise the brain can not concentrate on the text properly. Exactly the same is the case with empty lines, to structure a text. These are merely simple basic rules. Typesetters know hundreds of them, and often, texts produced by professionals just look better than others. Sometimes you do not even know why, but they simply look better. Typesetters were additionally also able to correct mistakes of the author – what sometimes led to newly introduced errors.

Now, as a private person with little time and without being equipped with this special knowledge, you still want to print *good looking* texts, this program is supposed to relieve you of many details. TEX is very friendly from this point of view and – unless you do not want it to – takes most of the burden off of you. This way you can concentrate on the contents.

2.1. Man and machine

"Computers are stupid", I still hear being said by childrens voices. And as so often, they are right. Computers are limited. You first have to tell them everything specificly, so that they do what they should be doing. However, as they are not (yet -TT) capable of understanding the human voice (let alone interpreting what you said), you must move to the level of the machine and command the computer with the keyboard what it has to do.

In the age of Mice and GUIs, the keyboard is moving more and more towards oblivion, when typing texts however, it is (fortunately) indispensable as yet. As you have to get the texts into the computer through use of the keyboard anyway, it is also practical to build the commands concerning which texts, how and where are to be placed on a page, into the text.

TEXfunctions the same way. You write a text, as you would do with a typewriter. Every time though, when you want to make something special – like for example printing a word bold or italicized – you have to tell the computer. This happens through the use of commands, that you build into the text right in front of it. How should a computer otherwise understand that a particular word should be printed in a bold typeface?

Though the computer knows how to handle the text, the characters on the screen or on harddisk are again Chinese for the printer. So, the text must be processed in a way that it will produce a good looking text from it. To this end you need the TEXprogram. The conversion is separated into two stages. The first stage is done by the program `virtex` (which means "virgin" TEX). It performs the conversions which are the same for every printer. As a result, it delivers a file, which ends with the suffix `.dvi`. The second stage is taken over by the printerdriver `DVIprint`, which completes the specific conversions for the printers connected. It takes the file ending in `.dvi` and sends the result of its conversion to the printer.

"Should this mean, that I have to compile (convert) every text twice first and even have to print it, before I can see what it looks like?"

Yes and no. Yes, because both of the conversions are always necessary. Without them, you cannot see, what became of the text-command-mix. No because, there is a possibility to convert the texts in a way that you can view them on screen.

To this end, the program with the name `ShowDVI` exists, which converts every file ending in `.dvi` and displays it on the screen. The second stage is thus different. `ShowDVI` does so to say not print out on paper, but to the screen. Without having to waste paper, you can preview this way, whether something became of the text. If it is according to your taste, you employ the `DVIPrint` program to print the document. Should changes be applied, both stages just have to be followed through again.

Summary To create a document with TEX, you have to

- type a text and intersperse it with commands
- convert this with the `virtex` program into a file, that ends with the suffix `.dvi`
- display this with the `ShowDVI` program on the screen and decide whether you like it
- if you don't, start again
- if you do, print it on the printer with `DVIprint`

To some, this method may not seem entirely modern. I dare to pose however, that I am considerably faster with this method than with a so called DTP program and that the printouts are aesthetically more pleasant.

2.2. TEX's parts

TEX consists – apart from the macrofiles – of two programs: `initex` and `virtex`.

With `initex` a format file (ending in '`fmt`') is created. To this end, this program initializes everything from the ground up, reads macros and separation tables and writes its memory contents in a relatively compact format, the format file. This procedure is usually called the "dumping" of a format.

`virtex`, a "virgin" version of `initex`, can now load this format file relatively fast, and can begin with formatting a text. Because of the now superfluous initializing `virtex` needs less memory than `initex`.

Now, what is this format file for?

TEX (more specificly plainTEX) and LATEX themselves are no private programs, but only macro packages dumped in a format file, which you can load with `virtex`. In a shell you can define the following *aliases* to this end, e.g.

```
alias tex    virtex &plain
alias latex  virtex &lplain
```

2.3. Hardware requirements

TEX itself runs with at least 1 MB memory, just for creating of a format files with `initex`, a half MB more does not hurt in any case.

For the Big version of TEX at least 1,5 MB should be available (for dumping accordingly more).

Running TEX with only one, resp. with two disk-drives does function, but is close to masochism. Recommended is in any case a hard-drive with ca. 5–10 MB room (according to how many fonts, macro- and format-files you want to keep on the drive at the same time).

3. Particularities of the implementation

The current implementation is based on the C-version of TEX from the Unix Web2C-Package[1] and has been extended with some other features.

– Configurability
– Selection of the searchpath with environment variables or in a configuration file
– Support of CodePages for mapping at different charactersets in Amiga and in TEX
– inkl. "Statistics"[2]

3.1. Commandline options

On the commandline, some other options can be given, apart from the file that is to be compiled.

The syntax then looks like this:

`initex [`*options*`] [&`*fmt*`[.fmt]] [`*file*`[.tex]]`

bzw.

`virtex [`*options*`] [&`*fmt*`[.fmt]] [`*file*`[.tex]]`

Valid options are:

`-c`*Dir list*[3]

With this you can indicate a list of directories, separated by comma, in which for the configuration file `tex.cnf` is searched. A further possibility would be to set the environment variable `TEXCONFIG` to *dir list*.

[1] This version has thus not been translated manually to C and typed in.

[2] TEX can be compiled also without the so called "Statistics", at which you get a version which runs at ca. 1% faster. Then, however, the `\tracing..`-commands are not available anymore.

`-b` `\batchmode`
`-n` `\nonstopmode`
`-s` `\scrollmode`
`-e` `\errorstopmode` (Default)
Only one of these options at a time can be selected. With this the *Interaction Mode* can be set according to the TEXcommands `\batchmode`, `\nonstopmode`, `\scrollmode` resp. `\errorstopmode`.

`-l`*language number*
From Version 3.0 on, you can load separation tables for up to 256 languages at the same time. Switching between these tables happens through the new TEXcounter `\language`. If you have loaded in position 0 the English and in position 1 the German separations, then you can switch to the German separation through `-l1`, without a change in the file to be formatted. If you give a number > 255 or < 0, then this means `\language=0`, has for the given number no table been loaded, then the separation is switched off[4].

`-d` Debugflag, with this some more or less explicit messages are additionally output.

The options must be given one at a time, separated by spaces. Valid is e.g. `-s -e`, invalid on the other hand is `-se`.

3.2. Environment variables

To be able to work with TEX practically, apart from the program itself, you need some additional files. Where these files should be searched for, can be indicated by environment variables[5] (or in the configuration file).

`TEXINPUTS`
Where should TEX look for the macro-, the style-files and the text to be formatted? Here should not forget to include the current directory (to be indicated as '`.`') in the list. (Files ending in '`tex`', '`sty`')
(*Default:* "`.,tex:macros`")[6]

`TEXFORMATS`
As mentioned before, to be able to format a text, you need a format file, in which e.g. the LaTEXmacro package is contained. (Files ending in '`fmt`')
(*Default:* "`tex:formats`")

`TEXFONTS`
TEX additionally needs information about the fonts used, which can be found in the 'TEX Font Metric' files. (Files ending in '`tfm`')
(*Default:* "`.,tex:fonts`")

`TEXPOOL`

[4] The style option `german.sty` (Version 2.3c) switches e.g. to position 1. Are no separation patterns loaded for this, TEX will not separate anymore.

[5] The standard ENV: variables are used

[6] These default values are directly compiled into the TEXprogram, but can also be partly changed through the config file.

`initex` first reads all strings used in TEX (e.g. error messages) from a "Poolfile". (File '`tex.pool`')
(*Default:* "`.,tex:`")

`EDITOR`
In case there is an error in the input for `initex` or `virtex` (and you are in the interactive mode) you can start an editor with '`e`' You can put the name of the editor you want to be called in this variable. For `%s` the filename, and for `%d` the line number of the error is substituted.
(*Default:* "`ed %s -i`")

`TEXREXX`
This variable determines, whether the editor should be started as a system- command or as an ARexx script. When the variable is not set, the editor is started as a system command. Does the variable contain an arbitrary string the editor is not started directly, but an ARexx script is called. Does `TEXREXX` now contain the string "`edit`", then, instead of waiting for the user to start the editor with the `e` command, it is started right after the first error occurs.

`REXXEDITOR`
This variable indicates which ARexx script should be started, in case the '`e`' command is given and `TEXREXX` is set. Normally, this script is in `rexx:` and has the extension `.rexx`.
(*Default:* "`texedit %s %d`")

`TEXCONFIG`
Searchpath for the configuration file `tex.cnf`. Is also possible through the '`-c`' option when called. (File '`tex.cnf`') This variable is also used by the drivers ShowDVI and DVIprint. There, however, it can not be a list of paths, but only a single path.
(*Default:* "`tex:config`")

3.3. Configuration file "tex.cnf"

The disadvantage of most implementations is the fixed size of the individual arrays in TEX. If new macro packages are published, like e.g. the new *Font Selection Scheme* of LaTEX, then you mostly reach one of the boundaries of the many arrays in TEX (in the above mentioned one, this is the *String Pool*) and the user gets the tidy message: "you can ask a wizard to enlarge me".

With the configuration file `tex.cnf` the user himself can now become a TEX-Wizard. In case the aforementioned message should appear, you simply change the size of the array that is too small.

Four types of information is contained in the configuration file:
(1) Searchpaths
(2) Preselections
(3) CodePage information
(4) Size of TEXarrays

The information is indicated as a keyword-/value pair, where the keyword must begin at the beginning of the line and be separated from the value with spaces or tabs.

Keywords not recognized are ignored without a warning, you can use this to include comments in the file. Better is, however, the use of the percent sign '`%`' as the beginning of the comment. The rest of the line after the percent sign is ignored.

The configuration file is read to the end, but you can stop reading before that by including a '`#`' as the first character in a line. This is meaningful when e.g. a longer comment follows after that.

3.3.1. Searchpaths

Instead of indicating searchpaths through the environment variables, these can also be given here. As keywords are known: `TEXINPUTS`, `TEXFORMATS`, `TEXFONTS` and `TEXPOOL`. These must be written in capitals.

Here, the value of the corresponding environment variable writes over the value in the configuration file.

3.3.2. Default language

The line

```
language    197
```

sets the TEXcounter `\language` to the given value, here 197. This corresponds to '`\language=197` ' on the beginning of the file to be formatted. Should you have given a value on the command line through the `-l` option, then the value in the configuration file is ignored.

3.3.3. TEXArrays

The original TEXsource has been written in the Pascal language without enhancements like those contained in e.g. Turbo Pascal. Owing to this, ports to other systems and languages could be done without major changes. The disadvantage: the size of all the arrays is given static at the point of compilation.

This implementation allows to set the size of most of the arrays that TEX internally uses to be set only at the time of starting the program. What sizes can be set and what meaning these arrays have will be dealt with in a following section.

For values not contained in the configuration file default values are used, when setting repeatedly, the last value is taken.

With the `-d` option you can display the values.

3.3.4. CodePage

The CodePage definition begins with the keyword `codepage` at the beginning of a line, it is ended with two smaller than signs `<<`. In case several definitions are given in one configuration file, then all are regarded as a single definition.

In chapter 5 this is explained thoroughly.

3.3.5. An example configuration file

This is a way the configuration file `tex.cnf` could look:

```
% tex.cnf   Example of a configuration file for TeX
```

```
%
% firstly set all possible environment variables...
%
TEXINPUTS       .,TeX:macros,Work:tex/macros,Work:tex/texts
TEXFORMATS      .,TeX:formats,Work:tex/formats
TEXFONTS        .,TeX:fonts
% only for IniTeX:
TEXPOOL         .,TeX:,Work:tex/pool
%
% ... and now to re set some values
%
stringvacancies 10000
maxstrings       6000
triesize        16000
itriesize       19000    % triesize (for IniTeX only)
memmax          33000
memtop          33000
#
^-- Marks the end of TeX.cnf: '#' on the beginning of the line

Here an arbitrary number of comment lines can now follow.
```

3.4. TEX – ARexx interface

3.4.1. Calling the editor

As in normal Unix[7]-TEX you can also start an editor from PasTEX. Has TEX found an error, then you can call an editor with the '`e`' command.

As already mentioned in chapter 3.2, you can determine that the editor should not be called by a system command but by an ARexx script by setting the variable `TEXREXX`. The means of using a script has been chosen to guarantee a largest possible flexibility.

3.4.2. ARexx scripts

There are also a number of ARexx scripts available, which sort of emulate a "TEX-shell". These scripts are:

`texedit.rexx`
: This script is called by TEX and displays the spot where the error occurred in the TEX-Files in CygnusEd[8] Professional 2.

`start_tex.rexx`
: This script is the central TEX-shell script. This should be started in a CLI with `rx start_tex`.

`start_tex.sd`
: As all ARexx scripts with the extension `sd` this is called from the previewer ShowDVI. It searches for the port of the `start_tex.rexx` script and prompts it to recompile

[7] Unix is a registered Trademark of AT&T...

[8] short CED; this is a commercial editor published by ASDG Inc.

the file which is currently shown in the previewer.

`start_tex.ced`
: The equivalent for the `start_tex.sd` script. Only that this one is called from CED.

`quit_tex.sd`
: With calling this script from the previewer, the script `start_tex.rexx` closes its port and ends itself.

`quit_tex.ced`
: Again, the equivalent for above. Only that this one is called from CED.

`callmf.rexx`
: This script is only interesting in conjunction with Metafont. When the variable `CALLMF` contains "`callmf`", and the drivers cannot find a font, then this script is called, so that this font can be generated. Up till now, however, this script is just a test version!

`cedtofront.sd`
: With this script you can bring the editor CED to the front from ShowDVI.

`sdvitofront.ced`
: This script brings the previewer from editor to the front.

4. Arrays in TEX

In the configuration file the following parameters can be assigned:

`memmax`
`memtop`
: `memmax` resp. `memtop` indicate the size of the most important arrays in TEXİn this "main memory" e.g. the complete page is built and macro definitions are stored.
`initex` ignores the value of `memmax` and sets `memmax` = `memtop`. The value of `memtop` should be chosen as large as possible when dumping with `initex`, this is stored in the format file as well.
For `virtex` and `initex` must apply: `memtop` $\leq$ `memmax` $\leq$ 65534. Thus you can use a bigger "main memory" for `virtex` than for `initex`.
Memory usage: 4 Bytes $*$ `memmax` resp. `memtop`

`triesize`
`itriesize`
`trieopsize`
: All separation tables, that are read by `initex`, are stored very compact in a "Trie". `initex`, however, needs a lot of memory for this Trie, as it has to be completely built first, before it can be compressed[9]. Because of the very large memory requirement of the Trie in `initex`, `membot` must be set to a value $<$ 65534 in case of little memory. As the compressed Trie for `virtex` needs much less space, `memmax` can be usually set to 65534, so that you have the largest possible 'main memory' available for formatting. The size of the Trie is indicated by `triesize` (for `virtex`) resp. `itriesize` (for `initex`). Both must be smaller than 65536. The number of "trie operands" is `trieopsize` ($\leq$ 32767).

[9] This is the main difference between `initex` and `virtex`

Memory usage:

	(i)triesize	trieopsize
initex	15 Bytes	10 Bytes
virtex	5 Bytes	3 Bytes

`fontmax`
`fontmemsize`

`fontmax` gives the maximum number of fonts loaded (must be ≤ 255). The font information from the `tfm` files is loaded into an array of size `fontmemsize`. Where `fontmemsize` can be any size.
Memory usage: 79 Bytes $*$ `fontmax` + 4 Bytes $*$ `fontmemsize`

`maxstrings`
`poolsize`
`stringvacancies`

In an array called "String Pool" all strings are stored, which also includes all TEXmacros and -primitives, like e.g. `\relax`. The size of this array is determined by `poosize`, the number of strings by `maxstrings`. `stringvacancies` indicates, how much space in the String Pool after loading of a format file must at least be present for user defined strings. All three parameters can be chosen to be of infinite size.
Memory usage: 1 Byte $*$ `poolsize` + 4 Byte $*$ `maxstrings`

`bufssize`
`maxinopen`

Every line entered (from a file or from the keyboard) is stored in an array of the size `bufsize`. `bufsize` must be smaller than 65536. `maxinopen` gives the number of simultaniously open files (must be smaller than 128).
Memory usage: 1 Byte $*$ `bufsize` + 8 Byte $*$ `maxinopen`

`maxprintline`
`errorline`
`halferrorline`

`maxprintline` gives the maximum length of the log output to the screen resp. to the log file (usually = 79, must be ≥ 60). `errorline` gives the maximum length of the context information in case of an error (must be ≥ 45), `halferrorline` is the length of the first line of this context ($30 \leq$ `halferrorline` $\leq$ errorline $- 15$).
Memory usage: 1 Byte $*$ `errorline`

`savesize`
`stacksize`
`dvibufsize`

For saving values within groups (local assignments, `\aftergroup`) space of size `savesize` is used. `stacksize` gives the number of "Input-Sources" (file, macros, token-lists, ...). The buffer for writing of the `dvi` file is `dvibufsize` Bytes[10].
Memory usage: 2 Bytes $*$ `savesize` + 10 Bytes $*$ `stacksize` + 1 Byte $*$ `dvibufsize`

What values should be assigned to the individual parameters? For that you simply seek

[10] `dvibufsize` must be divisible by 8!

the help of the log file, which has been created while dumping a format file[11].

When dumping TEX (German `plain.fmt`) you get the following log file:

```
This is a PD-Version of Pas-TeX (made Jan 26 1991 [br]/[hes])
This is TeX, C Version 3.1 (INITEX)  28 JAN 1991 02:51
**plain \input amiga \input /doc/nice \dump
(plain.tex Preloading the plain format: codes, registers,
             ... [lines deleted] ...
Beginning to dump on file plain.fmt
 (format=plain 91.1.28)
2121 strings of total length 28932
7874 memory locations dumped; current usage is 118&7748
1063 multiletter control sequences
\font\nullfont=nullfont
\font\footfont=cmr10
             ... [lines deleted] ...
16011 words of font info for 54 preloaded fonts
0 hyphenation exceptions
Hyphenation trie of length 9980&11780 has 281 ops out of 500
  281 for language 0
No pages of output.
```

From this you recognize the lower bounds for some important parameters: `maxstrings` $>$ 2121, `poolsize` $>$ 28932, `memmax` $>$ 7874, `fontmax` $>$ 54, `fontmemsize` $>$ 16011. The actual values used should be chosen larger: `memmax` should, when possible, be set to 65534, the other parameters to at least 10% larger values.

The Trie with the separation tables can, after it has been compressed, no longer become larger. Because of this you can set `triesize` to a value $\geq$ 9980 and `triopsize` to $\geq$ 281. (For dumping of this format, for `initex`, the parameter `trieopsize` must be set $>$ 11780.)

All other parameters should be left unchanged and only after the error message `TeX capacity exceeded` occurs should you enlarge the particular parameter.

4.1. Difference between TEX–BigTEX

Now, what is this ominous BigTEX, that has been mentioned before in the manual?

This is practically identical with normal TEX, only some of the elements of internal arrays have been enlarged. Consequently, only some arrays can be made bigger, if this is possible in "small" TEX.

These arrays are:

[11] Because of this, these log files should not be deleted.

Array	TEX	BigTEX
`memmax/memtop`	65532	524284
`maxstrings`	65536	*arbitrary*
`bufsize`	65536	*arbitrary*

Of course, you have to pay a price for this achieved flexibility. The memory usage increases considerably. Not only because larger arrays can be allocated now, but also because the individual array elements now use more memory. In case of `memmax` this is about 1.5 times more bytes.

But what do you need such large arrays for?

Really only for a few special cases. The one probably occurring most is the macro package *pictex*. This creates pictures from only characters. You can imagine that a whole lot of individual characters are gathered like this, until a picture is ready. To be able to work reasonably with this macro package, you need BigTEX. However, to be able to work acceptably with BigTEX, you need at least 3MB of main memory.

5. CodePage — What is its ?

TEX uses a coding for the individual characters that is standardized for all implementations, ASCII. This has been fixed for only characters from `0x20` (Space) till `0x7f` (Delete) and some others like `^^@` (NUL), `^^M` (CR), `^^J` (LF), `^^L` (FF) and `^^I` (Tab) though. (As you see, only 7 bit characters and some control characters.)

In case also for special characters like e.g. umlauts a set for the 8 bit characters (characters ≥ 128) exists, in any case a translation of the coding of the Amiga will become necessary to the one used in TEX(A very awkward sentence to translate, I hope I got it right -TT) Now, you can "hardcode" this into the implementation or you leave the user the freedom to be able to determine this in a `Codepage`, so that changes will become quite easy.

The codepage is machine dependant, i.e. if you transfer TEXfiles between two computers with a different character set, you must also change the Codepage. Only the coding that is used internally by TEX is the same on all machines[12].

5.1. Operation

TEX uses two array internally, `xord[]` and `xchr[]` for transformation on input resp. on output. When reading, every character `s` is replaced by the character `t:=xord[s]`, when writing, the character `t` is subsequently again replaced by `s:=xchr[t]`. Eventually, the character of the computer is represented in TEX by the character `t`.

[12] You can take advantage of this by making TEXfiles machine independant: You simply use the notation `^^xx` (xx is a hex number in lower case), `xx` is thereby not(!) the coding of the character in the character set of the computer, but rather the one used by TEX.

Special characters (i.e. control characters and 8 bit characters) are normally represented by `^^xx`, (`xx` is thereby the characters position in the TEXcharacter set in hex notation), this has the advantage that this character can be again read correctly machine independant. For the user though it is making more sense to present displayable special characters as such, that is why TEX uses a further array, in which the display mode is recorded[13].

In the Codepage definition you now have the possibility, to change the default content of this array.

5.2. Syntax

The Codepage definitions are given in the configuration file `tex.cnf`, there you can also give the definition in several parts. Every part is prepended with `codepage` on the start of the line, subsequently a command per line follows. The definition is terminated by `<<`.

Following commands are allowed:

(1) `x = y`
The character `x` is transformed to `y` when reading, `y` is written as `x`, and not in the notation `^^yy` anymore.
Example: (`"a` would be the Amiga character (a letter!) for a-umlaut, `^^80` would be TEXcode for the a-umlaut)
After

```
"a=^^80
```

every `"a` is transformed to 128 immediately, apart from that TEX no longer outputs `^^80` for character 128, but `"a`.
(xord[x]=y; xchr[y]=x; printable(y)=true;)

(2) `x > y`
The character `x` is transformed to `y` when reading. Here, `y` can be chosen from the complete range `^^00-^^ff`.
Example:
After

```
"a>a
```

`"a` is read as `a`.
(xord[x]=y;)

(3) `< y`
The character `y` is no longer output in the notation `^^yy`, but as *one* character `y`. Thereby at the output a possible transformation occurs, in case a command `x = y` has been given in the Codepage definition.
(printable(y)=true;)
Default: all characters < 32 or > 126 are output in the `^^y` notation.
(printable(y)=false; for y in [0...32, 126...255])

(4) `<| y`
Negation of (3). `y` is after that output as `^^yy`.
Example:

[13] This further array is equivalent to the first 256 strings of the *String Pool*.

After

```
"a=^^80
```

the character 128 is output as a character `"a`. If this does not occur, you should give after that:

```
<|^^80
```

(printable(y)=false;)

The individual commands are interpreted line by line, the order in which the commands are executed is important.

General note:

If you give characters in TEX in `^^` notation, then these are not mapped anymore, i.e. with `^^` notation the character code must be in TEX and not in the currently used machine's code.

Only directly given characters are mapped. Mapping takes place when reading in `input_line()`, apart from that when writing with `print_char()` and at the transformation of filenames.

5.3. Limitations

The codepage definition is only useful in `initex`, as the arrays in the format file are stored as well. I.e., if you use several codepages, then for every one of them a new format file must be created. (This limitation should be gone in a later release.)

In the first line that `virtex` reads no translation is taking place yet, as at this point in time, the format file has not yet been read.

6. A testrun

In this chapter I exercise a test run with you, so that you see how texts can be compiled with TEX and how easy that is. I have grouped this chapter in four sections:

1. Installing a directory
2. Compiling with `virtex`
3. Previewing with `ShowDVI`
4. Printing with `DVIprint`

6.1. Installing a directory

Go to the directory in which you want to install a further subdirectory. I have a directory `TeX:documents` in which all subsequent subdirectories are located. You should also create such a directory and then type:

```
cd TeX:documents
makedir testdir
cd testdir
```

Now it is time to type in a text with an editor. This one should end with the postfix `.tex`, otherwise `virtex` will not find it:

```
ed testfile.tex
```

An example for a short but famous text:

```
\hrule
\vskip 1in
\centerline{\bf A SHORT STORY}
\vskip 6pt
\centerline{\sl by A. U. Thor}
\vskip .5cm
Once upon a time, in a distant
  galaxy called \"O\"o\c c,
there lived a computer
named R.~J. Drofnats.

Mr.~Drofnats---or ``R. J.,'' as
he preferred to be called---
was happiest when he was at work
typesetting beautiful documents.
\vskip 1in
\hrule
\vfill\eject
\bye
```

When you have typed in and saved this text, you can start with the compilation.

6.2. Compiling with "virtex"

Before you can translate the text, you should make sure, that all programs and data are present in the right directories and that the environment variables are set correctly. At least the variable `TEXFORMATS` should be set and as an entry contain the name of the directory `TeX:formats` In this directory, a file with the name `plain.fmt` should be present. If you rather want to use the German version of TEX (the macro package is called `plaing.fmt`), then you must substitute the `plain` by `plaing` in the next call.

The call for `virtex` looks like this:

```
virtex &plain testfile
```

After the input you should get the following output on the screen:

```
This is a PD-Version of Pas-TeX (made Jan 22 1991 [br]/[hes])
This is TeX, C Version 3.1
(testfile.tex [1] )
Output written on testfile.dvi (1 page, 672 bytes).
Transcript written on testfile.log.
```

`virtex` has created an output file with the name `testfile.dvi` and a protocol file with the name `testfile.log`.

6.3. Previewing with ShowDVI

The contents of this file can now be displayed by you with a so-called *Previewer*. With this previewer you can look at what the printout will look like *in advance*. The program for this purpose is called `ShowDVI` and is located on one of the PreviewDisks.

Please note that you need more for viewing the document in `testfile.dvi` as what you get on the distribution disks `TeXDisk1` and `TeXDisk2`. For this end, you need the disks PreviewDisks 1, 2 and 3, which you can order from us. It is important that the font libraries are located in the directory `TeX:fontlib` or that you have copied the needed fonts[14] to a place that is known to the previewer.

The call looks like this:

```
ShowDVI testfile
```

After that a new screen in interlace mode should appear and display the result. You can leave `ShowDVI` by pressing `CTRL-C`.

6.4. Printing with DVIprint

The printing takes place in exactly the same way as previewing the document with `ShowDVI`. Please take note again, that you here also need more for printing than what there is to find on both disks `TeXDisk1` and `TeXDisk2`.

As printers of different manufacturers sometimes distinctly differ, you must tell the printer driver, what printer is connected. It can drive 9 pins, 24 pins (NEC-P6 compatible) and HP printers. You can tell it what printer is connected with the `-d` option.

Following calls are for example possible:

For a 9 pins printer:

```
DVIprint -d 5 testfile
```

For a 24 pins printer:

```
DVIprint -d 1 testfile
```

For a HP printer:

```
DVIprint -d 3 testfile
```

After that, you should hold a really *nice* printout in your hands.

[14] Which these are is explained in the ShowDVI manual

7. Errors, problems and solutions

Of course, something always can go wrong. And usually something does go wrong. As you are not a robot, you make mistakes just as much as I do. Programs react to something like this usually quite unsubtly and terminate their work.

TEX reacts to errors with extensive error messages, that are at first very confusing. To explain these here would be too laborious. Please look up the appropriate pages in a book; The books described in the next chapter contain enough information for servicing errors.

Important for the search for errors is, that you must make sure that you have made or set all important assigns, paths and environment variables. Please check in case of an error everything very carefully. Really inspecting everything on your machine, and not only briefly making a mental note, makes the difference, whether you really have done something or not. A call of `dir` or `assign` is often very instructive.

Who has apparently unconquerable problems, can always turn to Georg Heßmann. However, for this, following rules must be respected:

Send us a letter (do not phone, that usually gets you nowhere), in which you clearly explain the problem. Print out a listing of the text, which causes the error to occur, give us a printout of your harddisk directory, make a printout of all settings of the environment variables. If possible, send us a disk with the text, which causes the errors to occur.

And very important: Without a sufficiently stamped envelope, addresses to you, we cannot send you an answer, and will not do it either. This is not malice, but financially necessary. We are students with very low earnings and can only distribute this software for manufacturing costs if we do not suffer a loss at it.

Here the addresses:

Home address:

Georg Heßmann
Oberer Markt 7
8712 Volkach
Germany

Study address:

Georg Heßmann
Ingling 17
4784 Schardenberg
Austria

E-Mail:

`hessmann@unipas.fmi.uni-passau.de`

7.1. Questions and Answers

Here is an attempt at answering some questions in advance.

Question: Does this TEX know *virtual fonts*?
Answer: No, TEX itself has no knowledge of virtual fonts. It only reads the information for the font from the tfm-file (TEX Font Metric) belonging to the font, this exists for "normal", as well as for virtual fonts.

Question: I cannot find `tex`?!?
Answer: `tex` is `virtex` with a format (e.g. plainTEX or LaTEX) to be loaded.

Question: What is a *preloaded format*?
Answer: On some operating systems, you can stop `virtex` after it has loaded the format file and create a so-called *core* file. From this core file, and the executable file `virtex` you can create a further executable file, in which the format file has already been loaded. The advantage, a mostly faster loading of `virtex` and formatfile, is offset by the disadvantage of the space requirements of the new executable file, which is a great deal larger as that of `virtex` only.

Question: I get the following error message, when loading the format file `(Fatal format file error; I'm stymied)`.
Answer: All other filetypes (`tfm`, `dvi`, `pk`, ...) are fully exchangeable between different implementations – on different computers as well – The format file takes a special position in that, as it should be loadable at the greatest speed possible, and represents an image of the internal TEXarrays. Therefore it cannot be exchanged that easily. Workaround: you create a new format file with `initex`.

8. Literature

Without a book, you'll suffer. Even when you partly get TEX very cheaply, you will not get around buying a book, that introduces you to TEXTEX is so powerful as a programming language, knows more commands than any DTP program, grumbles in various ways and now and then produces things that do not make sense. Only a book can help here.

As there is also quite good German literature, I would like to introduce two books that are worth buying.

Einführung in TEX, Norbert Schwarz, Addison Wesley, 1988, ISBN 3-925118-97-7

LaTEX – Eine Einführung, Helmut Kopka, Addison Wesley, 1988, ISBN 3-89319-136-4

If feel your English is adequate (another smiley -TT) there is only one choice: The TEX book by the inventor himself.

The TEXbook, Donald E. Knuth, Addison Wesley, Reading, 1988.

Thank's to Thomas Tavoly for translating the docs!
Georg Heßmann, 30. July 1991.