ȘäferTEX: Source Code Esthetics for Automated Typesetting

Frank-Rene Schaefer
Franzstr. 21
50931 Cologne
Germany
fschaef@users.sourceforge.net
http://safertex.sourceforge.net

Abstract

While TEX [4] provides high quality typesetting features, its usability suffers due to its macro-based command language. Many tools have been developed over the years simplifying and extending the TEX interface, such as LATEX [5], LATEX3 [6], pdfTEX [2], and NTS [8]. Front-ends such as TEXmacs [10] follow the visual/ graphical approach to facilitate the coding of documents. The system introduced in this paper, however, is radical in its targetting of optimized *code appearance*.

The primary goal of ȘäferTEX is to make the typesetting source code as close as possible to human-readable text, to which we have been accustomed over the last few centuries. Using indentation, empty lines and a few triggers allows one to express interruption, scope, listed items, etc. A minimized frame of 'paradigms' spans a space of possible typesetting commands. Characters such as '_' and '$' do not have to be backslashed. Transitions from one type of text to another are automatically detected, with the effect that environments do not have to be bracketed explicitly.

The following paper introduces the programming language ȘäferTEX as a user interface to the TEX typesetting engine. It is shown how the development of a language with reduced redundancy increases the beauty of code appearance.

1 Introduction

The original role of an author in the document production process is to act as an *information source*. To optimize the flow of information, the user has to be freed from tasks such as text layout and document design. The user should be able to delegate the implementation of visual document features and styles to another entity. With this aim in mind, the traditional relationship between an author and his typesetter before the electronic age can be considered the optimal case. Modern technology has increased the speed and reduced the cost of document processing. However, the border between *information specification* and *document design* has blurred or even vanished.

In typesetting engines with a graphical user interface, an editor often takes full control over page breaks, font sizes, paragraph indentation, references and so on. Script-oriented engines such as TEX take care of most typesetting tasks and provide high quality document design. However, quite often the task to produce a document requires detailed insight into the underlying philosophy.

ȘäferTEX tries to get back to the basics, as depicted in Figure 1. Like the traditional writer, a user shall specify information as *redundancy-free* as possible with a minimum of commands that are alien to him. Layout, features, and styles shall be implemented according to predefined standards with a minimum of specification by the user.

To the user, the engine provides a simple interface, only requiring plain text, tables and figures. A second interface allows a human expert to adapt the engine to local requirements of style and output. Ideally, the added features in the second interface do not appear to the user, but are activated from context. Then, the user can concentrate on the core information he wants to produce, and not be distracted by secondary problems of formatting.

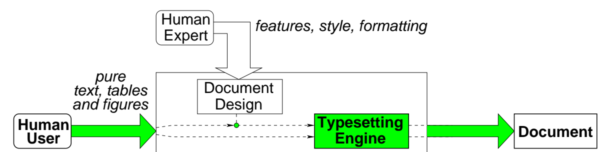

Figure 1: Neo-traditional typesetting.

The abovementioned ideal configuration of engine, user, and expert can hardly be achieved with present automated text formatting systems. While relying on TEX as a typesetting engine, ŞäferTEX tries to progress towards a minimal-redundancy programming language that is at the same time intuitive to the human writer.

2 The ŞäferTEX Engine

As shown in figure 2, the ŞäferTEX engine is based on a three-phase compilation, namely: lexical analysis, parsing and code generation. Along with the usual advantages of such modularization, this structure allows us to describe the engine in a very formal manner. In this early phase of the project, it further facilitates adding new features to the language. Using interfacing tools such as SWIG [1] and .NET [9], it would be possible in the future to pass the generated parse tree to different programming languages. Such an approach would open a whole new world to typesetters and document designers. Plugins for ŞäferTEX could then be designed in the person's favorite programming language (C++, Java, Python, C#, anything). Currently, automated document production mainly happens by preprocessing LATEX code. Using a parse tree, however, gives access to document contents in a structured manner, i.e., through dedicated data structures such as objects of type `section`, `item group`, and so on.

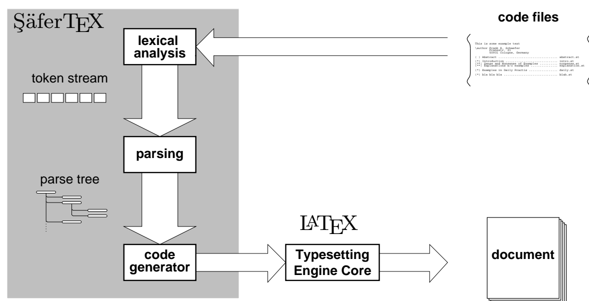

Figure 2: The ŞäferTEX compilation process.

GNU flex [7] (a free software package), is used to create the lexical analyzer. It was, though, necessary to deviate from the traditional idea of a lexical analyzer as a pure finite state automaton. A wrapper around flex implements inheritance between modes (start conditions). Additionally, the lexical analyzer produces implicit tokens and deals with indentation as a scope delimiter.

The parser is developed using the Lemon parser generator [3] (also free software). Using such a program, the ŞäferTEX language can be described with a context-free grammar. The result of the parser is a parse tree, which is currently processed in C++. The final product is a LATEX file that is currently fed into the LATEX engine. A detailed discussion of the engine is not the intention of this paper, though. The present text focuses on the language itself.

3 Means for Beauty

ŞäferTEX tries to optimize code appearance. The author identifies three basic means by which this can be achieved:

1. The first means is intuitive treatment of characters. For example, '$' and '_' are used as normal characters and do not function as commands, as they do in LATEX.

2. The second is to use indentation as the scope delimiter. This is reminiscent of the Python programming language. It allows the user to reduce brackets and enforces proper placement of scopes. For table environments, this principle is extended so that the column positions can be used as cell delimiters.

3. The third principle is automatic environment detection. If an item appears, then the 'itemize' environment is automatically assumed. This reduces redundancy, and makes the source file much more readable.

Applying these principles leads to the eight rules of ŞäferTEX as they are explained at the end (section 5). We now discuss them in more detail.

3.1 Intuitive Treatment of Characters

In the design of a typesetting language, the user has to be given the ability to enter both normal text and commands specifying document structure and non-text content. This can be achieved by defining functions, i.e., using character sequences as triggers for a specific functionality. This happens when we define, say, `sin(x)` as a function computing the sine of `x`. For a typesetter this is not a viable option, since the character chain can be easily confused with normal text. As a result, one would have to 'bracket' normal text or 'backslash' functions. Another solution is to use extra characters. This was the method Donald Knuth chose when he designed TEX [4]. The first solution is still intuitive to most users. The second, however, is rather confusing, implying that '%', '$' and '_' have a meaning different from what one sees in the file.

Historically, at the time TEX was designed, keyboards had a very restricted number of characters. Moreover, ASCII being the standard text encoding in Knuth's cultural context, the high cost of data

Table 1: Comparison of treatment of special characters in LaTeX and ŞäferTeX.

LaTeX:	`According to Balmun \& Refish $<$www.b-and-r.org$>$ a conversion of module \#5, namely ‘propulsion_control,’ into a metric system increases code safety up to 98.7\% at cost of \~ \ \$17,500.`
ŞäferTeX:	`According to Balmun & Refish <www.b-and-r.org> a conversion of module #5, namely ‘propulsion_control,’ into a metric system increases code safety up to 98.7% at cost of ~ $17,500.`

storage, and the lack of advanced programming languages also all may have contributed to the design choices made. Although the documents produced still equal and even outclass most commercial systems of our days, the input language, it must be admitted, is rather cryptic.

The first step towards readability of code is to declare a maximum number of characters as ‘normal’. In ŞäferTeX, the only character that is not considered normal is the backslash. All other characters, such as ‘%’, ‘$’ and ‘_’, appear in the text as they are. Special characters only act abnormal if they appear twice without whitespace in between. These tokens fall into the category of *alien things*, meaning that they look strange and thus are expected to not appear verbatim in the output.

Table 1 compares LaTeX code to ŞäferTeX code, showing the improvement with respect to code appearance. The advantages may seem minor. Consider, however, the task of learning the difference between the characters that can be typed normally and others that have to be backslashed or bracketed. The abovementioned simplification already removes the chance of subtle errors appearing when LaTeX code is compiled. The subsequent sections show how the code appearance and the ease of text input can be further improved.

3.2 Scope by Indentation

In the preceding, we discussed how commands are best defined in a typesetting engine. One way to organize information is to create specific regions, called scopes or environments. Most programming languages use explicit delimiters for scopes without giving any special meaning to white space of

```
Einstein clearly stated his disbelief in the
boundedness of the human spirit as becomes
clear through his sayings:

\quote The difference between genius and
       stupidity is that genius has its limits.

       Only two things are infinite, the
       universe and human stupidity, and I'm
       not sure about the former.

Similar reports have been heard from Frank
Zappa and others.
```

Figure 3: Scope by indentation.

any kind. This implies that the delimiters must be visible. C++, for example, uses curly braces, while LaTeX uses `\begin{...} ... \end{...}` constructs to determine scope. This approach allows one to place the scopes very flexibly. However, it pollutes the text with symbols not directly related to the information being described. The more scopes that are used, and the deeper they are nested, the more the source text loses readability.

Another approach is *scoping by indentation*. A scope of a certain indentation envelopes all subsequent lines and scopes as long as they have more indentation. Figure 3 shows an example of scope by indentation. LaTeX‘s redundancy-rich delimiters add nothing but visual noise to the reader of the file. ŞäferTeX, however, uses a single backslashed command `\quote` in order to open a quote domain. The scope of the quote is then simply closed by the lesser indentation of the subsequent sentence.

This simple example was chosen to display the principle. It is easy to imagine that for more deeply nested scopes (e.g., `picture` in `minipage` in `center` in `figure`), LaTeX code converges to unreadability, while ŞäferTeX code still allows one to get a quick overview about the document structure. Scope by indentation has proven to be a very convenient and elegant tool.

An extension of this concept is using *columns as cell delimiters* in a table scope. The implementation of tables in ŞäferTeX allows the source to omit many ‘parboxes’ and explicit ‘&’-cell delimiters. To begin with, a row is delimited by an empty line. This means that line contents are glued together as long as only one line break separates them. The cell content, though, is collected using the position of the cell markers ‘&&’ and ‘||’. Additionally, the symbol ‘~~’ glues two cells together. This makes cumbersome declarations with `\multicolumn` and

```
\table Food suppliers, prices and amounts.

                 --------------------------------------------------------
   Product      || Price/kg  && Supplier           &&   kg || Total Price
   ======================================================================
   Sugar        ||    $0.25  && Jackie O‘Neil      &&   34 ||    $8.50
   --------------------------------------------------------
   Yellow Swiss ||    $12.2  && United Independent &&  100 || $1220.00
   Cheese                       Farmers of
                                Switzerland
   --------------------------------------------------------
   Green Pepper ||    $25.0  && Anonymous          &&    2 ||   $50.00
   Genuine                      Indians Tribes
   Mexican
   ======================================================================
   Sum          ~~           ~~                    ~~      && $1278.50
   ----------------------------------------------------------------------
```

Figure 4: Example of writing a table: identifying cell borders by column.

`\parbox` unnecessary. Figure 4 shows an example of a ŞäferTEX table definition.

4 Implicit Environment Detection

A basic means of improving convenience of programming is reducing redundancy. In LaTeX, for example, the environment declarations are sometimes unnecessary. To declare a list of items, one has to specify something like

```
\begin{itemize}
   \item This is the first item and
   \item this one is the second.
\end{itemize}
```

Considering the information content, the occurrence of the `\item` should be enough to know that an itemize environment has started. Using our second paradigm, 'scope by indentation', the closing of the environment could be detected by the first text block that has less indentation than the item itself. The `\begin` and `\end` statements are therefore redundant. In ŞäferTEX, the token '`--`' (two dashes) is used to mark an item. Thus, in ŞäferTEX, the item list above simply looks like:

```
   -- This is the first item and
   -- this one is the second.
```

As implied previously, this paradigm's power really unfolds in combination with scope by indentation. Subsequent paragraphs simply need to be indented more than the text block to which they belong. Nested item groups are specified by higher levels of indentation, as seen in figure 5.

Some important points from the example:

- The appearance of a '`--`' at the beginning of a line tells ŞäferTEX that there is an item and that an implicit token 'list begin' has to be created before the token 'item start' is sent. The next '`--`' signals the start of the next item.
- The '`[[`'-symbol appears at the beginning of the line. It indicates a descriptor item. Since it has a higher indentation than the '`--`' items, it is

```
Items provide a good means to

    -- structure information
    -- emphasize important points. There are
       three basic ways to do this:

       [[Numbers]]: Enumerations are good when
                    there is a sequential order
                    in the information being
                    presented.

       [[Descriptions]]: Descriptions are
                         suitable if keywords
                         or key phrases are
                         placeholders for more
                         specific information.

       [[Bullets]]: Normal items indicate that
                    the presented set of
                    information does not
                    define any priorization.

    -- classify basic categories

There may be other things to consider of
which the author is currently unaware.
```

Figure 5: Example code showing 'scope by indentation'.

identified as a nested list. Therefore, an implicit token 'list begin' has to be created again.

- The final sentence having less indentation than anything before closes all lists, i.e., it produces implicit 'list end' tokens for all lists that are to be closed. Thus, the parser and code generator are able to produce environment commands corresponding to the given scopes.

The above has discussed the fundamental ideas to improve programming convenience for a typesetting system. We now turn to defining a best set of rules for expressions that implements these rules.

5 The Eight Rules of ŞäferTEX

Rules for command design shall be consistent with the paradigms of intuitive treatment of characters, scope by indentation, and automatic environment detection. The following set of rules was designed to meet these goals for ŞäferTEX while striving for a intuitive code appearance:

[1] Every character and every symbol in the code appears in the final output as in the source document, except for ALIEN THINGS.

[2] ALIEN THINGS look alien.

In plain TEX, characters such as '$', '%' and '_' do not appear in the document as typed. The fact that they look natural but trigger some TEX specific behavior is prone to confuse the layman. In ŞäferTEX, they appear as typed on the screen. Alien things can be identified by their look. The next four rules define the 'alien look:'

[3] Any word starting with a single backslash `\`. Examples are `\figure` and `\table`.

[4] Any non-letter character that appears twice or more, such as '`##`' (this triggers the start of an enumeration item at the beginning of the line).

[5] Parentheses (at the beginning of a line) that only contain asterisks '`*`' or whitespace. Sequences such as '`(*)`', '`( )`', '`(***)`' indicate sections and subsections.

[6] The very first paragraph of the file. It is interpreted as the title of the document.

Except for the first case, alien things do not interfere with readability. In fact, the double minus '`--`' for items and the '`(*)`' for sections are used naturally in many ASCII files. Internally, alien things are translated into commands for the typesetting engine, but the user does not need to know.

The last two issues are separation of the text stream and identification of scope of an environment:

[7] Termination of paragraphs, interruptions of the text flow, etc., are indicated by an EMPTY LINE.

[8] The scope of an environment, table cells, etc. is determined by its INDENTATION. A line with less indentation closes all scopes of higher indentation.

These are the eight rules of ŞäferTEX which enable one to operate the typesetter. They are defined as 'rules' but, in fact, they do not go much beyond common organization of text files.

6 Commands

This section gives a brief overview of the commands that are currently implemented. In this early stage of development, the system's structure and language design has been in the foreground, in order to build the framework for a more powerful typesetting engine. In the current version of ŞäferTEX, the following commands are implemented:

`--`, `++` starts a bullet item. The two can be used interchangeably to distinguish different levels of nested item groups.

`##` starts an enumeration item.

`[[ ]]` bracket the beginning of a description item.

`\table` opens a table environment. It is followed by a caption and the table body as described in section 3.2.

`\figure` opens a figure environment. The text following this command is interpreted as the caption. Then file names of images are to be listed. Images that are to be shown side by side are separated by '`&&`'. Vertically adjacent images are separated by empty lines.

`\quote` opens a quote environment.

`(*)` starts a section. The number of asterisks indicates the level of the section.

`( )` starts a section without a section number. The number of blanks indicates the section level.

`....` includes a file (more than four dots is equivalent to four). The next non-whitespace character sequence is taken as the filename to be included.

`\author` specifies information about the author of the document.

Commands have been designed for footnotes, labels, and more. However, due to the early stage of development, no definite decision about their format has been made. In the appendix, two example files are listed in order to provide an example of ŞäferTEX code in practical applications.

7 Conclusion and Outlook

Using simple paradigms for improving code appearance and reducing redundancy, a language has been developed that allows more user-friendly input than is currently possible with TEX and LATEX. These paradigms are: the intuitive processing of special characters, the usage of indentation for scope and the implicit identification of environments. As an implementation of these paradigms, the eight rules of ŞäferTEX were formed, which describe the fundamental structure of the language.

While developing ŞäferTEX, the author quickly realized that the ability to provide the parse tree to layout designers extends the usage beyond the domain of TEX. Currently, much effort remains to provide appropriate commands for document production. Functionality of popular tools such as psfrag, fancyheaders, bibtex, makeindex, etc., are to be implemented as part of the language. In the long run, however, it may be interesting to extend its usage towards a general markup language.

8 Acknowledgments

The author would like to thank the TEX Stammtisch of Cologne in Germany for their valuable comments. Special thanks to Holger Jakobs who helped translate this text from Genglish to English.

References

[1] D. M. Beazley. Automated scientific software scripting with SWIG. In *Tools for program development and analysis*, volume 19, pages 599–609. Elsevier Science Publishers B. V., Amsterdam, The Netherlands, 2003.

[2] T. T. Hàn, S. Rahtz, and H. Hagen. The pdftex manual. `http://www.ntg.nl/doc/han/pdftex-a.pdf`, 1999.

[3] R. D. Hipp. The Lemon Parser Generator. `http://www.hwaci.com/sw/lemon`, 1998.

[4] D. E. Knuth. *The TEXbook*. Addison Wesley, 1983.

[5] H. Kopka and P. Daly. *A Guide to LATEX*. Addison Wesley, 1992.

[6] Frank Mittelbach and Chris Rowley. The LATEX3 Project. *TUGboat*, 18(3):195–198, 1997.

[7] J. Poskanzer and V. Paxson. Flex, a fast lexical analyzer generator. `http://sourceforge.net/projects/lex`, 1995.

[8] P. Taylor, J. Zlatuška, and K. Skoupy. *The NTS project: from conception to implementation*. Cahiers GUTenberg, May 2000.

[9] A. Troelsen. *C# and the .NET Platform*. APress, 2001.

[10] Joris van der Hoeven. GNU TEXmacs. `http://www.texmacs.org/tmweb/home/welcome.en.html`, 2003.

```
                          Details about
                   The Elves and The Shoemaker


   \Author Original:
           Brothers Jakob & Wilhelm Grimm
           Somewhere in Germany

( ) Abstract ........................................... abstract.st

(*) Nocturne shoe productions .......................... strange.st
  (**) Living in confusion ............................ confusion.st
  (**) Women make trouble ............................. trouble.st

(*) Midnight observations ............................. midnight.st
   (**) Elves in the cold ............................. freezing-elves.st

(*) New era for elves: luxury ......................... luxury.st

(*) Elves leave their job undone ...................... spoiled-elves.st
```

Figure 6: Example input 'main.st'.

```
\figure ::fig:plots:: Performance a) productivity of shoemaker. b) gain.

   ferry-tales/prod.eps   &&   ferry-tales/capital.eps

Reviewing the plots of shoes produced (figure --<fig:plots>), the shoemaker
realized an instantaneous increase during the night period. He only could
think of two possible reasons:

 ## He was sleepworking. Since he even used to work few when awake this
    assumption was quickly refuted.

 ## Elves must have come over night and did some charity work.

He further based his theory on the influence of the tanning material used. In
fact, there were differences in the number of shoes produced depending on acid
number and pH value (see table --<tab:tan-mat>).

\table ::tab:tan-mat::Influence of tanning materials on shoe production.


     Tanning Mat. &&  pH value   &&  acid number &&  shoes prod.
     -------------------------------------------------------------------
       European    &&  3.4 - 3.7  &&   30 - 40    &&    32         @@
       Indian          2.0 - 2.1       31 - 45          35         @@
       African         4.5 - 4.6       33 - 37          36         @@
       Australian      3.0 - 7.0       27 - 45          15         @@
     -------------------------------------------------------------------

Resourcing several leathers from indian & african suppliers allowed him to
increase profit ranges tremendously. Moreover, these shoes were sold at an
even higher price around $0.50. Pretty soon, the shoemaker was able to save a
good sum of $201.24.
```

Figure 7: Example input 'confusion.st'.