# Extending TEX in a Literate Way

Bernd Raichle

EuroTEX 2001, Kerkrade/NL

- Extending TEX, why?
- The first TEX extension
- `\scantokens`: example of a TEX extension
- Is TEX (easily) extendible?
- NTS—a solution?

*Extending TEX, why?*
*The first TEX . . .*
*\scantokens: . . .*
*Is TEX (easily) . . .*
*NTS— . . .*
*Addendum*

# 1. Extending TEX, why?

When Knuth started, TEX was intented for *one purpose*:

To enable Donald E. Knuth to typeset his book series
"The Art of Computer Programming"

And TEX was meant to be used by *two persons* only:

- Donald E. Knuth
- his secretary.

Nonetheless TEX was used by more and more persons, and Knuth added new features. He saw ...

- the need for extensions,
- the problems of extensions:
  - selecting necessary extensions vs. "featurism",
  - no fixed/stable kernel program for a complete typesetting system,
  - when to finish!

Extending TEX, why?
The first TEX ...
\scantokens: ...
Is TEX (easily) ...
NTS— ...
Addendum

## 1.1. TEX extensions

There is a need for TEX extensions:

- pdfTEX,
- ε-TEX,
- Omega,
- NTS,
- etc.

## 1.2. TEX and its Restrictions

Extensions are possible in different parts of TEX:

- Removing the restrictions of the internal representation in `tex.web` and the standardized files like `tfm` (e.g., 256 glyphs per font, limited number of heights, depths, widths per font).
- Better support on the input level, i.e., text file encodings (e.g., Unicode) or the programming language (e.g., imperative instead of macro language?).
- Better support for color, for new output formats (e.g., PDF).
- Implementing typesetting concepts not supported by TEX (e.g., typesetting on a grid)
- etc.

Extending TEX, why?
The first TEX . . .
\scantokens: . . .
Is TEX (easily) . . .
NTS— . . .
Addendum

# 2. The first TEX extension

The first TEX extension was implemented by Knuth and is integrated into the source code of TEX82 since 1980:

- the text output commands `\write`, `\openout`, `\closeout`, `\immediate`
- the command `\special`

This extension was one of the *most important*, otherwise TEX users were not able . . .

- to create a table of contents or list of figures automatically,
- to communicate with other programs (BibTEX, makeindex, xindy etc.) via text files,
- to incorporate pictures,
- to rotate text (e.g., landscape printing),
- to use colors,
- etc.

Extending TEX, why?
The first TEX . . .
\scantokens: . . .
Is TEX (easily) . . .
NTS— . . .
Addendum

# 3. \scantokens: example of a TEX extension

**From the ε-TEX manual:**

\scantokens{...} absorbs a list of unexpanded tokens, converts it into a character string that is treated as if it were an external file, and starts to read from this 'pseudo-file'. A rather similar effect can be achieved by the commands

```
\toks0={...}
\immediate\openout0=file
\immediate\write0{\the\toks0}
\immediate\closeout0
\input file
```

In particular every occurrence of the current newline character is interpreted as start of a new line, and input characters will be converted into tokens as usual.

The \scantokens command is, however, expandable and does not use token registers, write streams, or external files. Furthermore the conversion from TEX's internal ASCII codes to external characters and back to ASCII codes is skipped.


Extending TEX, why?
The first TEX...
\scantokens:...
Is TEX (easily)...
NTS—...
Addendum

# \scantokens: example of a TEX extension . . . (cont.)

## a. The first steps: simple

1. Define the new primitive

   If possible reuse an existing command code to keep the change simple. In our case, we can use the internal command code for \input and \endinput.

```
primitive("scantokens",input,2);
@!@:scan_tokens_}{\.{\\scantokens} primitive@>
```

2. Provide a print form for trace, log, and string output

```
@x [25] m.377 l.7779 - e-TeX scan_tokens
input: if chr_code=0 then print_esc("input")
  @+else print_esc("endinput");
@y
input: if chr_code=0 then print_esc("input")@/
  else if chr_code=2 then print_esc("scantokens")@/
  else print_esc("endinput");
@z
```

Extending TEX, why?
The first TEX . . .
\scantokens: . . .
Is TEX (easily) . . .
NTS— . . .
Addendum

3. Call the appropriate function when the command token appears in the input stream

```
@x [25] m.378 l.7782 - e-TeX scan_tokens
if cur_chr>0 then force_eof:=true
@y
if cur_chr=1 then force_eof:=true@/
else if cur_chr=2 then pseudo_start@/
@z
```

Extending TEX, why?
The first TEX . . .
\scantokens: . . .
Is TEX (easily) . . .
NTS— . . .
Addendum

# \scantokens: example of a TEX extension . . . (cont.)

## b. The next steps: necessary work

Implement the action which should be done when the command token appears in the input stream, which should be equivalent to . . .

```
\toks0={...}
%\immediate\openout0=file
\immediate\write0{\the\toks0}
%\immediate\closeout0
\input file
```

– Scan the argument resulting in a token list,

```
procedure pseudo_start;
var old_setting:0..max_selector; {holds |selector| setting}
@!s:str_number; {string to be converted into a pseudo file}
@!l,@!m:pool_pointer; {indices into |str_pool|}
@!p,@!q,@!r:pointer; {for list construction}
@!w: four_quarters; {four ASCII codes}
@!nl,@!sz:integer;
begin scan_general_text;
```

Extending TEX, why?
The first TEX . . .
\scantokens: . . .
Is TEX (easily) . . .
NTS— . . .
Addendum

– convert the token list into a string,

```
old_setting:=selector; selector:=new_string;
token_show(temp_head); selector:=old_setting;
flush_list(link(temp_head));
str_room(1); s:=make_string;
```

– write the string to a pseudo file, release the string,

```
@<Convert string |s| into a new pseudo file@>;
flush_string;
```

– start reading this pseudo file.

```
@<Initiate input from new pseudo file@>;
end;
```

Extending TEX, why?
The first TEX . . .
\scantokens: . . .
Is TEX (easily) . . .
NTS— . . .
Addendum

We should not forget to declare and initialize the variable which points to a linked list of open pseudo files.

```
@<Glob...@>=
@!pseudo_files:pointer; {stack of pseudo files}

@ @<Set init...@>=
pseudo_files:=null;
```

At this place we have (hopefully!) not forgotten that it is possible to nest `\scantoken` calls:

```
\scantokens{\scantokens{...}}
```

This means that at the same time, more than one pseudo files can be “open”: To have only a variable pointing to one open pseudo file is not sufficient, it has to point to a list of open pseudo files.


Extending TEX, why?
The first TEX . . .
\scantokens: . . .
Is TEX (easily) . . .
NTS— . . .
Addendum

# \scantokens: example of a TEX extension . . . (cont.)

Here is the code to write the string into a pseudo file, i.e., a list of nodes in TEX's main memory, one node contains all characters of one line and is padded with spaces at the end.

```
@ @<Convert string |s| into a new pseudo file@>=
str_pool[pool_ptr]:=si(" "); l:=str_start[s];
nl:=si(new_line_char);
p:=get_avail; q:=p;
while l<pool_ptr do
  begin m:=l;
  while (l<pool_ptr)and(str_pool[l]<>nl) do incr(l);
  sz:=(l-m+7)div 4;
  if sz=1 then sz:=2;
  r:=get_node(sz); link(q):=r; q:=r; info(q):=hi(sz);
  while sz>2 do
    begin decr(sz); incr(r);
    w.b0:=qi(so(str_pool[m])); w.b1:=qi(so(str_pool[m+1]));
    w.b2:=qi(so(str_pool[m+2])); w.b3:=qi(so(str_pool[m+3]));
    mem[r].qqqq:=w; m:=m+4;
    end;
  w.b0:=qi(" "); w.b1:=qi(" "); w.b2:=qi(" "); w.b3:=qi(" ");
  if l>m then
    begin w.b0:=qi(so(str_pool[m]));
    if l>m+1 then
      begin  w.b1:=qi(so(str_pool[m+1]));
      if l>m+2 then
        begin  w.b2:=qi(so(str_pool[m+2]));
        if l>m+3 then w.b3:=qi(so(str_pool[m+3]));
        end;
      end;
    end;
  mem[r+1].qqqq:=w;
  if str_pool[l]=nl then incr(l);
  end;
info(p):=link(p); link(p):=pseudo_files; pseudo_files:=p
```


Extending TEX, why?
The first TEX . . .
\scantokens: . . .
Is TEX (easily) . . .
NTS— . . .
Addendum

## \scantokens: example of a TEX extension . . . (cont.)

Here is the code to start reading the pseudo file.

```
\begin{verbatim}
@ @<Initiate input from new pseudo file@>=
begin_file_reading; {set up |cur_file| and new level of input}
line:=0; limit:=start; loc:=limit+1; {force line read}
if tracing_scan_tokens>0 then
  begin if term_offset>max_print_line-3 then print_ln
  else if (term_offset>0)or(file_offset>0) then print_char(" ");
  name:=19; print("( "); incr(open_parens); update_terminal;
  end
else name:=18
```

To distinguish the pseudo file from other file descriptors, either 18 (no tracing) or 19 (tracing) is used as TEX's internal name index.


Extending TEX, why?
The first TEX . . .
\scantokens: . . .
Is TEX (easily) . . .
NTS— . . .
Addendum

# \scantokens: example of a TEX extension . . . (cont.)

## c. The finishing steps: more work

After converting the argument of \scantoken into a pseudo file, this file was "opened" to get read by TEX's input mechanism.

If the current file name index 18 or 19 is seen, the input mechanism is reading from the current pseudo file instead of any other file:

```
@x [24] m.362 l.7538 - e-TeX scan_tokens, every_eof
if not force_eof then
@y
if not force_eof then
  if name<=19 then
    begin if pseudo_input then {not end of file}
      firm_up_the_line {this sets |limit|}
    else force_eof:=true;
    end
  else
@z
```

Extending TEX, why?
The first TEX . . .
\scantokens: . . .
Is TEX (easily) . . .
NTS— . . .
Addendum

The line read function pseudo_input copies the characters of one line from the main memory into the input buffer:

```
function pseudo_input: boolean; {inputs the next line or returns |false|}
var p:pointer; {current line from pseudo file}
@!sz:integer; {size of node |p|}
@!w:four_quarters; {four ASCII codes}
@!r:pointer; {loop index}
begin last:=first; {cf.\ Matthew 19\thinspace:\thinspace30}
p:=info(pseudo_files);
if p=null then pseudo_input:=false
else  begin info(pseudo_files):=link(p); sz:=ho(info(p));
  if 4*sz-3>=buf_size-last then
    @<Report overflow of the input buffer, and abort@>;
  last:=first;
  for r:=p+1 to p+sz-1 do
    begin w:=mem[r].qqqq;
    buffer[last]:=w.b0; buffer[last+1]:=w.b1;
    buffer[last+2]:=w.b2; buffer[last+3]:=w.b3;
    last:=last+4;
    end;
  if last>=max_buf_stack then max_buf_stack:=last+1;
  while (last>first)and(buffer[last-1]=" ") do decr(last);
  free_node(p,sz);
  pseudo_input:=true;
  end;
end;
```


Extending TEX, why?
The first TEX . . .
\scantokens: . . .
Is TEX (easily) . . .
NTS— . . .
Addendum

When reaching the end of a file ...

```
@x [23] m.329 l.7047 end_file_reading - e-TeX scan_tokens
if name>17 then a_close(cur_file); {forget it}
@y
if (name=18)or(name=19) then pseudo_close else
if name>17 then a_close(cur_file); {forget it}
@z
```

... we have to make sure to release the used main memory:

```
procedure pseudo_close; {close the top level pseudo file}
var p,@!q: pointer;
begin p:=link(pseudo_files); q:=info(pseudo_files);
free_avail(pseudo_files); pseudo_files:=p;
while q<>null do
  begin p:=q; q:=link(p); free_node(p,ho(info(p)));
  end;
end;
```

Extending TEX, why?
The first TEX ...
\scantokens: ...
Is TEX (easily) ...
NTS— ...
Addendum

## \scantokens: example of a TEX extension . . . (cont.)

### d. The hard steps: Unthought dependencies

The argument token list of\scantokens can contain the \dump command.

When dumping the format file we have to close all open pseudo files, otherwise the used main memory is not released. The used main memory structures are written to the format file without any reference to them: memory leak!

```
@ @<Dump the \eTeX\ state@>=
while pseudo_files<>null do pseudo_close; {flush pseudo files}
```

There are more possible problems to think about:

- Have we thought about \endinput inside \scantokens?
- What about \newlinechar, \endlinechar?
- etc.

Extending TEX, why?
The first TEX . . .
\scantokens: . . .
Is TEX (easily) . . .
NTS— . . .
Addendum

# 4. Is TEX (easily) extendible?

The answer is: **No!**

There are many reasons:

- To extend TEX you have to know its internals: data structures, functions, procedures, WEB macros, restrictions, dependencies.
- TEX is a monolithic program.
- `tex.web` is 600 pages of documented source code (Volume B, "TEX: The Program").
- TEX is written in WEB in a style where the code for one thing is spread all over the WEB source code.

Extending TEX, why?
The first TEX . . .
\scantokens: . . .
Is TEX (easily) . . .
NTS— . . .
Addendum

- TEX was written in 1977–1982
  - using a subset of PASCAL (*1971–1974) because Knuth's used Compiler had not supported all constructs or was not free of bugs,
  - with memory constraints (64K of TEX main memory was unusual at the beginning),
  - memory optimized: TEX has his own memory management and structures (e.g., a memory word is a union of a word, two halves or four quarters),
  - run-time optimized.
- Knuth's coding style is "unusual", his own kind of structured programming.
- No real use of abstract data types.
- Use of global variables instead of local variables and/or function/procedure parameters (historic reason: avoid stack overflow, parameters are slower).
- Functions and procedures are large, have too many lines, instead of much more short supporting functions/procedures (historic reason: limitations of the compiler/run-time system, a function call is slow).

Extending TEX, why?
The first TEX . . .
\scantokens: . . .
Is TEX (easily) . . .
NTS— . . .
Addendum

# 5. NTS—simplifying TEX extensions?

The answer is: **...**

Please ask this question in 10–20 years again! :-)

Extending TEX, why?
The first TEX . . .
\scantokens: . . .
Is TEX (easily) . . .
NTS— . . .
Addendum

Enjoy your own experiences extending TEX.

Extending TEX, why?
The first TEX . . .
\scantokens: . . .
Is TEX (easily) . . .
NTS— . . .
Addendum

# 6. Addendum

Extending TEX, why?
The first TEX . . .
\scantokens: . . .
Is TEX (easily) . . .
NTS— . . .
Addendum

# Historical background

## 6.1. Short summary of TEX's historical dates

**1977, May** Knuth decided to create a program called TEX.

**1977, May 13** `TEX.DRAFT`

**1977, May 18** "Sketched programs for font generation"

**1977, July 12** `TEX.ONE`

**1978, September** Book "Tau Epsilon Chi, a system for technical text" (198 pp)

**1979, September** Book "METAFONT, a system for alphabet design" (105 pp)

**1980, July 29** Book "Seminumerical Algorithms", Volume 2 of TAoCP, second edition.

**1982, October** TEX82, Version 0 released.

**1983, December 3** TEX82, Version 1.0 released.

**1989, August** Work on TEX 3 started.

**1990, March** TEX 3 released.

Extending TEX, why?
The first TEX . . .
\scantokens: . . .
Is TEX (easily) . . .
NTS— . . .
Addendum

## 6.2. Status Quo in 1977/78 to implement TEX

### Structured Programming & (Abstract) Data Types


– Dijkstra, E.W. Notes on Structured Programming. In *Structured Programming*. Academic Press, London, New York. 1972.

– Hoare, C.A.R. Notes on Data Structuring. In *Structured Programming*. Academic Press, London, New York. 1972.


### Programming Language Pascal (used to implement TEX)


– Wirth, N. The Programming Language PASCAL. Acta Informatica, 1, No. 1. 1971. Pp. 35–63.

– Jensen, K.; Wirth, N. PASCAL – *User Manual and Report*. Springer-Verlag, Berlin, Heidelberg, New York. 1974.


### Compiler Theory


– Aho, A.V.; Ullman, J.D. *The Theory in Parsing, Translation, and Compiling*. Prentice-Hall, Eaglewood Cliffs, N.J. 1972+1973. (In two volumes)

– Aho, A.V.; Ullman, J.D. *Principles in Compiler Design*. Addison-Wesley, Reading, Mass. 1977.



*Extending TEX, why?*
*The first TEX . . .*
*\scantokens: . . .*
*Is TEX (easily) . . .*
*NTS— . . .*
*Addendum*