ConTEXt

Documentation

Hans Hagen

PRAGMA

pragma@pi.net — 1997 July 25

Contents

1 The Main Module

1.1 CONTEXT Format Generation

Welcome to the main module. When this module is ran through `initex` or `tex -i` or `whatevertex` using `whatever switch`, the CONTEXT format file is generated. During this process the user is asked for an interface language. Supplying `dutch` will generate a dutch version of CONTEXT, supplying `english` will of course end op in a english version.

First we load the system modules. These implement a lot of manipulation macros. The first one loads PLAIN TEX, as minimal as possible.

1
```
\input syst-tex.tex
\input syst-gen.tex
\input syst-ext.tex
\input syst-new.tex
```

After this we're ready for the multi–lingual interface modules.

2
```
\input mult-ini.tex
\input mult-sys.tex
\input mult-con.tex
\input mult-com.tex
```

Now we're ready for some general support modules. These modules implement some basic typesetting functionality.

3
```
\input supp-ini.tex
\input supp-fil.tex
\input supp-ver.tex
\input supp-box.tex
\input supp-mrk.tex
\input supp-vis.tex
```

```
\input supp-mul.tex
\input supp-fun.tex
\input supp-pdf.tex
\input supp-spe.tex
\input supp-mps.tex
\input supp-tpi.tex
```

CONTEXT does not implement its own table handling. We just go for the best there is and load TABLE. Just to be sure we do it here, before we redefine `|`.

4
```
\doinputonce{table}
```

Here comes the last support module.

5
```
\input supp-lan.tex
```

The next three modules do what their names state. They load additional definition modules when needed.

6
```
\input lang-ini.tex
\input spec-ini.tex
\input colo-ini.tex
```

The special modules need some additional macro's:

7
```
\input spec-mis.tex
```

Next we load some core macro's. These implement the macros' that are seen by the users.

8
```
\input core-gen.tex
\input core-ver.tex
\input core-vis.tex
```

```
\input core-01a.tex
```

Of course we do need fonts. There are no TFM files loaded yet, so the format file is independant of their content.

9
```
\input font-ini.tex
```

Now we're ready for more core modules.

10
```
\input core-fnt.tex
\input core-01b.tex
\input core-01c.tex
\input core-01d.tex
\input core-01e.tex
```

11
```
\input core-02a.tex
\input core-02b.tex
\input core-02d.tex
```

The next two modules implement some additional functionality concenring classes of documents and output.

12
```
\input docs-ini.tex
\input list-ini.tex
```

TEX related logo's are always typeset in a special way. Here they come:

13
```
\input cont-log.tex
```

Dumping the format is all that's left to be done.

14
```
\dump
```

2 System Programming Support

syst-tex
syst-gen
syst-ext
syst-new

contents register context syst mult supp lang font colo spec core cont m s exit go back

2.1 Efficient PLAIN TEX loading

syst-tex
syst-gen
syst-ext
syst-new

We've build CONTEXT on top of PLAIN TEX. Because we want to make the format file as independant as possible of machine dependant font encodings, we have to bypass the loading of fonts.

Let's start at the beginning. Because PLAIN is not yet loaded we have to define some ⟨*catcodes*⟩ ourselves.

1
```
\catcode`\{=1 % left brace is begin-group character
\catcode`\}=2 % right brace is end-group character
\catcode`\#=6 % hash mark is macro parameter character
```

We are going to report to the user what we are skipping.

2
```
\def\skipmessage#1{\immediate\write16{skipping #1 in plain}}
```

We want to be able to use the `\newsomething` declarations not only on the `\outer` level. This can be done by redefining `\outer` so we have to save its original meaning.

3
```
\let\normalouter = \outer
\let\outer       = \relax
```

We also want to postpone the loading of hyphenation patters, so we redefine and therefore save `\input`.

4
```
\let\normalinput = \input
\def\input       #1 {\skipmessage{\string\input}}
```

Finaly are going to we redefine some font specification commands and that's why we save them too. The redefinitions are straightforward because the macros have to do nothing but skipping.

5
```
\let\normalskewchar          = \skewchar
\def\skewchar                #1=#2 {\skipmessage{\string\skewchar}}
```

6
```
\let\normaltextfont         = \textfont
\let\normalscriptfont       = \scriptfont
\let\normalscriptscriptfont = \scriptscriptfont
```

7
```
\def\textfont               #1=#2{\skipmessage{\string\textfont}}
\def\scriptfont             #1=#2{\skipmessage{\string\scriptfont}}
\def\scriptscriptfont       #1=#2{\skipmessage{\string\scriptscriptfont}}
```

The redefinition of `\font` is a bit more complicated, because in version 3.14159 a scaled specification was introduced.

8
```
\let\normalfont = \font
```

9
```
\def\skipscaled scaled #1 {}
```

10
```
\long\def\font#1=#2 #3%
  {\ifx#3s%
     \skipmessage{scaled \string\font}%
     \let\next=\skipscaled
   \else
     \skipmessage{\string\font}%
     \let\next=\relax
   \fi
   \next#3}
```

Relaxing some font switching macros is needed because we don't want any error messages during loading. These unharmfull messages could be ingored.

The next substitution is needed for determining `\p@renwd` in the macro `\bordermatrix`.

11
```
\def\tenex#1%
  {\skipmessage{used \string\tenex}\hskip8.75002pt}
```

We need to define `\tenrm` for switching to `\rm`.

12
```
\def\tenrm%
  {\skipmessage{\string\tenrm}}
```

In CONTEXT all PLAIN TEX fonts are available, just like `\p@renwd`. We only postpone loading them until they are actually needed.

By bypassing fonts, some definitions become less valid so we have to redefine them afterwards.

```
   \let\normalbordermatrix=\bordermatrix

   \def\bordermatrix%
     {\bgroup
      \setbox0=\hbox{\getvalue{\textface\c!mm\c!ex}B}%
      \global\p@renwd=\wd0\relax
      \egroup
      \normalbordermatrix}
```

Now we are ready for loading PLAIN TEX. Of couse we use `\normalinput` and not `\input`.

13
```
\normalinput plain.tex
```

We restore some redefined primitives to their old meaning.

14
```
\let\font                = \normalfont
\let\skewchar            = \normalskewchar
\let\textfont            = \normaltextfont
\let\scriptfont          = \normalscriptfont
\let\scriptscriptfont = \normalscriptscriptfont
\let\input               = \normalinput
\let\outer               = \normalouter
```

syst-tex
syst-gen
syst-ext
syst-new

We reset some of the used auxiliary macro's to `\undefined`. One never knows what testing on them is done elsewhere.

15
```
\let\skipmessage              = \undefined
\let\skipscaled               = \undefined
\let\normalfont               = \undefined
\let\normalskewchar           = \undefined
\let\normaltextfont           = \undefined
\let\normalscriptfont         = \undefined
\let\normalscriptscriptfont = \undefined
\let\normalinput              = \undefined
\let\normalouter              = \undefined
```

We want a bit more statistics and some less logging info in the `log` file.

16
```
\def\wlog#1{}
```

17
```
\tracingstats=1
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

2.2 General

syst-tex
syst-gen
syst-ext
syst-new

The following macros are responsible for the interaction with CONTEXT. These macros have proven their use. These macros are optimized as far as possible within of course, the know how of the author.

In this module we also show some of the optimizations, mainly because we don't want to forget them and start doing things over and over again. If showing them has a learing effect for others too, we've surved another purpose too.

\abortinputifdefined

Because this module can be used in a different context, we want to prevent it being loaded more than once. This can be done using:

```
\abortinputifdefined\command
```

where `\command` is a command defined in the module to be loaded only once.

```
\def\abortinputifdefined#1%
  {\ifx#1\undefined
     \let\next=\relax
   \else
     \let\next=\endinput
   \fi
   \next}
```

This macro can be speed up in terms of speed as well as memory. Because this is a nice example of a bit strange command (`\endinput`), we spend some more lines on this.

If we perform such actions directly, we can say:

```
\ifx\somecommand\undefined
  \let\next=\relax
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
\else
  \let\next=\endinput
\fi
\next
```

We need the `\next` because we need to end the `\fi`. The efficient one is:

```
\ifx\somecommand\undefined
\else
  \expandafter\endinput
\fi
```

Because `\endinput` comes into action after the current line, we can also say:

```
\ifx\somecommand\undefined \else \endinput \fi
```

When we define a macro, we tend to use a format which shows as besat as can how things are done. TEX however stores the definitions as a sequence of tokens, so in fact we can use a formatted definition:

1
```
\def\abortinputifdefined#1%
  {\ifx#1\undefined \else
     \endinput
   \fi}
```

which also works. Keep in mind that this is entirely due to the fact that `\endinput` after the line, i.e. at the end of the macro. We therefore can burry this primitive quite deep in code.

And because this module implements `\writestatus`, we just say:

syst-tex
syst-gen
syst-ext
syst-new

2
```
\abortinputifdefined\writestatus
```

Normally we tell the users what module is being loaded. However, the command that is needed for this is not yet defined.

```
   \writestatus{laden}{Context Systeem Macro's (a)}
```

\protect
\unprotect

We can shield macros from users by using some special characters in their names. Some characters that are normally no letters and therefore often used are: `@`, `!` and `?`. Before and after the definition of protected macros, we have to change the ⟨*catcode*⟩ of these characters. This is done by `\unprotect` and `\protect`, for instance:

```
   \unprotect
   \def\!test{test}
   \protect
```

The defined command `\!test` can of course only be called upon when we are in the `\unprotect`'ed state, otherwise TEX reads `\!` and probably complains loudly about not being in math mode.

Both commands can be used nested, but only the ⟨*catcode*⟩ of the outermost level is saved. We make use of an auxilary macro `\doprotect` to prevent us from conflicts with existing macro's `\protect`. When nesting deeper than one level, the system shows the protection level.

3
```
\newcount\protectionlevel
```

4
```
\ifx\protect\undefined
  \def\protect{\message{<too much protection>}}
\fi
```

5
```
\let\normalprotect=\protect
```

6
```
\def\unprotect%
  {\ifnum\protectionlevel=0
```

```
      \edef\doprotectcharacters%
        {\catcode`@=\the\catcode`@\relax
         \catcode`!=\the\catcode`!\relax
         \catcode`?=\the\catcode`?\relax}%
      \catcode`@=11
      \catcode`!=11
      \catcode`?=11
      \let\protect=\doprotect
    \fi
    \advance\protectionlevel by 1
    \ifnum\protectionlevel>1
      \message{<unprotect \the\protectionlevel>}%
    \fi}
```

7

```
\def\doprotect%
  {\ifnum\protectionlevel=1
      \doprotectcharacters
      \let\protect=\normalprotect
    \fi
    \ifnum\protectionlevel>1
      \message{<protect \the\protectionlevel>}%
    \fi
    \advance\protectionlevel by -1\relax}
```

Now it is defined, we can make use of this very useful macro.

8

```
\unprotect
```

\@@escape
\@@begingroup
\@@endgroup
\@@mathshift
\@@alignment
\@@endofline
\@@parameter
\@@superscript
\@@subscript
\@@ignore
\@@space
\@@letter
\@@other
\@@active
\@@comment

In CONTEXT we sometimes manipulate the ⟨*catcodes*⟩ of certain characters. Because we are not that good at numbers, we introduce some symbolic names.

9
```
\chardef\@@escape      =  0
\chardef\@@begingroup  =  1
\chardef\@@endgroup    =  2
\chardef\@@mathshift   =  3
\chardef\@@alignment   =  4
\chardef\@@endofline   =  5
\chardef\@@parameter   =  6
\chardef\@@superscript =  7
\chardef\@@subscript   =  8
\chardef\@@ignore      =  9
\chardef\@@space       = 10
\chardef\@@letter      = 11
\chardef\@@other       = 12   \chardef\other  = 12
\chardef\@@active      = 13   \chardef\active = 13
\chardef\@@comment     = 14
```

\normalspace

We often need a space as defined in PLAIN TEX. Because we cannot be sure of `\space` is redefined, we define:

10
```
\def\normalspace{ }
```

syst-tex
syst-gen
syst-ext
syst-new

contents register context syst mult supp lang font colo spec core cont m s exit go back

\scratchcounter
\scratchdimen
\scratchskip
\scratchmuskip
\scratchbox
\scratchtoks ifdone

Because we often need counters on a temporary basis, we define the ⟨*counter*⟩ `\scratchcounter`. This is a real ⟨*counter*⟩, and not a pseudo one, as we will meet further on. We also define some other scratch registers.

11
```
\newcount  \scratchcounter
\newdimen  \scratchdimen
\newskip   \scratchskip
\newmuskip \scratchmuskip
\newbox    \scratchbox
\newtoks   \scratchtoks
\newif     \ifdone
```

\ifCONTEXT

In the system and support modules we sometimes show examples that make use of core commands. We can skip those parts of the documentation when we use another macropackage. Of course we default to false.

12
```
\newif \ifCONTEXT
```

syst-tex
syst-gen
syst-ext
syst-new

syst-tex
syst-gen
syst-ext
syst-new

\!!count
\!!toks
\!!dimen
\!!box
\!!width
\!!height
\!!depth
\!!string
\!!done

We define some more ⟨*counters*⟩ and ⟨*dimensions*⟩. We also define some shortcuts to the local scatchregisters 0, 2, 4, 6 and 8.

```
\newcount\!!counta \toksdef\!!toksa=0 \dimendef\!!dimena=0 \chardef\!!boxa=0
\newcount\!!countb \toksdef\!!toksb=2 \dimendef\!!dimenb=2 \chardef\!!boxb=2
\newcount\!!countc \toksdef\!!toksc=4 \dimendef\!!dimenc=4 \chardef\!!boxc=4
\newcount\!!countd \toksdef\!!toksd=6 \dimendef\!!dimend=6 \chardef\!!boxd=6
\newcount\!!counte \toksdef\!!tokse=8 \dimendef\!!dimene=8 \chardef\!!boxe=8
\newcount\!!countf
```
13

```
\def\!!stringa{} \def\!!stringb{} \def\!!stringc{}
\def\!!stringd{} \def\!!stringe{} \def\!!stringf{}
```
14

```
\newdimen\!!widtha \newdimen\!!heighta \newdimen\!!deptha \newif\if!!donea
\newdimen\!!widthb \newdimen\!!heightb \newdimen\!!depthb \newif\if!!doneb
```
15

\s!
\c!
\e!
\p!
\v!
\@@
\??

To save memory, we use constants (sometimes called variables). Redefining these constants can have desastrous results.

```
\def\v!prefix! {v!}          \def\c!prefix! {c!}
\def\s!prefix! {s!}          \def\p!prefix! {p!}

\def\s!next    {next}        \def\s!default {default}
\def\s!dummy   {dummy}       \def\s!unknown {unknown}

\def\s!do      {do}          \def\s!dodo    {dodo}

\def\s!complex {complex}     \def\s!start   {start}
\def\s!simple  {simple}      \def\s!stop    {stop}
```
19

contents register context syst mult supp lang font colo spec core cont m s exit go back

syst-tex
syst-gen
syst-ext
syst-new

\@EA
\expanded

When in unprotected mode, to be entered with `\unprotect`, one can use `\@EA` as equivalent of `\expandafter`.

20
```
\let\@EA=\expandafter
```

Sometimes we pass macros as arguments to commands that don't expand them before interpretation. Such commands can be enclosed with `\expanded`, like:

```
   \expanded{\setupsomething[\alfa]}
```

Such situations occur for instance when `\alfa` is a commalist or when data stored in macros is fed to index of list commands. If needed, one should use `\noexpand` inside the argument. Later on we will meet some more clever alternatives to this command.

21
```
\def\expanded#1%
  {\edef\@@expanded{\noexpand#1}\@@expanded}
```

\gobbleoneargument
\gobble...arguments

The next set of macros just do nothing, except that they get rid of a number of arguments.

22
```
\long\def\gobbleoneargument    #1{}
\long\def\gobbletwoarguments   #1#2{}
\long\def\gobblethreearguments #1#2#3{}
\long\def\gobblefourarguments  #1#2#3#4{}
\long\def\gobblefivearguments  #1#2#3#4#5{}
\long\def\gobblesixarguments   #1#2#3#4#5#6{}
\long\def\gobblesevenarguments #1#2#3#4#5#6#7{}
\long\def\gobbleeightarguments #1#2#3#4#5#6#7#8{}
\long\def\gobbleninearguments  #1#2#3#4#5#6#7#8#9{}
```

syst-tex
syst-gen
syst-ext
syst-new

\doifnextcharelse

When we started using TEX in the late eighties, our first experiences with programming concerned a simple shell around LATEX. The commands probably use most at PRAGMA, are the itemizing ones. One of those few shell commands took care of an optional argument, that enabled us to specify what kind of item symbol we wanted. Without understanding anything we were able to locate a LATEX macro that could be used to inspect the next character.

It's this macro that the ancester of the next one presented here. It executes one of two actions, dependant of the next character. Disturbing spaces and line endings, which are normally interpreted as spaces too, are skipped.

```
   \doifnextcharelse {karakter} {then ...} {else ...}
```

This macro differs from the original in testing on `\endoflinetoken`, which of course we have to define first. We also use `\localnext` because we don't want clashes with `\next`.

23
```
\let\endoflinetoken=^^M
```

24
```
\long\def\doifnextcharelse#1#2#3%
   {\let\charactertoken=#1%
    \def\!!stringa{#2}%
    \def\!!stringb{#3}%
    \futurelet\nexttoken\inspectnextcharacter}
```

25
```
\def\inspectnextcharacter%
  {\ifx\nexttoken\blankspace
     \let\localnext\reinspectnextcharacter
   \else\ifx\!!stringc\endoflinetoken
     \let\localnext\reinspectnextcharacter
   \else\ifx\nexttoken\charactertoken
     \let\localnext\!!stringa
   \else
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
    \let\localnext\!!stringb
  \fi\fi\fi
  \localnext}
```

This macro uses some auxiliary macros. Although we were able to program quite complicated things, I only understood these after rereading the TEXbook. The trick is in using a command with a one character name. Such commands differ from the longer ones in the fact that trailing spaces are *not* skipped. This enables us to indirectly define a long named macro that gobbles a space.

In the first line we define `\blankspace`. Next we make `\:` equivalent to `\reinspect...`. This one–character command is expanded before the next `\def` comes into action. This way the space after `\:` becomes a delimiter of the longer named `\reinspectnextcharacter`. The chain reaction is visually compatible with the next sequence:

```
  \expandafter\def\reinspectnextcharacter %
    {\futurelet\nexttoken\inspectnextcharacter}
```

However complicated it may look, I'm still glad I stumbled into this construction.

```
\def\:{\let\blankspace= }  \:

\def\:{\reinspectnextcharacter}

\expandafter\def\: {\futurelet\nexttoken\inspectnextcharacter}
```

\setvalue
\setgvalue
\setevalue
\setxvalue
\letvalue
\getvalue
\resetvalue

TEX's primitive `\csname` can be used to construct all kind of commands that cannot be defined with `\def` and `\let`. Every macro programmer sooner or later wants macros like these.

```
\setvalue   {naam}{...} = \def\naam{...}
\setgvalue  {naam}{...} = \gdef\naam{...}
\setevalue  {naam}{...} = \edef\naam{...}
\setxvalue  {naam}{...} = \xdef\naam{...}
\letvalue   {naam}=\... = \let\naam=\...
\getvalue   {naam}      = \naam
\resetvalue {naam}      = \def\naam{}
```

As we will see, CONTEXT uses these commands many times, which is mainly due to its object oriented and parameter driven character.

29
```
\def\setvalue#1%
  {\expandafter\def\csname#1\endcsname}
```

30
```
\def\setgvalue#1%
  {\expandafter\gdef\csname#1\endcsname}
```

31
```
\def\setevalue#1%
  {\expandafter\edef\csname#1\endcsname}
```

32
```
\def\setxvalue#1%
  {\expandafter\xdef\csname#1\endcsname}
```

33
```
\def\getvalue#1%
  {\csname#1\endcsname}
```

34
```
\def\letvalue#1%
  {\expandafter\let\csname#1\endcsname}
```

35
```
\def\resetvalue#1%
  {\setvalue{#1}{}}
```

\donottest
\unexpanded

When expansion of a macro gives problems, we can precede it by `\donottest`. It seems that protection is one of the burdens of developers of packages, so maybe that's why in e-TEX protection will be solved in a more robust way.

Sometimes prefixing the macro with `\donottest` leads to defining an auxiliary macro, like

```
\def\dosomecommand {... ... ...}
\def\somecommand   {\donottest\dosomecommand}
```

This double definition can be made transparant by using `\protecte`, as in:

```
\unexpanded\def\somecommand{... ... ...}
```

The protection mechanism uses:

36
```
\def\dontprocesstest#1%
  {==}
```

37
```
\def\doprocesstest#1%
  {#1}
```

38
```
\let\donottest=\doprocesstest
```

By the way, we use a placeholder because we don't want interference when testing on empty strings. Using a placeholder of 8 characters increases the processing time of simple `\doifelse` tests by about 10 %. When we process the test, we have to remove the braces and therefore explictly gobble `#1`.

The fact that many macros have the same prefix, could have a negative impact on searching in the hash table. Because some simple testing does not show differences, we just use:

```
\def\unexpanded#1#2%
  {\@EA#1\@EA#2\@EA{\@EA\donottest\csname\s!do\string#2\endcsname}%
   \@EA#1\csname\s!do\string#2\endcsname}
```

syst-tex
syst-gen
syst-ext
syst-new

contents register context syst mult supp lang font colo spec core cont m s exit go back

Well, in fact we use the bit more versaatile alternative:

39

```
\def\dosetunexpanded#1#2%
  {\@EA#1\@EA{\@EA#2\@EA}%
     \@EA{\@EA\donottest\csname\s!do\@EA\string\csname#2\endcsname\endcsname}%
   \@EA#1{\s!do\@EA\string\csname#2\endcsname}}
```

40

```
\def\docomunexpanded#1#2%
  {\@EA#1\@EA#2\@EA{\@EA\donottest\csname\s!do\string#2\endcsname}%
   \@EA#1\csname\s!do\string#2\endcsname}
```

41

```
\def\unexpanded#1%
  {\def\dounexpanded%
     {\ifx\next\bgroup
        \@EA\dosetunexpanded
      \else
        \@EA\docomunexpanded
      \fi#1}%
   \futurelet\next\dounexpanded}
```

This one accepts the more direct `\def` and cousins as well as the CONTEXT specific `\setvalue` ones.

And so the definition in our example turns out to be:

```
\def\csname do\somecommand\endcsname{... ... ...}
\def\somecommand{\donottest\csname do\somecommand\endcsname}
```

In which `do\somecommand` is hidden from the user and cannot lead to confusion. It's still permitted to define auxiliary macros like `\dosomecommand`.

When we are going to use e-TEX, we'll probably end up redefining some commands, but we can probably keep the `\unexpanded` ones unchanged.

\doifundefined
\doifdefined
\doifundefinedelse
\doifdefinedelse
\doifalldefinedelse

The standard way of testing if a macro is defined is comparing its meaning with another undefined one, usually `\undefined`. To garantee correct working of the next set of macros, `\undefined` may never be defined!

```
\doifundefined      {string}    {...}
\doifdefined        {string}    {...}
\doifundefinedelse  {string}    {then ...} {else ...}
\doifdefinedelse    {string}    {then ...} {else ...}
\doifalldefinedelse {commalist} {then ...} {else ...}
```

Every macroname that TEX builds gets an entry in the hash table, which is of limited size. It is expected that e-TEX will offer a less memory–consuming alternative.

Although it will probably never be a big problem, it is good to be aware of the difference between testing on a macro name to be build by using `\csname` and `\endcsname` and testing the `\name` directly.

```
\expandafter\ifx\csname NameA\endcsname\relax ... \else ... \fi

\ifx\NameB\undefined ... \else ... \fi
```

I became aware of this when I mistakenly testen the first one against `\undefined`. When TEX build a name using `\csname` it automatically sets it to `\relax`, which is definitely not the same as `\undefined`. The quickest way to check these things is asking TEX to show the meaning of the names:

```
\expandafter\show\csname NameA\endcsname

\show\NameB
```

The main reason why this never will be a big problem is that when one uses the `\csname` way, one probably has to do with some macroname that always is dealt with that way. Confusion can however

syst-tex
syst-gen
syst-ext
syst-new

contents register context syst mult supp lang font colo spec core cont m s exit go back

arise when one applies both testing methods to the same macroname. By the way, the assignment of `\relax` obeys grouping.

The first one gets rid of `#1`, but still expands to something and the second one expands to `#1`. Because we accept arguments between `{}`, we have to get rid of one level of braces.

Our first implementation of `\ifundefined` was straightforward and readable:

```
\def\ifundefined#1%
  {\expandafter\ifx\csname#1\endcsname\relax}%

\def\doifundefinedelse#1#2#3%
  {\let\donottest=\dontprocesstest
   \ifundefined{#1}%
     \let\donottest=\doprocesstest#2%
   \else
     \let\donottest=\doprocesstest#3%
   \fi}

\def\doifdefinedelse#1#2#3%
  {\doifundefinedelse{#1}{#3}{#2}}

\def\doifundefined#1#2%
  {\doifundefinedelse{#1}{#2}{}}

\def\doifdefined#1#2%
  {\doifundefinedelse{#1}{}{#2}}

\def\doifalldefinedelse#1#2#3%
  {\bgroup
```

```
    \donetrue
    \def\checkcommand##1%
      {\doifundefined{##1}{\donefalse}}%
    \processcommalist[#1]\checkcommand
    \ifdone
      \egroup#2%
    \else
      \egroup#3%
    \fi}
```

When this module was optimized, timing showed that the next alternative can be upto twice as fast, especially when longer arguments are used.

42
```
\def\ifundefined#1%
  {\expandafter\ifx\csname#1\endcsname\relax}
```

43
```
\def\p!doifundefined#1%
  {\let\donottest=\dontprocesstest
   \expandafter\ifx\csname#1\endcsname\relax}
```

44
```
\def\doifundefinedelse#1#2#3%
  {\p!doifundefined{#1}%
     \let\donottest=\doprocesstest#2%
   \else
     \let\donottest=\doprocesstest#3%
   \fi}
```

45
```
\def\doifdefinedelse#1#2#3%
  {\p!doifundefined{#1}%
     \let\donottest=\doprocesstest#3%
   \else
```

```
    \let\donottest=\doprocesstest#2%
  \fi}
```

46
```
\def\doifundefined#1#2%
  {\p!doifundefined{#1}%
    \let\donottest=\doprocesstest#2%
  \else
    \let\donottest=\doprocesstest
  \fi}
```

47
```
\def\doifdefined#1#2%
  {\p!doifundefined{#1}%
    \let\donottest=\doprocesstest
  \else
    \let\donottest=\doprocesstest#2%
  \fi}
```

Before we start using this variant, we used another one, which is even a bit faster. This one looked like:

```
  \def\p!doifundefined%
    {\begingroup
     \let\donottest=\dontprocesstest
     \ifundefined}

  \def\doifundefinedelse#1#2#3%
    {\p!doifundefined{#1}%
       \endgroup#2%
     \else
       \endgroup#3%
     \fi}
```

syst-tex
syst-gen
syst-ext
syst-new

A even more previous version used `\bgroup` and `\egroup`. In math mode however, `$1{x}2$` differs from `$1x2$`. This can been seen when one compares the output of:

```
$\kern10pt\showthe\lastkern$
$\kern10pt{\showthe\lastkern}$
$\kern10pt\begingroup\showthe\lastkern\endgroup$
```

When we were developing the scientific units module, we encountered different behavior in text and math mode, which was due to this grouping subtilities. We therefore decided to use `\begingroup` instead of `\bgroup`. Later, when we had optimized some macro's the grouped solution turned out to be unsafe when typesetting this documentation, especially when using `\globaldefs`.

We still have to define `\doifalldefinedelse`. Watch the use of grouping, which garantees local use of the boolean `\ifdone`.

48
```
\def\docheckonedefined#1%
  {\ifundefined{#1}%
     \donefalse
   \fi}
```

49
```
\def\doifalldefinedelse#1#2#3%
  {\begingroup
   \let\donottest=\dontprocesstest
   \donetrue
   \processcommalist[#1]\docheckonedefined
   \ifdone
     \endgroup\let\donottest=\doprocesstest#2%
   \else
     \endgroup\let\donottest=\doprocesstest#3%
   \fi}
```

\doif
\doifelse
\doifnot
\donottest

syst-tex
syst-gen
syst-ext
syst-new

Programming in TEX differs from programming in procedural languages like MODULA. This means that one — well, let me speek for myself — tries to do the things in the well known way. Therefore the next set of `\ifthenelse` commands were between the first ones we needed. A few years later, the opposite became true: when programming in MODULA, I sometimes miss handy things like grouping, runtime redefinition, expansion etc. While MODULA taught me to structure, TEX taught me to think recursive.

```
\doif     {string1} {string2} {...}
\doifnot  {string1} {string2} {...}
\doifelse {string1} {string2} {then ...}{else ...}
```

When expansion gives problems, we can precede the troublemaker with `\donottest`.

This implementatie does not use the construction which is more robust for nested conditionals.

```
\ifx\!!stringa\!!stringb
  \def\next{#3}%
\else
  \def\next{#4}%
\fi
\next
```

In practice, this alternative is at least 20% slower than the alternative used here. The few cases in which we really need the `\next` construction, often need some other precautions and or adaptions too.

```
\long\def\doif#1#2#3%
  {\let\donottest=\dontprocesstest
   \edef\!!stringa{#1}%
   \edef\!!stringb{#2}%
   \let\donottest=\doprocesstest
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

syst-tex
syst-gen
syst-ext
syst-new

```
   \ifx\!!stringa\!!stringb
     #3%
   \fi}
```

51
```
\long\def\doifnot#1#2#3%
  {\let\donottest=\dontprocesstest
   \edef\!!stringa{#1}%
   \edef\!!stringb{#2}%
   \let\donottest=\doprocesstest
   \ifx\!!stringa\!!stringb
   \else
     #3%
   \fi}
```

52
```
\long\def\doifelse#1#2#3#4%
  {\let\donottest=\dontprocesstest
   \edef\!!stringa{#1}%
   \edef\!!stringb{#2}%
   \let\donottest=\doprocesstest
   \ifx\!!stringa\!!stringb
     #3%
   \else
     #4%
   \fi}
```

One could wonder why we don't follow the the same approach as in `\doifdefined` c.s. and use `\begingroup` and `\endgroup`. In this case, this alternative is slower, which is probably due to the fact that more meanings need to be restored.

The in terms of memory more efficient alternative using a auxiliary macro also proved to be slower, so we definitely did not choose for:

contents register context syst mult supp lang font colo spec core cont m s exit go back

syst-tex
syst-gen
syst-ext
syst-new

```
   \def\p!doifelse#1#2%
     {\let\donottest=\dontprocesstest
      \edef\!!stringa{#1}%
      \edef\!!stringb{#2}%
      \let\donottest=\doprocesstest
      \ifx\!!stringa\!!stringb}

   \long\def\doif#1#2#3%
     {\p!doifelse{#1}{#2}#3\fi}

   \long\def\doifnot#1#2#3%
     {\p!doifelse{#1}{#2}\else#3\fi}

   \long\def\doifelse#1#2#3#4%
     {\p!doifelse{#1}{#2}#3\else#4\fi}
```

Optimizations like this are related of course to the bottlenecks in TEX. It seems that restoring saved meanings and passing arguments takes some time.

\doifempty
\doifemptyelse
\doifnotempty

We complete our set of conditionals with:

```
   \doifempty     {string} {...}
   \doifnot       {string} {...}
   \doifemptyelse {string} {then ...} {else ...}
```

This time, the string is not expanded.

53
```
\long\def\doifemptyelse#1#2#3%
  {\def\!!stringa{#1}%
   \ifx\!!stringa\empty
     #2%
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
   \else
     #3%
   \fi}
```

54
```
\long\def\doifempty#1#2%
  {\def\!!stringa{#1}%
   \ifx\!!stringa\empty
     #2%
   \fi}
```

55
```
\long\def\doifnotempty#1#2%
  {\def\!!stringa{#1}%
   \ifx\!!stringa\empty
   \else
     #2%
   \fi}
```

\doifinset
\doifnotinset
\doifinsetelse

We can check if a string is present in a comma separated set of strings. Depending on the result, some action is taken.

```
   \doifinset     {string} {string,...} {...}
   \doifnotinset  {string} {string,...} {...}
   \doifinsetelse {string} {string,...} {then ...} {else ...}
```

The second argument is the comma separated set of strings.

```
   \long\def\doifinsetelse#1#2#3#4%
     {\doifelse{#1}{}
        {#4}
        {\donefalse
         \def\v!checkiteminset##1%
```

syst-tex
syst-gen
syst-ext
syst-new

```
           {\doif{#1}{##1}
              {\donetrue
               \let\v!checkiteminset=\gobbleoneargument}}%
         \processcommalist[#2]\v!checkiteminset
         \ifdone
           #3%
         \else
           #4%
         \fi}}

   \long\def\doifinset#1#2#3%
     {\doifinsetelse{#1}{#2}{#3}{}}

   \long\def\doifnotinset#1#2#3%
     {\doifinsetelse{#1}{#2}{}{#3}}
```

Because this macro is called quite often we've spent some time optimizing it. This time, the gain in speed is due to (1) defining an external auxiliary macro, (2) not calling any other macros and (3) minimizing the passing of arguments. The gain in speed is impressive.

56
```
\def\p!dodocheckiteminset#1%
  {\edef\!!stringb{#1}%
   \ifx\!!stringa\!!stringb
     \donetrue
     \let\p!docheckiteminset=\gobbleoneargument
   \fi}
```

57
```
\def\p!doifinsetelse#1#2%
  {\let\donottest=\dontprocesstest
   \donefalse
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
    \edef\!!stringa{#1}%
    \ifx\!!stringa\empty
    \else
      \let\p!docheckiteminset=\p!dodocheckiteminset
      \processcommalist[#2]\p!docheckiteminset
    \fi
    \let\donottest=\doprocesstest
    \ifdone}
```

58
```
\long\def\doifinsetelse#1#2#3#4%
  {\p!doifinsetelse{#1}{#2}%
     #3%
   \else
     #4%
   \fi}
```

59
```
\long\def\doifinset#1#2#3%
  {\p!doifinsetelse{#1}{#2}%
     #3%
   \fi}
```

60
```
\long\def\doifnotinset#1#2#3%
  {\p!doifinsetelse{#1}{#2}%
   \else
     #3%
   \fi}
```

\doifcommon
\doifnotcommon
\doifcommonelse

Probably the most time consuming tests are those that test for overlap in sets of strings.

```
\doifcommon     {string,...} {string,...} {...}
\doifnotcommon  {string,...} {string,...} {...}
\doifcommonelse {string,...} {string,...} {then ...} {else ...}
```

We show the slower alternative first, because it shows us how things are done.

```
\long\def\doifcommonelse#1#2#3#4%
  {\donefalse
   \def\p!docommoncheck##1%
     {\def\p!dodocommoncheck####1%
        {\doif{####1}{##1}
           {\donetrue
            \def\commalistelement{##1}%
            \let\p!docommoncheck=\gobbleoneargument
            \let\p!dodocommoncheck=\gobbleoneargument}}%
      \processcommalist[#2]\p!dodocommoncheck}%
   \processcommalist[#1]\p!docommoncheck
   \ifdone
     #3%
   \else
     #4%
   \fi}

\long\def\doifcommon#1#2#3%
  {\doifcommonelse{#1}{#2}{#3}{}}

\long\def\doifnotcommon#1#2#3%
  {\doifcommonelse{#1}{#2}{}{#3}}
```

syst-tex
syst-gen
syst-ext
syst-new

contents register context syst mult supp lang font colo spec core cont m s exit go back

The processing time is shortened by getting the auxiliary macro to the outermost level and using less `\edef`'s. Sometimes it makes more sence to define local macro's not only because this way we can be sure that they are not redefined, but also because it shows the dependance. In compiled languages, this is no problem at all. It can even save us bytes and processing time. In interpreted languages like TEX it nearly always slows down processing.

61
```
\def\p!dododocommoncheck#1%
  {\edef\!!stringb{#1}%
   \ifx\!!stringa\!!stringb
     \donetrue
     \let\p!docommoncheck\gobbleoneargument
     \let\p!dodocommoncheck\gobbleoneargument
   \fi}
```

62
```
\def\p!doifcommonelse#1#2%
  {\donefalse
   \let\donottest\dontprocesstest
   \let\p!dodocommoncheck\p!dododocommoncheck
   \def\p!docommoncheck##1%
     {\edef\!!stringa{##1}%
      \def\commalistelement{##1}%
      \processcommalist[#2]\p!dodocommoncheck}%
   \processcommalist[#1]\p!docommoncheck
   \let\donottest\doprocesstest
   \ifdone}
```

63
```
\long\def\doifcommonelse#1#2#3#4%
  {\p!doifcommonelse{#1}{#2}%
     #3%
   \else
```

syst-tex
syst-gen
syst-ext
syst-new

```
    #4%
  \fi}
```

64
```
\long\def\doifcommon#1#2#3%
  {\p!doifcommonelse{#1}{#2}%
    #3%
  \fi}
```

65
```
\long\def\doifnotcommon#1#2#3%
  {\p!doifcommonelse{#1}{#2}%
  \else
    #3%
  \fi}
```

\processcommalist
\processcommacommand
\processcommalistwithp..

We've already seen some macros that take care of comma separated lists. Such list can be processed with

```
  \processcommalist[string,string,...]\commando
```

The user supplied command `\commando` receives one argument: the string. This command permits nesting and spaces after commas are skipped. Empty sets are no problem.

```
  \def\dosomething#1{(#1)}

  \processcommalist [\hbox{$a,b,c,d,e,f$}] \dosomething \par
  \processcommalist [{a,b,c,d,e,f}]        \dosomething \par
  \processcommalist [{a,b,c},d,e,f]        \dosomething \par
  \processcommalist [a,b,{c,d,e},f]        \dosomething \par
  \processcommalist [a{b,c},d,e,f]         \dosomething \par
  \processcommalist [{a,b}c,d,e,f]         \dosomething \par
  \processcommalist []                     \dosomething \par
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

syst-tex
syst-gen
syst-ext
syst-new

```
  \processcommalist [{[}]                   \dosomething \par
```

Before we show the result, we present the macro's:

66
```
\newcount\commalevel
```

67
```
\def\dodododoprocesscommaitem%
  {\csname\s!next\the\commalevel\endcsname}
```

68
```
\def\dodoprocesscommaitem%
  {\ifx\nexttoken\blankspace
     \let\nextcommaitem\redoprocesscommaitem
  %\else\ifx\nexttoken\endoflinetoken
    %\let\nextcommaitem\redoprocesscommaitem
   \else\ifx\nexttoken]%
     \let\nextcommaitem=\gobbleoneargument
   \else
     \let\nextcommaitem=\dododoprocesscommaitem
   \fi\fi%\fi
   \nextcommaitem}
```

69
```
\def\doprocesscommaitem%
  {\futurelet\nexttoken\dodoprocesscommaitem}
```

70
```
\def\doprocesscommalist#1]#2%
  {\advance\commalevel by 1\relax
   \long\expandafter\def\csname\s!next\the\commalevel\endcsname##1,%
     {#2{##1}\doprocesscommaitem}%
   \doprocesscommaitem#1,]\relax
   \advance\commalevel by -1\relax}
```

Empty arguments are not processed. Empty items (,,) however are treated.

contents register context syst mult supp lang font colo spec core cont m s exit go back

71
```
\def\docheckcommaitem%
  {\ifx\nexttoken]%
     \let\nextcommaitem=\gobbletwoarguments
   \else
     \let\nextcommaitem=\doprocesscommalist
   \fi
   \nextcommaitem}
```

72
```
\def\processcommalist[%
  {\futurelet\nexttoken\docheckcommaitem}
```

We use the same hack for checking the next character, that we use in `\doifnextcharelse`.

73
```
\def\:{\redoprocesscommaitem}
```

74
```
\expandafter\def\: {\futurelet\nexttoken\dodoprocesscommaitem}
```

The previous examples lead to:

(a, b, c, d, e, f)

(a)(b)(c)(d)(e)(f)

(a,b,c)(d)(e)(f)

(a)(b)(c,d,e)(f)

(ab,c)(d)(e)(f)

(a,bc)(d)(e)(f)

([)

When a list is saved in a macro, we can use a construction like:

```
\expandafter\processcommalist\expandafter[\list]\command
```

Such solutions suit most situations, but we wanted a bit more.

```
\processcommacommand[string,\stringset,string]\commando
```

where `\stringset` is a predefined set, like:

```
\def\first{aap,noot,mies}
\def\second{laatste}

\processcommacommand[\first]\message
\processcommacommand[\first,second,third]\message
\processcommacommand[\first,between,\second]\message
```

Commands that are part of the list are expanded, so the use of this macro has it slimits.

75
```
\def\processcommacommand[#1]%
  {\edef\commacommand{#1}%
   \toks0=\expandafter{\expandafter[\commacommand]}%
   \expandafter\processcommalist\the\toks0 }
```

The argument to `\command` is not delimited. Because we often use `[]` as delimiters, we also have:

```
\processcommalistwithparameters[string,string,...]\command
```

where `\command` looks like:

```
\def\command[#1]{... #1 ...}
```

76
```
\def\processcommalistwithparameters[#1]#2%
  {\def\docommand##1{#2[##1]}%
   \processcommalist[#1]\docommand}
```

\processaction
\processfirstactioninset
\processallactionsinset

syst-tex
syst-gen
syst-ext
syst-new

CONTEXT makes extensive use of a sort of case or switch command. Depending of the presence of one or more provided items, some actions is taken. These macros can be nested without problems.

```
\processaction            [x]     [a=>\a,b=>\b,c=>\c]
\processfirstactioninset  [x,y,z] [a=>\a,b=>\b,c=>\c]
\processallactionsinset   [x,y,z] [a=>\a,b=>\b,c=>\c]
```

We can supply both a `default` action and an action to be undertaken when an `unknown` value is met:

```
\processallactionsinset
  [x,y,z]
  [      a=>\a,
         b=>\b,
         c=>\c,
   default=>\default,
   unknown=>\unknown{... \commalistelement ...}]
```

When `#1` is empty, this macro scans list `#2` for the keyword `default` and executed the related action if present. When `#1` is non empty and not in the list, the action related to `unknown` is executed. Both keywords must be at the end of list `#2`. Afterwards, the actually found keyword is available in `\commalistelement`. An advanced example of the use of this macro can be found in PPCHTEX, where we completely rely on TEX for interpreting user supplied keywords like `SB`, `SB1..6`, `SB125` etc.

Even a quick glance at the macros below show some overlap, which means that more efficient alternatives are possible. Because these macro's are very sensitive to subtle changes, we've decided to present the readable originals first Maybe these these macros look complicated, but this is a direct result of the support of nesting. Protection is only applied in `\processaction`.

```
\newcount\processlevel
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
\def\processaction[#1]#2[#3]%
  {\doifelse{#1}{}
     {\def\c!compareprocessaction[##1=>##2]%
        {\edef\!!stringa{##1}%
         \ifx\!!stringa\s!default
           \def\commalistelement{#1}%
           ##2%
         \fi}}
     {\let\donottest=\dontprocesstest
      \edef\!!stringb{#1}%
      \let\donottest=\doprocesstest
      \def\c!compareprocessaction[##1=>##2]%
        {\edef\!!stringa{##1}%
         \ifx\!!stringa\!!stringb
           \def\commalistelement{#1}%
           ##2%
           \let\c!doprocessaction=\gobbleoneargument
         \else\ifx\!!stringa\s!unknown
           \def\commalistelement{#1}%
           ##2%
         \fi\fi}}%
   \def\c!doprocessaction##1%
     {\c!compareprocessaction[##1]}%
   \processcommalist[#3]\c!doprocessaction}

\def\processfirstactioninset[#1]#2[#3]%
  {\doifelse{#1}{}
     {\processaction[][#3]}
     {\def\c!compareprocessaction[##1=>##2][##3]%
```

```
            {\edef\!!stringa{##1}%
             \edef\!!stringb{##3}%
             \ifx\!!stringa\!!stringb
               \def\commalistelement{##3}%
               ##2%
               \let\c!doprocessaction=\gobbleoneargument
               \let\c!dodoprocessaction=\gobbleoneargument
             \else\ifx\!!stringa\s!unknown
               \def\commalistelement{##3}%
               ##2%
             \fi\fi}%
          \def\c!doprocessaction##1%
            {\def\c!dodoprocessaction####1%
               {\c!compareprocessaction[####1][##1]}%
             \processcommalist[#3]\c!dodoprocessaction}%
          \processcommalist[#1]\c!doprocessaction}}

    \def\processallactionsinset[#1]#2[#3]%
      {\doifelse{#1}{}
         {\processaction[][#3]}
         {\advance\processlevel by 1\relax
          \def\c!compareprocessaction[##1=>##2][##3]%
            {\edef\!!stringa{##1}%
             \edef\!!stringb{##3}%
             \ifx\!!stringa\!!stringb
               \def\commalistelement{##3}%
               ##2%
               \let\c!dodoprocessaction=\gobbleoneargument
             \else\ifx\!!stringa\s!unknown
```

syst-tex
syst-gen
syst-ext
syst-new

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
              \def\commalistelement{##3}%
              ##2%
            \fi\fi}%
         \setvalue{\s!do\the\processlevel}##1%
           {\def\c!dodoprocessaction####1%
              {\c!compareprocessaction[####1][##1]}%
            \processcommalist[#3]\c!dodoprocessaction}%
         \processcommalist[#1]{\getvalue{\s!do\the\processlevel}}%
         \advance\processlevel by -1\relax}}
```

The gain of speed in the final implementation is around 20%, depending on the application.

77
```
\newcount\processlevel
```

78
```
\def\v!compareprocessactionA[#1=>#2]%
  {\edef\!!stringb{#1}%
   \ifx\!!stringb\s!default
     #2%
   \fi}
```

79
```
\def\v!compareprocessactionB[#1=>#2]%
  {\expandedaction\!!stringb{#1}%
   \ifx\!!stringa\!!stringb
     \def\commalistelement{#1}%
     #2%
     \let\p!doprocessaction=\gobbleoneargument
   \else
     \edef\!!stringb{#1}%
     \ifx\!!stringb\s!unknown
       \def\commalistelement{#1}%
       #2%
```

syst-tex
syst-gen
syst-ext
syst-new

```
      \fi
    \fi}
```

80
```
\def\processaction[#1]#2[#3]%
  {\let\donottest=\dontprocesstest
   \expandedaction\!!stringa{#1}%
   \let\donottest=\doprocesstest
   \ifx\!!stringa\empty
     \let\v!compareprocessaction=\v!compareprocessactionA
   \else
     \let\v!compareprocessaction=\v!compareprocessactionB
   \fi
   \def\p!doprocessaction##1%
     {\v!compareprocessaction[##1]}%
   \processcommalist[#3]\p!doprocessaction
   \expandactions}
```

81
```
\def\v!compareprocessactionC[#1=>#2][#3]%
  {\expandedaction\!!stringa{#1}%
   \expandedaction\!!stringb{#3}%
   \ifx\!!stringa\!!stringb
     \def\commalistelement{#3}%
     #2%
     \let\p!doprocessaction=\gobbleoneargument
     \let\p!dodoprocessaction=\gobbleoneargument
   \else
     \edef\!!stringa{#1}%
     \ifx\!!stringa\s!unknown
       \def\commalistelement{#3}%
       #2%
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
    \fi
  \fi}
```

82
```
\def\processfirstactioninset[#1]#2[#3]%
  {\expandedaction\!!stringa{#1}%
   \ifx\!!stringa\empty
     \processaction[][#3]%
   \else
     \def\p!doprocessaction##1%
       {\def\p!dodoprocessaction####1%
          {\v!compareprocessactionC[####1][##1]}%
        \processcommalist[#3]\p!dodoprocessaction}%
     \processcommalist[#1]\p!doprocessaction
   \fi
   \expandactions}
```

83
```
\def\v!compareprocessactionD[#1=>#2][#3]%
  {\expandedaction\!!stringa{#1}%
   \expandedaction\!!stringb{#3}%
   \ifx\!!stringa\!!stringb
     \def\commalistelement{#3}%
     #2%
     \let\p!dodoprocessaction=\gobbleoneargument
   \else
     \edef\!!stringa{#1}%
     \ifx\!!stringa\s!unknown
       \def\commalistelement{#3}%
       #2%
     \fi
   \fi}
```

syst-tex
syst-gen
syst-ext
syst-new

84
```
\def\processallactionsinset[#1]#2[#3]%
  {\expandedaction\!!stringa{#1}%
   \ifx\!!stringa\empty
     \processaction[][#3]%
   \else
     \advance\processlevel by 1\relax
     \setvalue{\s!do\the\processlevel}##1%
       {\def\p!dodoprocessaction####1%
          {\v!compareprocessactionD[####1][##1]}%
        \processcommalist[#3]\p!dodoprocessaction}%
     \processcommalist[#1]{\getvalue{\s!do\the\processlevel}}%
     \advance\processlevel by -1\relax
   \fi
   \expandactions}
```

\unexpandedprocessaction
\unexpandedprocessfirs..
\unexpandedprocessalla..

Now what are those expansion commands doing there. Well, sometimes we want to compare actions that may consist off commands (i.e. are no constants). In such occasions we can use the a bit slower alternatives:

85
```
\def\unexpandedprocessfirstactioninset{\dontexpandactions\processfirstactioninset}
\def\unexpandedprocessaction          {\dontexpandactions\processaction}
\def\unexpandedprocessallactionsinset {\dontexpandactions\processallactionsinset}
```

By default we expand actions:

86
```
\def\expandactions%
  {\let\expandedaction=\edef}
```

87
```
\expandactions
```

But when needed we convert the strings to meaningful sequences of characters.

contents register context syst mult supp lang font colo spec core cont m s exit go back

syst-tex
syst-gen
syst-ext
syst-new

88
```
\def\unexpandedaction#1>{}
```

89
```
\def\noexpandedaction#1#2%
  {\def\convertedargument{#2}%
   \@EA\edef\@EA#1\@EA{\@EA\unexpandedaction\meaning\convertedargument}}
```

90
```
\def\dontexpandactions%
  {\let\expandedaction=\noexpandedaction}
```

\getfirstcharacter
\firstcharacter

Sometimes the action to be undertaken depends on the next character. This macro get this character and puts it in `\firstcharacter`.

```
   \getfirstcharacter {string}
```

A two step expansion is used to prevent problems with complicated arguments, for instance arguments that consist of two or more expandable tokens.

91
```
\def\dogetfirstcharacter#1#2\\%
  {\def\firstcharacter{#1}}
```

92
```
\def\getfirstcharacter#1%
  {\edef\!!stringa{#1}%
   \expandafter\dogetfirstcharacter\!!stringa\\}
```

\doifinstringelse

We can check for the presence of a substring in a given sequence of characters.

```
   \doifinsetelse {substring} {string} {then ...} {else ...}
```

An application of this command can be found further on. Like before, we first show some alternatives, like the one we started with:

```
   \long\def\p!doifinstringelse#1#2#3#4%
     {\def\c!doifinstringelse##1#1##2##3\war%
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
         {\if##2@%
            #4%
          \else
            #3%
          \fi}%
       \c!doifinstringelse#2#1@@\war}

   \def\doifinstringelse%
     {\ExpandBothAfter\p!doifinstringelse}
```

After this we came to:

```
   \def\p!doifinstringelse#1#2%
     {\def\c!doifinstringelse##1#1##2##3\war%
        {\if##2@}%
      \c!doifinstringelse#2#1@@\war}

   \def\doifinstringelse#1#2#3#4%
     {\ExpandBothAfter\p!doifinstringelse{#1}{#2}%
        #4%
      \else
        #3%
      \fi}
```

And finaly it became:

93
```
\def\v!ifinstringelse#1#2%
  {\def\c!ifinstringelse##1#1##2##3\war%
     {\csname\if##2@iffalse\else iftrue\fi\endcsname}%
   \c!ifinstringelse#2#1@@\war}
```

syst-tex
syst-gen
syst-ext
syst-new

94
```
\def\ifinstringelse#1#2%
  {\expanded{\v!ifinstringelse{#1}{#2}}}
```

95
```
\long\def\doifinstringelse#1#2#3#4%
  {\ifinstringelse{#1}{#2}%
     #3%
   \else
     #4%
   \fi}
```

\doifnumberelse

The next macro executes a command depending of the outcome of a test on numerals. This is probably one of the fastest test possible, exept from a less robust 10–step `\if`–ladder or some tricky `\lcode` checking.

```
   \doifnumberelse {string} {then ...} {else ...}
```

The macro accepts `123`, `abc`, `{}`, `\getal` and `\the\count...`.

96
```
\long\def\doifnumberelse#1#2#3%
  {\getfirstcharacter{#1}%
   \@EA\ifinstringelse\firstcharacter{1234567890}%
     #2%
   \else
     #3%
   \fi}
```

Before we had `\ifinstringelse` available, we used:

```
   \def\doifnumberelse#1%
     {\getfirstcharacter{#1}%
      \rawdoifinsetelse{\firstcharacter}{1,2,3,4,5,6,7,8,9,0}}
```

syst-tex
syst-gen
syst-ext
syst-new

A faster but less fail safe alternative is:

```
\dostepwiserecurse{0}{9}{1}
  {\@EA\uccode\@EA`\recurselevel=1}

\long\def\doifnumberelse#1#2#3%
  {\getfirstcharacter{#1}%
   \@EA\ifnum\@EA\uccode\@EA`\firstcharacter=1
     #2%
   \else
     #3%
   \fi}
```

This one only works when the `\firstcharacter` is indeed a character. Numbers and strings of characters go all right, but arguments like `\relax` let things go wrong.

\makerawcommalist
\rawdoinsetelse
\rawprocesscommalist
\rawprocessaction

Some of the commands mentioned earlier are effective but slow. When one is desperately in need of faster alternatives and when the conditions are predictable safe, the `\raw` alternatives come into focus. A major drawback is that they do not take `\c!constants` into account, simply because no expansion is done. This is no problem with `\rawprocesscommalist`, because this macro does not compare anything. Expandable macros are permitted as search string.

```
\makerawcommalist[string,string,...]\stringlist
\rawdoifinsetelse{string}{string,...}{...}{...}
\rawprocesscommalist[string,string,...]\commando
\rawprocessaction[x][a=>\a,b=>\b,c=>\c]
```

Spaces embedded in the list, for instance after commas, spoil the search process. The gain in speed depends on the length of the argument (the longer the argument, the less we gain).

contents register context syst mult supp lang font colo spec core cont m s exit go back

syst-tex
syst-gen
syst-ext
syst-new

97
```
\def\makerawcommalist[#1]#2%
  {\def\appendtocommalist##1%
     {\doifelse{#2}{}
        {\edef#2{##1}}
        {\edef#2{#2,##1}}}%
   \def#2{}%
   \processcommalist[#1]\appendtocommalist}
```

98
```
\def\rawprocesscommaitem#1,%
  {\if]#1\else
     \csname\s!next\the\commalevel\endcsname{#1}%
     \expandafter\rawprocesscommaitem
   \fi}
```

99
```
\def\rawprocesscommalist[#1]#2%
  {\advance\commalevel by 1\relax
   \expandafter\let\csname\s!next\the\commalevel\endcsname=#2%
   \expandafter\rawprocesscommaitem#1,],\relax
   \advance\commalevel by -1\relax}
```

100
```
\def\rawdoifinsetelse#1#2%
  {\doifinstringelse{,#1,}{,#2,}}
```

101
```
\def\v!rawprocessaction[#1][#2]%
  {\def\c!rawprocessaction##1,#1=>##2,##3\war%
     {\if##3@\else
        \def\v!processaction{##2}%
      \fi}%
   \c!rawprocessaction,#2,#1=>,@\war}
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

102
```
\def\rawprocessaction[#1]#2[#3]%
  {\edef\!!stringa{#1}%
   \edef\!!stringb{undefined}%
   \let\v!processaction=\!!stringb
   \ifx\!!stringa\empty
     \@EA\v!rawprocessaction\@EA[\s!default][#3]%
   \else
      \expandafter\v!rawprocessaction\expandafter[\!!stringa][#3]%
      \ifx\v!processaction\!!stringb
        \@EA\v!rawprocessaction\@EA[\s!unknown][#3]%
      \fi
   \fi
   \ifx\v!processaction\!!stringb
   \else
     \v!processaction
   \fi}
```

When we process the list `a,b,c,d,e`, the raw routine takes over 30% less time, when we feed 20+ character strings we gain about 20%. Alternatives which use `\futurelet` perform worse. Part of the speedup is due to the `\let` and `\expandafter` in the test.

\processunexpandedcomm..

When processing commalists, the arguments are expanded. The main reason for doing so lays in the fact that these macros are used for interfacing. The next alternative can be used for

```
   \processunexpandedcommalist
     [\alfa\beta,\gamma,\delta\epsilon]
     \handleitem
```

This time nesting is not supported.

103
```
\def\processunexpandedcommaitem#1,%
  {\if]\noexpand#1%
     \let\nextcommaitem=\relax
   \else
     \handleunexpandedcommaitem{#1}%
     \let\nextcommaitem=\processunexpandedcommaitem
   \fi
   \nextcommaitem}
```

104
```
\def\processunexpandedcommalist[#1]#2%
  {\def\handleunexpandedcommaitem{#2}%
   \processunexpandedcommaitem#1,],\relax}
```

Or faster:

105
```
\def\processunexpandedcommaitem#1,%
  {\if]\noexpand#1\else
     \handleunexpandedcommaitem{#1}%
     \expandafter\processunexpandedcommaitem
   \fi}
```

\dosetvalue
\dosetevalue
\docopyvalue
\doresetvalue
\dogetvalue

When we are going to do assignments, we have to take multi–linguality into account. For the moment we keep things simple and single–lingual.

```
   \dosetvalue   {label}    {variable}   {value}
   \dosetevalue  {label}    {variable}   {value}
   \docopyvalue  {to label} {from label} {variable}
   \doresetvalue {label}    {variable}
```

These macros are in fact auxiliary ones and are not meant for use outside the assignment macros.

syst-tex
syst-gen
syst-ext
syst-new

106
```
\def\dosetvalue#1#2% #3
  {\@EA\def\csname#1#2\endcsname} % {#3}}
```

107
```
\def\dosetevalue#1#2% #3
  {\@EA\edef\csname#1#2\endcsname} % {#3}}
```

108
```
\def\doresetvalue#1#2%
  {\@EA\def\csname#1#2\endcsname{}}
```

109
```
\def\docopyvalue#1#2#3%
  {\@EA\def\csname#1#3\endcsname{\csname#2#3\endcsname}}
```

\doassign
\undoassign
\doassignempty

Assignments are the backbone of CONTEXT. Abhorred by the concept of style file hacking, we took a considerable effort in building a parameterized system. Unfortunately there is a price to pay in terms of speed. Compared to other packages and taking the functionality of CONTEXT into account, the total size of the format file is still very acceptable. Now how are these assignments done.

Assignments can be realized with:

```
\doassign[label][variable=value]
\undoassign[label][variable=value]
```

and:

```
\doassignempty[label][variable=value]
```

Assignments like `\doassign` are compatible with:

```
\def\labelvariable{value}
```

We do check for the presence of an `=` and loudly complain of it's missed. We will redefine this macro later on, when a more advanced message mechanism is implemented.

contents register context syst mult supp lang font colo spec core cont m s exit go back

syst-tex
syst-gen
syst-ext
syst-new

110
```
\def\p!doassign#1[#2][#3=#4=#5]%
  {\ifx\empty#3\else  % and definitely not \ifx#3\empty
     \ifx\relax#5%
     \writestatus
       {setup}
       {missing '=' after '#3' in line \the\inputlineno}%
     \else
       #1{#2}{#3}{#4}%
     \fi
   \fi}
```

111
```
\def\doassign[#1][#2]%
  {\p!doassign\dosetvalue[#1][#2==\relax]}
```

112
```
\def\doeassign[#1][#2]%
  {\p!doassign\dosetevalue[#1][#2==\relax]}
```

113
```
\def\undoassign[#1][#2]%
  {\p!doassign\doresetvalue[#1][#2==\relax]}
```

114
```
\def\doassignempty[#1][#2=#3]%
  {\doifundefined{#1#2}
     {\dosetvalue{#1}{#2}{#3}}}
```

\getparameters
\geteparameters
\forgetparameters

Using the assignment commands directly is not our ideal of user friendly interfacing, so we take some further steps.

```
\getparameters    [label] [...=...,...=...]
\forgetparameters [label] [...=...,...=...]
```

Again, the label identifies the category a variable belongs to. The second argument can be a comma separated list of assignments.

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
\getparameters
  [demo]
  [alfa=1,
   beta=2]
```

is equivalent to

```
\def\demoalfa{1}
\def\demobeta{2}
```

In the pre–multi–lingual stadium CONTEXT took the next approach. With

```
\def\??demo {@@demo}
\def\!!alfa {alfa}
\def\!!beta {beta}
```

calling

```
\getparameters
  [\??demo]
  [\!!alfa=1,
   \!!beta=2]
```

lead to:

```
\def\@@demoalfa{1}
\def\@@demobeta{2}
```

Because we want to be able to distinguish the `!!` pre–tagged user supplied variables from internal counterparts, we will introduce a slightly different tag in the multi–lingual modules. There we will use `c!` or `v!`, depending on the context.

By calling `\p!doassign` directly, we save ourselves some argument passing and gain some speed. Whatever optimizations we do, this command will always be one of the bigger bottlenecks.

The alternative `\geteparameters` — it's funny to see that this alternative saw the light so lately — can be used to do expanded assigments.

115
```
\def\dogetparameters#1[#2]#3[#4]%
  {\def\p!dogetparameter##1%
     {\p!doassign#1[#2][##1==\relax]}%
   \processcommalist[#4]\p!dogetparameter}
```

116
```
\def\getparameters%
  {\dogetparameters\dosetvalue}
```

117
```
\def\geteparameters%
  {\dogetparameters\dosetevalue}
```

118
```
\def\forgetparameters%
  {\dogetparameters\doresetvalue}
```

119
```
\let\getexpandedparameters=\geteparameters
```

\getemptyparameters

Sometimes we explicitly want variables to default to an empty string, so we welcome:

```
   \getemptyparameters [label] [...=...,...=...]
```

120
```
\def\getemptyparameters[#1]#2[#3]%
  {\def\p!dogetemptyparameter##1%
     {\doassignempty[#1][##1]}%
   \processcommalist[#3]\p!dogetemptyparameter}
```

\copyparameters

Some CONTEXT commands take their default setups from others. All commands that are able to provide backgounds or rules around some content, for instance default to the standard command for ruled boxes. Is situations like this we can use:

```
\copyparameters [to-label] [from-label] [name1,name2,...]
```

For instance

```
\copyparameters
  [internal][external]
  [alfa,beta]
```

Leads to:

```
\def\internalalfa {\externalalfa}
\def\internalbeta {\externalbeta}
```

By using `\docopyvalue` we've prepared this command for use in a multi–lingual environment.

121
```
\def\copyparameters[#1]#2[#3]#4[#5]%
  {\doifnot{#1}{#3}
     {\def\docopyparameter##1%
        {\docopyvalue{#1}{#3}{##1}}%
      \processcommalist[#5]\docopyparameter}}
```

\doifassignmentelse

A lot of CONTEXT commands take optional arguments, for instance:

```
\dothisorthat[alfa,beta]
\dothisorthat[first=foo,second=bar]
\dothisorthat[alfa,beta][first=foo,second=bar]
```

Although a combined solution is possible, we prefer a seperation. The next command takes care of propper handling of such multi–faced commands.

syst-tex
syst-gen
syst-ext
syst-new

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
\doifassignmentelse {...} {then ...} {else ...}
```

122
```
\def\doifassignmentelse%
  {\doifinstringelse{=}}
```

\ifparameters
\checkparameters

A slightly different one is `\checkparameters`, which also checks on the presence of a `=`.

The boolean `\ifparameters` can be used afterwards. Combining both in one `\if`–macro would lead to problems with nested `\if`'s.

```
\checkparameters[argument]
```

123
```
\newif\ifparameters
```

124
```
\def\c!checkparameters#1=#2#3\war%
  {\if#2@\parametersfalse\else\parameterstrue\fi}
```

125
```
\def\checkparameters[#1]%
  {\c!checkparameters#1=@@\war}
```

\getfromcommalist
\getfromcommacommand
\commalistelement
\getcommalistsize
\getcommacommandsize

It's possible to get an element from a commalist or a command representing a commalist.

```
\getfromcommalist    [string] [n]
\getfromcommacommand [string,\strings,string,...] [n]
```

The difference betwee the two of them is the same as the difference between `\processcomma...`. The found string is stored in `\commalistelement`.

We can calculate the size of a comma separated list by using:

```
\getcommalistsize    [string,string,...]
\getcommacommandsize [string,\strings,string,...]
```

Afterwards, the length is available in the macro `\commalistsize` (not a ⟨*counter*⟩).

126
```
\def\commalistsize{0}
```

127
```
\def\p!dogetcommalistsize#1[#2]%
  {\scratchcounter=0\relax
   \def\p!dodogetcommalistsize##1%
     {\advance\scratchcounter by 1\relax}%
   #1[#2]\p!dodogetcommalistsize   % was [{#2}]
   \edef\commalistsize{\the\scratchcounter}}
```

128
```
\def\getcommalistsize%
  {\p!dogetcommalistsize\processcommalist}
```

129
```
\def\getcommacommandsize%
  {\p!dogetcommalistsize\processcommacommand}
```

130
```
\def\p!dodogetfromcommalist#1%
  {\advance\scratchcounter by -1\relax
   \ifnum\scratchcounter=0\relax
     \gdef\globalcommalistelement{#1}%
     \def\doprocesscommaitem##1]{}%
   \fi}
```

131
```
\def\p!dogetfromcommalist#1[#2]#3[#4]%
  {\global\let\globalcommalistelement=\empty
   \bgroup
   \scratchcounter=#4\relax
   #1[#2]\p!dodogetfromcommalist
   \egroup
   \let\commalistelement=\globalcommalistelement}
```

132
```
\def\getfromcommalist%
  {\p!dogetfromcommalist\processcommalist}
```

133
```
\def\getfromcommacommand%
  {\p!dogetfromcommalist\processcommacommand}
```

Watertight (and efficient) solutions are hard to find, due to the handling of braces during parameters passing and scanning. Nevertheless:

```
\def\dosomething#1{(#1=\commalistsize) }

\getcommalistsize [\hbox{$a,b,c,d,e,f$}] \dosomething 1
\getcommalistsize [{a,b,c,d,e,f}]        \dosomething 1
\getcommalistsize [{a,b,c},d,e,f]        \dosomething 4
\getcommalistsize [a,b,{c,d,e},f]        \dosomething 4
\getcommalistsize [a{b,c},d,e,f]         \dosomething 4
\getcommalistsize [{a,b}c,d,e,f]         \dosomething 4
\getcommalistsize []                     \dosomething 0
\getcommalistsize [{[}]                  \dosomething 1
```

reports:

(1=1) (1=6) (4=4) (4=4) (4=4) (4=4) (0=0) (1=1)

\dosingleargument
\dodoubleargument
\dotripleargument
\doquadrupleargument
\doquintupleargument
\dosixtupleargument

When working with delimited arguments, spaces and lineendings can interfere. The next set of macros uses TEX' internal scanner for grabbing everything between arguments.

```
\dosingleargument\commando    = \commando[#1]
\dodoubleargument\commando    = \commando[#1][#2]
\dotripleargument\commando    = \commando[#1][#2][#3]
\doquadrupleargument\commando = \commando[#1][#2][#3][#4]
\doquintupleargument\commando = \commando[#1][#2][#3][#4][#5]
\dosixtupleargument\commando  = \commando[#1][#2][#3][#4][#5][#6]
```

These macros are used in the following way:

```
\def\dosetupsomething[#1][#2]%
  {... #1 ... #2 ...}

\def\setupsomething%
  {\dodoubleargument\dosetupsomething}
```

The implementation can be surprisingly simple and needs no further explanation, like:

```
\def\dosingleargument#1[#2]%
  {#1[#2]}
\def\dotripleargument#1[#2]#3[#4]#5[#6]%
  {#1[#2][#4][#6]}
\def\doquintupleargument#1%
  {\def\dodoquintupleargument[##1]##2[##3]##4[##5]##6[##7]##8[##9]%
     {#1[##1][##3][##5][##7][##9]}%
   \dodoquintupleargument}
```

Because TEX accepts 9 arguments at most, we have to use two–step solution when getting five or more arguments.

When developing more and more of the real CONTEXT, we started using some alternatives that provided empty arguments (in fact optional ones) whenever the user failed to supply them. Because this more complicated macros enable us to do some checking, we reimplemented the non–empty ones.

134
```
\def\dosingleargument%
  {\def\expectedarguments{1}%
   \dosingleempty}
```

135
```
\def\dodoubleargument%
  {\def\expectedarguments{2}%
   \dodoubleempty}
```

136
```
\def\dotripleargument%
  {\def\expectedarguments{3}%
   \dotripleempty}
```

137
```
\def\doquadrupleargument%
  {\def\expectedarguments{4}%
   \doquadrupleempty}
```

138
```
\def\doquintupleargument%
  {\def\expectedarguments{5}%
   \doquintupleempty}
```

139
```
\def\doquintupleargument%
  {\def\expectedarguments{6}%
   \dosixtupleempty}
```

syst-tex
syst-gen
syst-ext
syst-new

\iffirstargument
\ifsecondargument
\ifthirdargument
\iffourthargument
\iffifthargument
\ifsixthargument

We use some signals for telling the calling macros if all wanted arguments are indeed supplied by the user.

140
```
\newif\iffirstargument
\newif\ifsecondargument
\newif\ifthirdargument
\newif\iffourthargument
\newif\iffifthargument
\newif\ifsixthargument
```

\dosingleempty
\dodoubleempty
\dotripleempty
\doquadrupleempty
\doquintupleempty

The empty argument supplying macros mentioned before, look like:

```
   \dosingleempty    \command
   \dodoubleempty    \command
   \dotripleempty    \command
   \doquadrupleempty \command
   \doquintupleempty \command
   \dosixtupleempty  \command
```

So `\dodoubleempty` leades to:

```
   \command[#1][#2]
   \command[#1][]
   \command[][]
```

Depending of the generousity of the user. Afterwards one can use the `\if...argument` boolean. For novice: watch the stepwise doubling of `#`'s

141
```
\def\noexpectedarguments {0}
\def\expectedarguments   {0}
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

142
```
\def\dogetargument#1#2#3#4%
  {\doifnextcharelse{#1}
     {\let\expectedarguments=\noexpectedarguments
      #3\dodogetargument}
     {\ifnum\expectedarguments>\noexpectedarguments
        \writestatus
          {setup}
          {\expectedarguments\space argument(s) expected
           in line \the\inputlineno\space}%
      \fi
      \let\expectedarguments=\noexpectedarguments
      #4\dodogetargument#1#2}}
```

143
```
\def\getsingleempty#1#2#3%
  {\def\dodogetargument%
     {#3}%
   \dogetargument#1#2\firstargumenttrue\firstargumentfalse}
```

144
```
\def\getdoubleempty#1#2#3%
  {\def\dodogetargument#1##1#2%
     {\def\dodogetargument%
        {#3#1##1#2}%
      \dogetargument#1#2\secondargumenttrue\secondargumentfalse}%
   \dogetargument#1#2\firstargumenttrue\firstargumentfalse}
```

145
```
\def\gettripleempty#1#2#3%
  {\def\dodogetargument#1##1#2%
     {\def\dodogetargument#1####1#2%
        {\def\dodogetargument%
           {#3#1##1#2#1####1#2}%
```

syst-tex
syst-gen
syst-ext
syst-new

contents register context syst mult supp lang font colo spec core cont m s exit go back

syst-tex
syst-gen
syst-ext
syst-new

```
         \dogetargument#1#2\thirdargumenttrue\thirdargumentfalse}%
      \dogetargument#1#2\secondargumenttrue\secondargumentfalse}%
   \dogetargument#1#2\firstargumenttrue\firstargumentfalse}
```

146
```
\def\getquadrupleempty#1#2#3%
  {\def\dodogetargument#1##1#2%
     {\def\dodogetargument#1####1#2%
        {\def\dodogetargument#1########1#2%
           {\def\dodogetargument%
              {#3#1##1#2#1####1#2#1########1#2}%
            \dogetargument#1#2\fourthargumenttrue\fourthargumentfalse}%
         \dogetargument#1#2\thirdargumenttrue\thirdargumentfalse}%
      \dogetargument#1#2\secondargumenttrue\secondargumentfalse}%
   \dogetargument#1#2\firstargumenttrue\firstargumentfalse}
```

147
```
\def\getquintupleempty#1#2#3%
  {\def\dodogetargument#1##1#2%
     {\def\dodogetargument#1####1#2%
        {\def\dodogetargument#1########1#2%
           {\def\dodogetargument#1################1#2%
              {\def\dodogetargument%
                 {#3#1##1#2#1####1#2#1########1#2#1################1#2}%
               \dogetargument#1#2\fifthargumenttrue\fifthargumentfalse}%
            \dogetargument#1#2\fourthargumenttrue\fourthargumentfalse}%
         \dogetargument#1#2\thirdargumenttrue\thirdargumentfalse}%
      \dogetargument#1#2\secondargumenttrue\secondargumentfalse}%
   \dogetargument#1#2\firstargumenttrue\firstargumentfalse}
```

148
```
\def\getsixtupleempty#1#2#3%
  {\def\dodogetargument#1##1#2%
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

syst-tex
syst-gen
syst-ext
syst-new

```
  {\def\dodogetargument#1####1#2%
     {\def\dodogetargument#1########1#2%
        {\def\dodogetargument#1################1#2%
           {\def\dodogetargument#1################################1#2%
              {\def\dodogetargument%
                 {#3#1##1#2#1####1#2#1########1#2#1################1%
                         #2#1################################1#2}%
              \dogetargument#1#2\sixthargumenttrue\sixthargumentfalse}%
            \dogetargument#1#2\fifthargumenttrue\fifthargumentfalse}%
         \dogetargument#1#2\fourthargumenttrue\fourthargumentfalse}%
      \dogetargument#1#2\thirdargumenttrue\thirdargumentfalse}%
   \dogetargument#1#2\secondargumenttrue\secondargumentfalse}%
\dogetargument#1#2\firstargumenttrue\firstargumentfalse}
```

149
```
\def\dosingleempty    {\getsingleempty    []}
\def\dodoubleempty    {\getdoubleempty    []}
\def\dotripleempty    {\gettripleempty    []}
\def\doquadrupleempty {\getquadrupleempty []}
\def\doquintupleempty {\getquintupleempty []}
\def\dosixtupleempty  {\getsixtupleempty  []}
```

\dosingleargumentwithset
\dodoubleargumentwithset
\dodoubleemptywithset
\dotripleargumentwithset
\dotripleemptywithset

These maybe too mysterious macros enable us to handle more than one setup at once.

```
\dosingleargumentwithset \command[#1]
\dodoubleargumentwithset \command[#1] [#2]
\dotripleargumentwithset \command[#1] [#2] [#3]
\dodoubleemptywithset    \command[#1] [#2]
\dotripleemptywithset    \command[#1] [#2] [#3]
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

syst-tex
syst-gen
syst-ext
syst-new

The first macro calls `\command[##1]` for each string in the set `#1`. The second one calls for `\commando[##1][#2]` and the third, well one may guess. These commands support constructions like:

```
\def\dodefinesomething[#1][#2]%
  {\getparameters[\??xx#1][#2]}

\def\definesomething%
  {\dodoubleargumentwithset\dodefinesomething}
```

Which accepts calls like:

```
\definesomething[alfa,beta,...][variable=...,...]
```

Now a whole bunch of variables like `\@@xxalfavariable` and `\@@xxbetavariable` is defined.

150
```
\def\dosingleargumentwithset#1%
  {\def\dodosinglewithset[##1]%
     {\def\dododosinglewithset####1%
        {#1[####1]}%
      \processcommalist[##1]\dododosinglewithset}%
   \dosingleargument\dodosinglewithset}%
```

151
```
\def\dodoublewithset#1#2%
  {\def\dododoublewithset[##1][##2]%
     {\doifnot{##1}{}
        {\def\dodododoublewithset####1%
           {#2[####1][##2]}%
         \processcommalist[##1]\dodododoublewithset}}%
   #1\dododoublewithset}%
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

152
```
\def\dodoubleemptywithset%
  {\dodoublewithset\dodoubleempty}
```

153
```
\def\dodoubleargumentwithset%
  {\dodoublewithset\dodoubleargument}
```

154
```
\def\dotriplewithset#1#2%
  {\def\dodotriplewithset[##1][##2][##3]%
     {\doifnot{##1}{}
        {\def\dododotriplewithset####1%
           {#2[####1][##2][##3]}%
         \processcommalist[##1]\dododotriplewithset}}%
   #1\dodotriplewithset}%
```

155
```
\def\dotripleemptywithset%
  {\dotriplewithset\dotripleempty}
```

156
```
\def\dotripleargumentwithset%
  {\dotriplewithset\dotripleargument}
```

\complexorsimple
\complexorsimpleempty

Setups can be optional. A command expecting a setup is prefixed by `\complex`, a command without one gets the prefix `\simple`. Commands like this can be defined by:

```
   \complexorsimple {command}
```

When `\command` is followed by a `[setup]`, then

```
   \complexcommand [setup]
```

executes, else we get

```
   \simplecommand
```

syst-tex
syst-gen
syst-ext
syst-new

An alternative for `\complexorsimple` is:

```
\complexorsimpleempty {command}
```

Depending on the presence of `[setup]`, this one leads to one of:

```
\complexcommando [setup]
\complexcommando []
```

Many CONTEXT commands started as complex or simple ones, but changed into more versatile (more object oriented) ones using the `\get..argument` commands.

157
```
\def\complexorsimple#1%
  {\doifnextcharelse{[}
     {\firstargumenttrue\getvalue{\s!complex#1}}
     {\firstargumentfalse\getvalue{\s!simple#1}}}
```

158
```
\def\complexorsimpleempty#1%
  {\doifnextcharelse{[}
     {\firstargumenttrue\getvalue{\s!complex#1}}
     {\firstargumentfalse\getvalue{\s!complex#1}[]}}
```

`\definecomplexorsimple`
`\definecomplexorsimple..`

The previous commands are used that often that we found it worthwile to offer two more alternatives.

159
```
\def\setnameofcommand#1%
  {\bgroup
   \escapechar=-1\relax
   \xdef\nameofcommand{\string#1}%
   \egroup}
```

160
```
\def\definewithnameofcommand#1#2% watch the \donottest
  {\setnameofcommand{#2}%
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
   \@EA\def\@EA#2\@EA{\@EA\donottest\@EA#1\@EA{\nameofcommand}}}
```

161
```
\def\definecomplexorsimple%
  {\definewithnameofcommand\complexorsimple}
```

162
```
\def\definecomplexorsimpleempty%
  {\definewithnameofcommand\complexorsimpleempty}
```

These commands are called as:

```
\definecomplexorsimple\command
```

Of course, we must have available

```
\def\complexcommand[#1]{...}
\def\simplecommand     {...}
```

Using this construction saves a few string now and then.

\definestartstopcommand

Those who get the creeps of expansion may skip the next one. It's one of the most recent additions and concerns `\start`–`\stop` pairs with complicated arguments.

We won't go into details here, but the general form of this using this command is:

```
\definestartstopcommand\somecommand\v!specifier{arg}{arg}%
  {do something with arg}
```

This expands to something like:

```
\def\somecommand arg \startspecifier arg \stopspecifier%
  {do something with arg}
```

The argumentss can be anything reasonable, but double #'s are needed in the specification part, like:

syst-tex
syst-gen
syst-ext
syst-new

```
\definestartstopcommand\somecommand\v!specifier{[##1][##2]}{##3}%
  {do #1 something #2 with #3 arg}
```

which becomes:

```
\def\somecommand[#1][#2]\startspecifier#3\stopspecifier%
  {do #1 something #2 with #3 arg}
```

We will see some real applications of this command in the core modules.

163

```
\def\definestartstopcommand#1#2#3#4%
  {\def\!stringa{#3}%
   \def\!stringb{\e!start#2}%
   \def\!stringc{#4}%
   \def\!stringd{\e!stop#2}%
   \@EA\@EA\@EA\@EA\@EA\@EA\@EA\@EA\@EA\@EA\@EA\@EA\@EA\@EA\@EA
   \def\@EA\@EA\@EA\@EA\@EA\@EA\@EA\@EA\@EA\@EA\@EA\@EA\@EA\@EA\@EA
   #1\@EA\@EA\@EA\@EA\@EA\@EA\@EA
   \!stringa\@EA\@EA\@EA
   \csname\@EA\@EA\@EA\!stringb\@EA\@EA\@EA\endcsname\@EA
   \!stringc
   \csname\!stringd\endcsname}
```

\dosinglegroupempty
\dodoublegroupempty
\dotriplegroupempty

We've already seen some commands that take care of optional arguments between `[]`. The next two commands handle the ones with `{}`. They are called as:

```
\dosinglegroupempty \IneedONEargument
\dodoublegroupempty \IneedTWOarguments
\dotriplegroupempty \IneedTREEarguments
```

where `\IneedONEargument` takes one and the others two and three arguments. These macro's were first needed in PPCHTEX.

164
```
\def\dogetgroupargument#1#2%
  {\def\nextnext%
     {\ifx\next\bgroup
        \let\expectedarguments=\noexpectedarguments
        \def\next{#1\dodogetargument}%
      \else\ifx\next\lineending
        \def\next{\bgroup\def\\ {\egroup\dogetgroupargument#1#2}\\}%
      \else\ifx\next\blankspace
        \def\next{\bgroup\def\\ {\egroup\dogetgroupargument#1#2}\\}%
      \else
        \ifnum\expectedarguments>\noexpectedarguments
          \writestatus
            {setup}
            {\expectedarguments\space argument(s) expected
             in line \the\inputlineno\space}%
        \fi
        \let\expectedarguments=\noexpectedarguments
        \def\next{#2\dodogetargument{}}%
      \fi\fi\fi
      \next}%
   \futurelet\next\nextnext}
```

165
```
\def\dosinglegroupempty#1%
  {\def\dodogetargument%
     {#1}%
   \dogetgroupargument\firstargumenttrue\firstargumentfalse}
```

166
```
\def\dodoublegroupempty#1%
  {\def\dodogetargument##1%
     {\def\dodogetargument%
```

syst-tex
syst-gen
syst-ext
syst-new

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
        {#1{##1}}%
      \dogetgroupargument\secondargumenttrue\secondargumentfalse}%
   \dogetgroupargument\firstargumenttrue\firstargumentfalse}
```

167
```
\def\dotriplegroupempty#1%
  {\def\dodogetargument##1%
     {\def\dodogetargument####1%
        {\def\dodogetargument%
           {#1{##1}{####1}}%
         \dogetgroupargument\thirdargumenttrue\thirdargumentfalse}%
      \dogetgroupargument\secondargumenttrue\secondargumentfalse}%
   \dogetgroupargument\firstargumenttrue\firstargumentfalse}
```

These macros explictly take care of spaces, which means that the next definition and calls are valid:

```
\def\test#1#2#3{[#1#2#3]}

\dotriplegroupempty\test {a}{b}{c}
\dotriplegroupempty\test {a}{b}
\dotriplegroupempty\test {a}
\dotriplegroupempty\test
\dotriplegroupempty\test {a} {b} {c}
\dotriplegroupempty\test {a} {b}
\dotriplegroupempty\test
  {a}
  {b}
```

And alike.

\wait The next macro hardly needs explanation. Because no nesting is to be expected, we can reuse `\wait` within `\wait` itself.

168
```
\def\wait%
  {\bgroup
   \read16 to \wait
   \egroup}
```

\writestring
\writeline
\writestatus
\statuswidth

Maybe one didn't notice, but we've already introduced a macro for showing messages. In the multi–lingual modules, we will also introduce a mechanism for message passing. For the moment we stick to the core macros:

```
   \writestring {string}
   \writeline
   \writestatus {category} {message}
```

Messages are formatted. One can provide the maximum with of the identification string with the macro `\statuswidth`.

169
```
\def\statuswidth {15}
```

170
```
\def\writestring%
  {\immediate\write16}
```

171
```
\def\writeline%
  {\writestring{}}
```

172
```
\def\dosplitstatus#1#2\end%
  {\ifx#1?%
     \loop
       \advance\scratchcounter by 1
       \ifnum\scratchcounter<\statuswidth\relax
```

```
        \edef\messagecontentA{\messagecontentA\space}%
      \repeat
    \else
      \advance\scratchcounter by 1
      \ifnum\scratchcounter<\statuswidth\relax
        \edef\messagecontentA{\messagecontentA#1}%
      \fi
      \dosplitstatus#2\end
    \fi}
```

173
```
\def\writestatus#1#2%
  {\bgroup
   \edef\messagecontentA{}%
   \edef\messagecontentB{#2}%  maybe it's \the\scratchcounter
   \scratchcounter=0
   \expandafter\dosplitstatus#1?\end
   \writestring{\messagecontentA\space:\space\messagecontentB}%
   \egroup}
```

\debuggerinfo For debugging purposes we can enhance macros with the next alternative. Here `debuggerinfo` stands for both a macro accepting two arguments and a boolean (in fact a few macro's too).

174
```
\newif\ifdebuggerinfo
```

175
```
\def\debuggerinfo#1#2%
  {\ifdebuggerinfo
     \writestatus{debugger}{#1:: #2}%
   \fi}
```

Finally we do what from now on will be done at the top of the files: we tell the user what we are loading.

176

```
\writestatus{loading}{Context System Macros / General}
```

Well, the real final command is the one that resets the unprotected characters `@`, `?` and `!`.

177

```
\protect
```

syst-tex
syst-gen
syst-ext
syst-new

syst-tex
syst-gen
syst-ext
syst-new

2.3 Extras

1
```
\writestatus{loading}{Context System Macro's / Extras}
```

In this second system module, we continue the definition of some handy commands.

2
```
\unprotect
```

\doglobal \redoglobal \dodoglobal

The two macros `\redoglobal` and `\dodoglobal` are used in this and some other modules to enforce a user specified `\doglobal` action. The last and often only global assignment in a macro is done with `\dodoglobal`, but all preceding ones with `\redoglobal`.

3
```
\let\dodoglobal=\relax
\let\redoglobal=\relax
```

4
```
\def\doglobal%
  {\let\redoglobal=\global
   \def\dodoglobal%
     {\let\redoglobal=\relax
      \let\dodoglobal=\relax
      \global}}
```

\newcounter \increment \decrement

Unfortunately the number of ⟨*counters*⟩ in TEX is limited, but fortunately we can store numbers in a macro. We can increment such pseudo ⟨*counters*⟩ with `\increment`.

```
   \increment(\counter,20)
   \increment(\counter,-4)
   \increment(\counter)
   \increment\counter
```

After this sequence of commands, the value of `\counter` is 20, 16, 17 and 18. Of course there is also the complementary command `\decrement`.

contents register context syst mult supp lang font colo spec core cont m s exit go back

syst-tex
syst-gen
syst-ext
syst-new

Global assignments are possible too, using `\doglobal`:

```
\doglobal\increment\counter
```

When `\counter` is undefined, it's value is initialized at 0. It is nevertheless better to define a ⟨*counter*⟩ explicitly. One reason could be that the ⟨*counter*⟩ can be part of a test with `\ifnum` and this conditional does not accept undefined macro's. The ⟨*counter*⟩ in our example can for instance be defined with:

```
\newcounter\counter
```

The command `\newcounter` must not be confused with `\newcount`! Of course this mechanism is much slower than using TEX's ⟨*counters*⟩ directly. In practice ⟨*counters*⟩ (and therefore our pseudo counters too) are seldom the bottleneck in the processing of a text. Apart from some other incompatilities we want to mention a pitfal when using `\ifnum`.

```
\ifnum\normalcounter=\pseudocounter \doif \else \doelse \fi
\ifnum\pseudocounter=\normalcounter \doif \else \doelse \fi
```

In the first test, TEX continues it's search for the second number after reading `\pseudocounter`, while in the second test, it stops reading after having encountered a real one. Tests like the first one therefore can give unexpected results, for instance execution of `\doif` even if both numbers are unequal.

5
```
\def\newcounter#1%
  {\dodoglobal\def#1{0}}
```

6
```
\def\dodododoincrement(#1,#2)%
  {\ifx#1\undefined
     \def#1{0}%
   \fi
   \scratchcounter=#2\relax
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

syst-tex
syst-gen
syst-ext
syst-new

```
   \scratchcounter=\incrementsign\scratchcounter
   \advance\scratchcounter by #1\relax
   \dodoglobal\edef#1{\the\scratchcounter}}
```

7
```
\def\dodododoincrement#1%
  {\dododoincrement(#1,1)}
```

8
```
\def\dodoincrement(#1%
  {\doifnextcharelse{,}
     {\dododoincrement(#1}
     {\dododoincrement(#1,1}}
```

9
```
\def\doincrement#1%
  {\def\incrementsign{#1}%
   \doifnextcharelse{(}
     {\dodoincrement}
     {\dododoincrement}}
```

10
```
\def\increment%
  {\doincrement+}
```

11
```
\def\decrement%
  {\doincrement-}
```

`\newsignal` When writing advanced macros, we cannot do without signaling. A signal is a small (invisible) kern or penalty that signals the next macro that something just happened. This macro can take any action depending onthe previous signal. Signals must be unique and the next macro takes care of that.

```
   \newsignal\somesignal
```

Signals old dimensions and can be used in skips, kerns and tests like `\ifdim`.

contents register context syst mult supp lang font colo spec core cont m s exit go back

12
```
\newdimen\currentsignal
```

13
```
\def\newsignal#1%
  {\advance\currentsignal by 0.00025pt
   \edef#1{\the\currentsignal}}
```

\newskimen

TEX offers 256 ⟨*dimensions*⟩ and ⟨*skips*⟩. Unfortunately this amount is too small to suit certain packages. Therfore when possible one should use

```
   \newskimen\tempskimen
```

This commands allocates a ⟨*dimension*⟩ or a ⟨*skip*⟩, depending on the availability. One should be aware of the difference between both. When searching for some glue TEX goes on searching till it's sure that no other glue component if found. This search can be canceled by using `\relax` when possible and needed.

14
```
\def\newskimen#1%
  {\ifx#1\undefined
     \ifnum\count11>\count12
       \alloc@2\skip \skipdef \insc@unt#1\relax
     \else
       \alloc@1\dimen\dimendef\insc@unt#1\relax
     \fi
   \fi}
```

\strippedcsname

The next macro can be very useful when using `\csname` like in:

```
   \csname if\strippedcsname\something\endcsname
```

This expands to `\ifsomething`.

15
```
\def\strippedcsname%
  {\expandafter\gobbleoneargument\string}
```

\newconditional
\settrue
\setfalse
\ifconditional

syst-tex
syst-gen
syst-ext
syst-new

TEX's lacks boolean variables, although the PLAIN format implements `\newif`. The main disadvantage of this scheme is that it takes three hash table entries. A more memory saving alternative is presented here. A conditional is defined by:

```
\newconditional\doublesided
\setfalse

Setting a conditional is done by \type{\settrue} and
\type{\setfalse}:

\starttypen
\settrue\doublesided
\setfalse

while testing is accomplished by:

\starttypen
\ifconditional\doublesided  ... \else ... \fi
\setfalse

We cannot use the simple scheme:

\starttypen
\def\settrue#1{\let#1=\iftrue}
\def\settrue#1{\let#1=\iffalse}
```

Such an implementation gives problems with nested conditionals. The next implementation is abaou as fast and just as straightforward:

contents register context syst mult supp lang font colo spec core cont m s exit go back

syst-tex
syst-gen
syst-ext
syst-new

16
```
\def\settrue#1%
  {\chardef#1=0 }
```

17
```
\def\setfalse#1%
  {\chardef#1=1 }
```

18
```
\let\newconditional = \setfalse
\let\ifconditional  = \ifcase
```

\dorecurse
\recurselevel
\recursedepth dostepwi..
\for

TEX does not offer us powerfull for–loop mechanisms. On the other hand its recursion engine is quite unique. We therefore identify the for–looping macros by this method. The most simple alternative is the one that only needs a number.

```
   \dorecurse {n} {whatever we want}
```

This macro can be nested without problems and therefore be used in situations where PLAIN TEX's `\loop` macro ungracefully fails. The current value of the counter is available in `\recurselevel`, before as well as after the `whatever we wat` stuff.

```
   \dorecurse                % inner loop
     {10}
     {\recurselevel:           % outer value
        \dorecurse           % inner loop
          {\recurselevel}      % outer value
          {\recurselevel}      % inner value
        \dorecurse           % inner loop
          {\recurselevel}      % outer value
          {\recurselevel}      % inner value
      \endgraf}
```

syst-tex
syst-gen
syst-ext
syst-new

In this example the first, second and fourth `\recurselevel` concern the outer loop, while the third and fifth one concern the inner loop. The depth of the nesting is available for inspection in `\recursedepth`.

Both `\recurselevel` and `\recursedepth` are macros. The real ⟨*counters*⟩ are hidden from the user because we don't want any interference.

19
```
\def\@@irecurse{@@irecurse}  % stepper
\def\@@nrecurse{@@nrecurse}  % number of steps
\def\@@srecurse{@@srecurse}  % step
\def\@@drecurse{@@drecurse}  % direction, < or >
\def\@@arecurse{@@arecurse}  % action
```

20
```
\newcount\outerrecurse
```

21
```
\def\recursedepth%
  {\the\outerrecurse}
```

22
```
\long\def\dostepwiserecurse#1#2#3#4%
  {\global\advance\outerrecurse by 1
   \scratchcounter=#1\setevalue{\@@irecurse\recursedepth}{\the\scratchcounter}%
   \scratchcounter=#2\setevalue{\@@nrecurse\recursedepth}{\the\scratchcounter}%
   \scratchcounter=#3\setevalue{\@@srecurse\recursedepth}{\the\scratchcounter}%
   \let\next=\donorecurse
   \ifnum#3>0\relax\ifnum#2<#1\relax
   \else
     \setevalue{\@@drecurse\recursedepth}{>}%
     \long\setvalue{\@@arecurse\recursedepth}{#4}%
     \let\next=\dodorecurse
   \fi\fi
   \ifnum#3<0\relax\ifnum#1<#2\relax
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

syst-tex
syst-gen
syst-ext
syst-new

```
   \else
     \setevalue{\@@drecurse\recursedepth}{<}%
     \long\setvalue{\@@arecurse\recursedepth}{#4}%
     \let\next=\dodorecurse
   \fi\fi
   \next}
```

23
```
\def\donorecurse%
  {}
```

24
```
\def\dodonorecurse%
  {\global\advance\outerrecurse by -1\relax}
```

25
```
\def\dododorecurse%
  {\edef\recurselevel{\getvalue{\@@irecurse\recursedepth}}%
   \getvalue{\@@arecurse\recursedepth}%
   \edef\recurselevel{\getvalue{\@@irecurse\recursedepth}}%
   \scratchcounter=\recurselevel
   \advance\scratchcounter by \getvalue{\@@srecurse\recursedepth}\relax
   \setevalue{\@@irecurse\recursedepth}{\the\scratchcounter}%
   \dodorecurse}
```

26
```
\def\dodorecurse%
  {\ifnum\getvalue{\@@irecurse\recursedepth}
         \getvalue{\@@drecurse\recursedepth}
         \getvalue{\@@nrecurse\recursedepth}\relax
     \let\next=\dodonorecurse
   \else
     \let\next=\dododorecurse
   \fi
   \next}
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

27
```
\def\dorecurse#1%
  {\dostepwiserecurse{1}{#1}{1}}
```

For those we like to offer visual beauty for efficiency we say however:[1]

28
```
\def\dodorecurse%
  {\ifnum\getvalue{\@@irecurse\recursedepth}
         \getvalue{\@@drecurse\recursedepth}
         \getvalue{\@@nrecurse\recursedepth}\relax
     \global\advance\outerrecurse by -1\relax
   \else
     \expandafter\dododorecurse
   \fi}
```

As we can see here, the simple command `\dorecurse` is a special case of the more general:

```
   \dostepwiserecurse {from} {to} {step} {action}
```

This commands accepts positive and negative steps. Illegal values are handles as good as possible and the macro accepts numbers and ⟨*counters*⟩.

```
   \dostepwiserecurse  {1} {10}  {2} {...}
   \dostepwiserecurse {10}  {1} {-2} {...}
```

The third alternative looks a bit different and uses a pseudo counter. When this macro is nested, we have to use different counters. This time we use keywords.

```
   \def\alfa{2} \def\beta{100} \def\gamma{3}

   \for \n=55    \to 100   \step  1      \do {... \n ...}
   \for \n=\alfa \to \beta \step  \gamma \do {... \n ...}
```

[1] In this kind of macro's we tend to minimalize the overhead.

```
\for \n=\n    \to 120   \step  1      \do {... \n ...}
\for \n=120   \to 100   \step -3      \do {... \n ...}
\for \n=55    \to 100   \step  2      \do {... \n ...}
```

Only in the third example we need to predefine `\n`. The use of `\od` as a dilimiter would have made nested use more problematic.

29

```
\def\for#1=#2\to#3\step#4\do#5%
  {\dostepwiserecurse{#2}{#3}{#4}
     {\edef#1{\recurselevel}%
      #5%
      \edef#1{\recurselevel}}}
```

`\doloop`
`\exitloop`

Sometimes loops are not determined by counters, but by (a combinations of) conditions. We therefore implement a straightforward loop, which can only be left when we explictly exit it. Nesting is supported. First we present a more extensive alternative.

```
\doloop
  {Some kind of typesetting punishment \par
   \ifnum\pageno>100 \exitloop \fi}
```

When needed, one can call for `\looplevel` and `\loopdepth`.

If we write this macros from scratch, we end up with something like the ones described above:

```
\def\@@eloop{@@eloop}  % exit
\def\@@iloop{@@iloop}  % stepper
\def\@@aloop{@@aloop}  % action

\newcount\outerloop

\def\loopdepth%
```

```
  {\the\outerloop}

\def\exitloop%
  {\setevalue{\@@eloop\loopdepth}{0}}

\long\def\doloop#1%
  {\global\advance\outerloop by 1
   \setevalue{\@@iloop\loopdepth}{1}%
   \setevalue{\@@eloop\loopdepth}{1}%
   \long\setvalue{\@@aloop\loopdepth}{#1}%
   \dodoloop}

\def\dodonoloop%
  {\global\advance\outerloop by -1\relax}

\def\dododoloop%
  {\edef\looplevel{\getvalue{\@@iloop\loopdepth}}%
   \scratchcounter=\looplevel
   \advance\scratchcounter by 1
   \setevalue{\@@iloop\loopdepth}{\the\scratchcounter}%
   \getvalue{\@@aloop\loopdepth}%
   \edef\looplevel{\getvalue{\@@iloop\loopdepth}}%
   \dodoloop}

\def\dodoloop%
  {\ifnum\getvalue{\@@eloop\loopdepth}=0
     \let\next=\dodonoloop
   \else
     \let\next=\dododoloop
   \fi
   \next}

\def\doloop%
  {\dostepwiserecurse{1}{\maxdimen}{1}}

\def\exitloop
  {\setvalue{\@@irecurse\recursedepth}{\maxdimen}}

\def\looplevel{\recurselevel}
\def\loopdepth{\recursedepth}
```

We prefer however a more byte saving implementation, that executes of course a bit slower.

30
```
\def\doloop%
  {\dostepwiserecurse{1}{\maxdimen}{1}}
```

31
```
\def\exitloop
  {\setvalue{\@@irecurse\recursedepth}{\maxdimen}}
```

We don't declare new counters for `\looplevel` and `\loopdepth` because one can use `\recurselevel` and `\recursedepth`.

The loop is executed at least once, so beware of situations like:

```
  \doloop {\exitloop some commands}
```

It's just a matter of putting the text into the `\if` statement that should be there anyway, like in:

```
  \doloop {\ifwhatever \exitloop \else some commands\fi}
```

\newevery
\everyline
\EveryLine
\EveryPar

Lets skip to something quite different. It's common use to use `\everypar` for special purposes. In CONTEXT we use this primitive for locating sidefloats. This means that when user assignments to `\everypar` can interfere with those of the package. We therefore introduce `\EveryPar`.

The same goes for `\EveryLine`. Because TEX offers no `\everyline` primitive, we have to call for `\everyline` when we are working on a line by line basis. Just by calling `\EveryPar{}` and `\EveryLine{}` we restore the old situation.

The definition command `\DoWithEvery` will be quite unreadable, so let's first show an implementation that shows how things are done:

```
\newtoks \everyline
\newtoks \oldeveryline
\newif   \ifeveryline

\def\DoWithEvery#1#2#3#4%
  {#3\else\edef\next{\noexpand#2={\the#1}}\next\fi
   \edef\next{\noexpand#1={\the#2\the\scratchtoks}}\next
   #4}

\def\doEveryLine%
  {\DoWithEvery\everyline\oldeveryline\ifeveryline\everylinetrue}

\def\EveryLine%
  {\afterassignment\doEveryLine\scratchtoks}

The real implementation is a bit more complicated but we
prefer something more versatile.
\stopdocumentation
```

```
\startdefinition
\def\DoWithEvery#1%
  {\csname if\strippedcsname#1\endcsname \else
     \edef\next%
       {\@EA\noexpand\csname old\strippedcsname#1\endcsname=
          {\the#1}}%
     \next
   \fi
   \edef\next%
     {\noexpand#1=
        {\@EA\the\csname old\strippedcsname#1\endcsname\the\scratchtoks}}%
   \next
   \csname\strippedcsname#1true\endcsname}
\stopdefinition

\startdefinition
\def\dowithevery#1%
  {\@EA\afterassignment\csname do\strippedcsname#1\endcsname\scratchtoks}
\stopdefinition

\startdefinition
\def\newevery#1#2%
  {\ifx#2\undefined
     \ifx#1\undefined\newtoks#1\fi
     \@EA\newtoks\csname old\strippedcsname#1\endcsname
     \@EA\newif  \csname  if\strippedcsname#1\endcsname
     \@EA\def    \csname  do\strippedcsname#2\endcsname{\DoWithEvery#1}%
     \def#2{\dowithevery#2}%
   \fi}
```

syst-tex
syst-gen
syst-ext
syst-new

```
\stopdefinition

\startdocumentation
This one permit sdefinitions like:
\stopdocumentation

\startdefinition
\newevery \everypar  \EveryPar
\newevery \everyline \EveryLine
\stopdefinition

\startdocumentation
Technically spoken we could have used the method we are
going to present in the visual debugger. First we save
the primitive \type{\everypar}:

\starttypen
\let\normaleverypar=\everypar
```

Next we allocate a ⟨*tokenlist*⟩ named `\everypar`, which means that `\everypar` is no longer a primitive but something like `\toks44`.

```
\newtoks\everypar
```

Because TEX now executes `\normaleverypar` instead of `\everypar`, we are ready to assign some tokens to this internally known and used ⟨*tokenlist*⟩.

```
\normaleverypar={all the things the system wants to do \the\everypar}
```

Where the user can provide his own tokens to be expanded every time he expects them to expand.

syst-tex
syst-gen
syst-ext
syst-new

```
\everypar={something the user wants to do}
```

We don't use this method because it undoubtly leads to confusing situations, especially when other packages are used, but it's this kind of tricks that make TEX so powerful.

\convertargument
\convertcommand

Some persistent experimenting led us to the next macro. This macro converts a parameter or an expanded macro to it's textual meaning.

```
\convertargument ... \to \command
```

For example,

```
\convertargument{one \two \three{four}}\to\ascii
```

The resulting macro `\ascii` can be written to a file or the terminal without problems. In CONTEXT we use this macro for generating registers and tables of contents.

The second conversion alternative accepts a command:

```
\convertcommand\command\to\ascii
```

Both commands accept the prefix `\doglobal` for global assignments.

32
```
\def\doconvertargument#1>{}
```

33
```
\def\convertedcommand%
  {\expandafter\doconvertargument\meaning}
```

34
```
\long\def\convertargument#1\to#2%
  {\long\def\convertedargument{#1}%
   \dodoglobal\edef#2%
     {\convertedcommand\convertedargument}}
```

syst-tex
syst-gen
syst-ext
syst-new

35
```
\long\def\convertcommand#1\to#2%
  {\dodoglobal\edef#2%
     {\convertedcommand#1}}
```

This is typically a macro that one comes to after reading the TEXbook carefully. Even then, the definite solution was found after rereading the TEXbook. The first implementation was:

```
\def\doconvertargument#1->#2\\\\{#2}
```

The -, the delimiter \\\\ and the the second argument are completely redundant.

\ExpandFirstAfter
\ExpandSecondAfter
\ExpandBothAfter

These three commands support expansion of arguments before executing the commands that uses them. We can best illustrate this with an example.

```
\def\first  {alfa,beta,gamma}
\def\second {alfa,epsilon,zeta}

\ExpandFirstAfter  \doifcommon {\first} {alfa}    {\message{OK}}
\ExpandSecondAfter \doifcommon {alfa}   {\second} {\message{OK}}
\ExpandBothAfter   \doifcommon {\first} {\second} {\message{OK}}

\ExpandFirstAfter\processcommalist[\first]\message

\ExpandAfter       \doifcommon {\first} {alfa}    {\message{OK}}
```

The first three calls result in the threefold message OK, the fourth one shows the three elements of \first. The command \ExpandFirstAfter takes care of (first) arguments that are delimited by [ ], but the faster \ExpandAfter does not.

RECONSIDER

contents register context syst mult supp lang font colo spec core cont m s exit go back

syst-tex
syst-gen
syst-ext
syst-new

36
```
\def\simpleExpandFirstAfter#1%
  {\edef\!!stringa{#1}%
   \@EA\ExpandCommand\@EA{\!!stringa}}
```

37
```
\def\complexExpandFirstAfter[#1]%
  {\edef\!!stringa{#1}%
   \@EA\ExpandCommand\@EA[\!!stringa]}
```

38
```
\def\ExpandFirstAfter#1%
  {\def\ExpandCommand{#1}%
   \complexorsimple{ExpandFirstAfter}}
```

39
```
\def\ExpandSecondAfter#1#2#3%
  {\def\!!stringa{#2}%
   \edef\!!stringb{#3}%
   \@EA#1\@EA{\@EA\!!stringa\@EA}\@EA{\!!stringb}}
```

40
```
\def\ExpandBothAfter#1#2#3%
  {\edef\!!stringa{#2}%
   \edef\!!stringb{#3}%
   \@EA\@EA\@EA#1\@EA\@EA\@EA{\@EA\!!stringa\@EA}\@EA{\!!stringb}}
```

41
```
\def\ExpandAfter#1#2%
  {\edef\!!stringa{#2}%
   \@EA#1\@EA{\!!stringa}}
```

Now we can for instance redefine `\ifinstringelse` as:

42
```
\def\ifinstringelse%
  {\ExpandBothAfter\v!ifinstringelse}
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

\ConvertToConstant
\ConvertConstantAfter

When comparing arguments with a constant, we can get into trouble when this argument consists of tricky expandable commands. One solution for this is converting the argument to a string of unexpandable characters. To make comparison possible, we have to convert the constant too

```
\ConvertToConstant\doifelse {...} {...} {then ...} {else ...}
```

This construction is only needed when the first argument can give troubles. Misuse can slow down processing.

```
\ConvertToConstant\doifelse{\c!alfa}        {\c!alfa}{...}{...}
\ConvertToConstant\doifelse{alfa}           {\c!alfa}{...}{...}
\ConvertToConstant\doifelse{alfa}           {alfa}   {...}{...}
\ConvertToConstant\doifelse{alfa \alfa test}{\c!alfa}{...}{...}
```

In examples 2 and 3 both arguments equal, in 1 and 4 they differ.

43
```
\def\ConvertToConstant#1#2#3%
  {\expandafter\convertargument\expandafter{#2}\to\!!stringa
   \expandafter\convertargument\expandafter{#3}\to\!!stringb
   #1{\!!stringa}{\!!stringb}}
```

When the argument `#1` consists of commands, we had better use

```
\ConvertConstantAfter\processaction[#1][...]
\ConvertConstantAfter\doifelse{#1}{\v!iets}{}{}
```

This commands accepts things like:

```
\v!constant
constant
\hbox to \hsize{\rubish}
```

syst-tex
syst-gen
syst-ext
syst-new

contents register context syst mult supp lang font colo spec core cont m s exit go back

As we will see in the core moudles, this macro permits constructions like:

```
\setupfoottexts[...][...]
\setupfoottexts[margin][...][...]
\setupfoottexts[\v!margin][...][...]
```

where ... can be anything legally TEX.

44
```
\def\CheckConstantAfter#1#2%
  {\@EA\convertargument\v!prefix!\to\ascii
   \convertargument#1\to#2\relax
   \doifinstringelse{\ascii}{#2}
     {\expandafter\convertargument#1\to#2}
     {}}
```

45
```
\def\simpleConvertConstantAfter#1#2%
  {\CheckConstantAfter{#1}\asciiA
   \CheckConstantAfter{#2}\asciiB
   \ConvertCommand{\asciiA}{\asciiB}}
```

46
```
\def\complexConvertConstantAfter[#1]%
  {\doConvertConstantAfter{#1}%
   \@EA\ConvertCommand\@EA[\!!stringa]}
```

47
```
\def\ConvertConstantAfter#1%
  {\def\ConvertCommand{#1}%
   \complexorsimple{ConvertConstantAfter}}
```

syst-tex
syst-gen
syst-ext
syst-new

contents register context syst mult supp lang font colo spec core cont m s exit go back

syst-tex
syst-gen
syst-ext
syst-new

\assignifempty

We can assign a default value to an empty macro using:

```
\assignifempty \macro {default value}
```

We don't explicitly test if the macro is defined.

48
```
\def\assignifempty#1#2%
  {\doifnot{#1}{}
     {\def#1{#2}}}
```

\gobbleuntil
\grabuntil
\processbetween

In TEX gobbling usually stand for skipping arguments, so here are our gobbling macros.

In CONTEXT we use a lot of `\start`–`\stop` like constructions. Sometimes, the `\stop` is used as a hard coded delimiter like in:

```
\def\startcommand#1\stopcommand%
  {... #1 ...}
```

In many cases the `\start`–`\stop` pair is defined at format generation time or during a job. This means that we cannot hardcode the `\stop` criterium. Only after completely understanding `\csname` and `\expandafter` I was able to to implement a solution, starting with:

```
\grabuntil{stop}\command
```

This commands executed, after having encountered `\stop` the command `\command`. This command receives as argument the text preceding the `\stop`. This means that:

```
\def\starthello%
  {\grabuntil{stophello}\message}

\starthello Hello world!\stophello
```

results in: `\message{Hello world!}`.

syst-tex
syst-gen
syst-ext
syst-new

49
```
\def\dograbuntil#1#2%
  {\long\def\next##1#1{#2{##1}}\next}
```

50
```
\def\grabuntil#1%
  {\expandafter\dograbuntil\expandafter{\csname#1\endcsname}}
```

The next command build on this mechanism:

```
\processbetween{string}\command
```

Here:

```
\processbetween{hello}\message
\starthello Hello again!\stophello
```

leads to: `\message{Hello again!}`. The command

```
\gobbleuntil\command
```

is related to these commands. This one simply throws away everything preceding `\command`.

51
```
\long\def\processbetween#1#2%
  {\setvalue{\s!start#1}%
     {\grabuntil{\s!stop#1}{#2}}}
```

52
```
\def\gobbleuntil#1%
  {\long\def\next##1#1{}\next}
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

\groupedcommand

syst-tex
syst-gen
syst-ext
syst-new

Commands often manipulate argument as in:

```
\def\doezomaarwat#1{....#1....}
```

A disadvantage of this approach is that the tokens that form `#1` are fixed the the moment the argument is read in. Normally this is no problem, but for instance verbatim environments adapt the ⟨*catcodes*⟩ of characters and therefore are not always happy with already fixed tokens.

Another problem arises when the argument is grouped not by `{}` but by `\bgroup` and `\egroup`. Such an argument fails, because the `\bgroup` is een as the argument (which is quite normal).

The next macro offers a solution for both unwanted situations:

```
\groupedcommand {before} {after}
```

Which can be used like:

```
\def\cite%
  {\groupedcommand{\rightquote\rightquote}{\leftquote\leftquote}}
```

This command is equivalent to, but more 'robust' than:

```
\def\cite#1%
  {\rightquote\rightquote#1\leftquote\leftquote}
```

One should say that the next implementation would suffice:

```
\def\groupedcommand#1#2%
  {\def\BeforeGroup{#1\ignorespaces}%
   \def\AfterGroup{\unskip#2\egroup}%
   \bgroup\bgroup
   \aftergroup\AfterGroup
   \afterassignment\BeforeGroup
```

```
      \let\next=}
```

It did indeed, but one day we decided to support the processing of boxes too:

```
   \def\rightword%
     {\groupedcommand{\hfill\hbox}{\parfillskip\!!zeropoint}}

   .......... \rightword{the right way}
```

Here TEX typesets `\bf the right way` unbreakable at the end of the line. The solution mentioned before does not work here.

```
   \long\unexpanded\def\groupedcommand#1#2%
     {\bgroup
      \long\def\BeforeGroup%
        {\bgroup#1\bgroup\aftergroup\AfterGroup}%
      \long\def\AfterGroup%
        {#2\egroup\egroup}%
      \afterassignment\BeforeGroup
      \let\next=}
```

We used this method some time until the next alternative was needed. From now on we support both

```
   to be \bold{bold} or not, that's the question
```

and

```
   to be {\bold bold} or not, that's the question
```

This alternative checks for a `\bgroup` token first. The internal alternative does not accept the box handling mentioned before, but further nesting works all right. The extra `\bgroup`–`\egroup` is needed to keep `\AfterGroup` both into sight and local.

syst-tex
syst-gen
syst-ext
syst-new

contents register context syst mult supp lang font colo spec core cont m s exit go back

53
```
\long\def\HandleGroup#1#2%
  {\bgroup
   \long\def\BeforeGroup%
     {\bgroup#1\bgroup\aftergroup\AfterGroup}%
   \long\def\AfterGroup%
     {#2\egroup\egroup}%
   \afterassignment\BeforeGroup
   \let\next=}
```

54
```
\long\def\HandleNoGroup#1#2%
  {\long\def\AfterGroup{#2\egroup}%
   \bgroup\aftergroup\AfterGroup#1}
```

55
```
\long\unexpanded\def\groupedcommand#1#2%
  {\def\dogroupedcommand%
     {\ifx\next\bgroup
        \let\next=\HandleGroup
      \else
        \let\next=\HandleNoGroup
      \fi
      \next{#1}{#2}}%
   \futurelet\next\dogroupedcommand}
```

Users should be aware of the fact that grouping can interfere with ones paragraph settings that are executed after the paragraph is closed. One should therefore explictly close the paragraph with `\par`, else the settings will be forgotten and not applied. So it's:

```
   \def\BoldRaggedCenter%
     {\groupedcommand{\raggedcenter\bf}{\par}}
```

\checkdefined

The bigger the system, the greater the change that user defined commands collide with those that are part of the system. The next macro gives a warning when a command is already defined. We considered blocking the definition, but this is not always what we want.

```
\checkdefined {category} {class} {command}
```

The user is warned with the suggestion to use CAPITALS. This suggestion is feasible, because CONTEXTonly defines lowcased macros.

56

```
\def\checkdefined#1#2#3%
  {\doifdefined{#3}
     {\writestatus{#1}{#2 #3 replaces a macro, use CAPITALS!}}}
```

\GotoPar
\GetPar

Typesetting a paragraph in a special way can be done by first grabbing the contents of the paragraph and processing this contents grouped. The next macro for instance typesets a paragraph in boldface.

```
\def\remark#1\par%
  {\bgroup\bf#1\egroup}
```

This macro has to be called like

```
\remark some text ... ending with \par
```

Instead of \par we can of course use an empty line. When we started typesetting with TEX, we already had produced lots of text in plain ASCII. In producing such simple formatted texts, we adopted an open layout, and when switching to TEX, we continued this open habit. Although TEX permits a cramped and badly formatted source, it adds to confusion and sometimes introduces errors. So we prefer:

```
\remark

some text ... ending with an empty line
```

We are going to implement a mechanism that allows such open specifications. The definition of the macro handling `\remark` becomes:

```
\def\remark%
  {\BeforePar{\bgroup\bf}%
   \AfterPar{\egroup}%
   \GetPar}
```

A macro like `\GetPar` can be defined in several ways. The recent version, the fourth one in a row, originally was far more complicated, but some functionality has been moved to other macros.

We start with the more simple but in some cases more appropriate alternative is `\GotoPar`. This one leaves `\par` unchanged and is therefore more robust. On the other hand, `\AfterPar` is not supported.

57
```
\newtoks\BeforePar
\newtoks\AfterPar
```

58
```
\def\doGotoPar%
  {\ifx\nextchar\blankspace
     \let\donext=\GotoPar
   \else\ifx\nextchar\endoflinetoken
     \let\donext=\GotoPar
   \else
     \def\donext%
       {\the\BeforePar
        \BeforePar{}%
        \nextchar}%
   \fi\fi
   \donext}
```

59
```
\def\GotoPar%
  {\afterassignment\doGotoPar\let\nextchar=}
```

Its big brother `\GetPar` redefines the `\par` primitive, which can lead to unexpected results, depending in the context.

60
```
\def\GetPar%
  {\edef\next%
     {\BeforePar
        {\the\BeforePar
         \BeforePar{}%
         \bgroup
         \def\par%
           {\egroup
            \par
            \the\AfterPar
            \BeforePar{}%
            \AfterPar{}}}}%
   \next
   \GotoPar}
```

\dowithpargument
\dowithwargument

The next macros are a variation on `\GetPar`. When macros expect an argument, it interprets a grouped sequence of characters a one token. While this adds to robustness and less ambiguous situations, we sometimes want to be a bit more flexible, or at least want to be a bit more tolerant to user input.

We start with a commands that acts on paragraphs. This command is called as:

```
   \dowithpargument\command
   \dowithpargument{\command ... }
```

syst-tex
syst-gen
syst-ext
syst-new

In CONTEXT we use this one to read in the titles of chapters, sections etc. The commands responsible for these activities accept several alternative ways of argument passing. In these examples, the `\par` can be omitted when an empty line is present.

```
\command{...}
\command ... \par
\command
  {...}
\command
  ... \par
```

We show two implementations, of which for the moment the we prefier to use the second one:

```
\def\dowithpargument#1%
  {\def\dodowithpargument%
     {\ifx\next\bgroup
        \def\next{#1}%
      \else
        \def\next####1 \par{#1{####1}}%
      \fi
      \next}%
   \futurelet\next\dodowithpargument}
```

A second and better implementation was:

```
\def\dowithpargument#1%
  {\def\nextpar##1 \par{#1{##1}}%
   \def\nextarg##1{#1{##1}}%
   \doifnextcharelse{\bgroup}
     {\nextarg}
     {\nextpar}}
```

We ended up with an alternative that also accepts en empty argument. This command permits for instance chapters to have no title.

61
```
\def\dowithpargument#1%
  {\def\nextpar##1 \par{#1{##1}}%
   \def\nextarg##1{#1{##1}}%
   \doifnextcharelse{\bgroup}
     {\nextarg}
     {\doifnextcharelse{\par}
        {#1{}}
        {\nextpar}}}
```

The `p` in the previous command stands for paragraph. When we want to act upon words we can use the `w` alternative.

```
   \dowithwargument\command
   \dowithwargument{... \command ...}
```

The main difference bwteen two alternatives is in the handling of `\par`'s. This time the space token acts as a delimiter.

```
   \command{...}
   \command ...
   \command
     {...}
   \command
     ...
```

Again there are two implementations possible:

```
   \def\dowithwargument#1%
```

```
      {\def\dodowithwargument%
         {\ifx\next\bgroup
            \def\next{#1}%
          \else
            \def\next####1 {#1{####1}}%
          \fi
          \next}%
       \futurelet\next\dodowithwargument}
```

We've chosen:

62
```
\def\dowithwargument#1%
  {\def\nextwar##1 {#1{##1}}%
   \def\nextarg##1{#1{##1}}%
   \doifnextcharelse{\bgroup}
     {\nextarg}
     {\nextwar}}
```

\dorepeat
\dorepeatwithcommand

When doing repetitive tasks, we stromgly advice to use `\dorecurse`. The next alternative however, suits better some of the CONTEXT interface commands.

```
   \dorepeat[n*\command]
```

The value of the used ⟨*counter*⟩ can be called within `\command` by `\repeater`.

A slightly different alternative is:

```
   \dorepeatwithcommand[n*{...}]\command
```

When we call for something like:

```
   \dorepeatwithcommand[3*{Hello}]\message
```

syst-tex
syst-gen
syst-ext
syst-new

we get ourselves three `\message{Hello}` messages in a row. In both commands, the `n*` is optional. When this specification is missing, the command executes once.

63
```
\long\def\dodorepeat[#1*#2*#3*]%
  {\doifelse{#3}{}
     {#1}
     {\dorecurse{#1}{#2}}}
```

64
```
\long\def\dorepeat[#1]%
  {\dodorepeat[#1***]}
```

65
```
\def\repeater%
  {\recurselevel}
```

66
```
\def\dorepeatwithcommand[#1]#2%
  {\def\p!dorepeatnot%
     {#2{#1}}%
   \def\p!dorepeatyes[##1*##2]%
     {\dorecurse{##1}{#2{##2}}}%
   \doifinstringelse{*}{#1}
     {\doifnumberelse{#1}{\p!dorepeatyes[#1]}{\p!dorepeatnot}}%
     {\p!dorepeatnot}}
```

`\appendtoks` `\prependtoks` `\flushtoks` `\dotoks`

We use ⟨*tokenlists*⟩ sparsely within CONTEXT, because the comma separated lists are more suitable for the user interface. Nevertheless we have:

```
(\doglobal) \appendtoks ... \to\tokenlist
(\doglobal) \prependtoks ... \to\tokenlist
(\doglobal) \flushtoks\tokenlist
            \dotoks\tokenlist
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

Er worden eerst enkele klad–registers gedefinieerd. These macros are clones of the ones implemented in page 378 of Knuth's TEXbook.

67
```
\def\appendtoks#1\to#2%
  {\scratchtoks={#1}%
   \edef\next{\noexpand#2={\the#2\the\scratchtoks}}%
   \next
   \doglobal#2=#2}
```

68
```
\def\prependtoks#1\to#2%
  {\scratchtoks={#1}%
   \edef\next{\noexpand#2={\the\scratchtoks\the#2}}%
   \next
   \doglobal#2=#2}
```

69
```
\def\flushtoks#1%
  {\scratchtoks=#1\relax
   \doglobal#1={}%
   \the\scratchtoks\relax}
```

70
```
\let\dotoks=\the
```

\makecounter
\pluscounter
\minuscounter
\resetcounter
\setcounter
\countervalue

Declaring, setting and resetting ⟨*counters*⟩ can be doen with the next set of commands.

```
\makecounter   {name}
\pluscounter   {name}
\minuscounter  {name}
\resetcounter  {name}
\setcounter    {name} {value}
\countervalue  {name}
```

We prefer the use of global counters. This means that we have to load PLAIN TEX in a bit different way:

```
\let\oldouter=\outer
\let\outer=\relax
\input plain.tex
\let\outer=\oldouter

\def\newcount%
  {\alloc@0\count\countdef\insc@unt}
```

First we show a solution in which we use real ⟨*counters*⟩. Apart from some expansion, nothing special is done.

```
\def\makecounter#1%
  {\expandafter\newcount\csname#1\endcsname}

\def\pluscounter#1%
  {\expandafter\global\expandafter\advance\csname#1\endcsname by 1 }

\def\minuscounter#1%
  {\expandafter\global\expandafter\advance\csname#1\endcsname by -1 }
```

syst-tex
syst-gen
syst-ext
syst-new

```
   \def\resetcounter#1%
     {\expandafter\global\csname#1\endcsname=0 }

   \def\setcounter#1#2%
     {\expandafter\global\csname#1\endcsname=#2 }

   \def\countervalue#1%
     {\the\getvalue{#1}}
```

Because these macros are already an indirect way of working with counters, there is no harm in using pseudo ⟨*counters*⟩ here:

71
```
\def\makecounter#1%
  {\setxvalue{#1}{0}}
```

72
```
\def\pluscounter#1%
  {\scratchcounter=\getvalue{#1}\relax
   \advance\scratchcounter by 1\relax
   \setxvalue{#1}{\the\scratchcounter}}
```

73
```
\def\minuscounter#1%
  {\scratchcounter=\getvalue{#1}\relax
   \advance\scratchcounter by -1\relax
   \setxvalue{#1}{\the\scratchcounter}}
```

74
```
\def\resetcounter#1%
  {\setxvalue{#1}{0}}
```

75
```
\def\setcounter#1#2%
  {\scratchcounter=#2\relax
   \setxvalue{#1}{\the\scratchcounter}}
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

76
```
\def\countervalue#1%
  {\getvalue{#1}}
```

\beforesplitstring \aftersplitstring These both commands split a string at a given point in two parts, so `x.y` becomes `x` or `y`.

```
   \beforesplitstring test.tex\at.\to\filename
   \aftersplitstring  test.tex\at.\to\extension
```

The first routine looks (and is indeed) a bit simpler than the second one. The alternative looking more or less like the first one did not always give the results we needed. Both implementations show some insight in the manipulation of arguments.

77
```
\def\beforesplitstring#1\at#2\to#3%
  {\def\dosplitstring##1#2##2#2##3\\%
     {\def#3{##1}}%
   \@EA\dosplitstring#1#2#2\\}
```

78
```
\def\aftersplitstring#1\at#2\to#3%
  {\def\dosplitstring##1#2##2@@@##3\\%
     {\def#3{##2}}%
   \@EA\dosplitstring#1@@@#2@@@\\}
```

\removesubstring A first application of the two routines defined above is:

```
   \removesubstringtest-\from first-last\to\nothyphenated
```

Which in terms of TEX looks like:

79
```
\def\removesubstring#1\from#2\to#3%
  {\doifinstringelse{#1}{#2}
     {\beforesplitstring#2\at#1\to\!!stringa
      \aftersplitstring #2\at#1\to\!!stringb
```

syst-tex
syst-gen
syst-ext
syst-new

```
      \edef#3{\!!stringa\!!stringb}%
      \def\next{\removesubstring#1\from#3\to#3}}
    {\let\next=\relax}%
  \next}
```

\addtocommalist
\removefromcommalist

When working with comma separated lists, oen sooner or later want the tools to append or remove items from such a list. When we add an item, we first check if it's already there. This means that every item in the list is unique.

```
\addtocommalist      {alfa}  \naam
\addtocommalist      {beta}  \naam
\addtocommalist      {gamma} \naam
\removefromcommalist {beta}  \naam
```

These commands can be prefixed with `\doglobal`. The implementation of the second command is more complecated, because we have to take leading spaces into account. Keep in mind that useres may provide lists with spaces after the commas. When one item is left, we also have to get rid of trailing spaces.

```
\def\words{alfa, beta, gamma, delta}
\def\words{alfa,beta,gamma,delta}
```

Removing an item takes more time than adding one.

80
```
\def\addtocommalist#1#2%
  {\doifelse{#2}{}
     {\dodoglobal\edef#2{#1}}
     {\edef\!!stringa{#2,,}%
      \beforesplitstring#2\at,,\to#2\relax
      \ExpandBothAfter\doifnotinset{#1}{#2}
        {\dodoglobal\edef#2{#2,#1}}}}
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

syst-tex
syst-gen
syst-ext
syst-new

81
```
\def\doremovefromcommalist#1#2#3%
  {\edef\!!stringa{,,#3,,}%
   \beforesplitstring\!!stringa\at,#1#2,\to\!!stringb
   \aftersplitstring\!!stringa\at,#1#2,\to\!!stringc
   \edef#3{\!!stringb,\!!stringc}%
   \aftersplitstring#3\at,,\to#3\relax
   \beforesplitstring#3\at,,\to#3}
```

82
```
\def\dodofrontstrip[#1#2]#3%
  {\ifx#1\space
     \def#3{#2}%
   \else
     \def#3{#1#2}%
   \fi}%
```

83
```
\def\dofrontstrip#1%
  {\edef\!!stringa{#1}%
   \ifx\!!stringa\empty
   \else
     \@EA\dodofrontstrip\@EA[#1]#1%
   \fi}
```

84
```
\def\removefromcommalist#1#2%
  {\doremovefromcommalist{ }{#1}{#2}%
   \doremovefromcommalist{}{#1}{#2}%
   \dofrontstrip#2%
   \dodoglobal\edef#2{#2}}
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

\withoutunit
\withoutpt
\PtToCm
\numberofpoints
\dimensiontocount

syst-tex
syst-gen
syst-ext
syst-new

We can convert point into centimeters with:

```
\PtToCm{dimension}
```

Splitting the value and the unit is done by:

85
```
\def\withoutunit#1#2%
  {\bgroup
   \dimen0=#1\relax
   \@EA\convertargument\the\dimen0\to\asciiA
   \@EA\convertargument#2\to\asciiB
   \@EA\@EA\@EA\beforesplitstring\@EA\asciiA\@EA\at\asciiB\to\!!stringa%
   \!!stringa
   \egroup}
```

86
```
\def\withoutpt#1%
  {\withoutunit{#1}{pt}}
```

87
```
\def\withoutcm#1%
  {\withoutunit{#1}{cm}}
```

A bit faster alternative is one that manipulates the ⟨*catcodes*⟩.

88
```
{\catcode`\.=\@@other
 \catcode`\p=\@@other
 \catcode`\t=\@@other
 \gdef\WITHOUTPT#1pt{#1}}
```

89
```
\def\withoutpt#1%
  {\expandafter\WITHOUTPT#1}
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

syst-tex
syst-gen
syst-ext
syst-new

The capitals are needed because `p` and `t` have ⟨*catcode*⟩ 12, while macronames only permit tokens with the ⟨*catcode*⟩ 11. As a result we cannot use the `.group` primitives. Those who want to know more about this kind of manipulations, we advice to study the TEXbook in detail. Because this macro does not do any assignment, we can use it in the following way too.

90
```
\def\PtToCm#1%
  {\bgroup
   \scratchdimen=#1\relax
   \scratchdimen=0.0351459804\scratchdimen % 2.54/72.27
   \withoutpt{\the\scratchdimen}cm%
   \egroup}
```

We also support:

```
   \numberofpoints    {dimension}
   \dimensiontocount {dimension} {\count}
```

Both macros return a rounded number.

91
```
\def\numberofpoints#1%
  {\scratchdimen=#1\relax
   \advance\scratchdimen by .5pt\relax
   \withoutpt{\the\scratchdimen}}
```

92
```
\def\dimensiontocount#1#2%
  {\scratchdimen=#1\relax
   \advance\scratchdimen by .5pt\relax
   #2=\scratchdimen
   \divide#2 by \!!maxcard\relax}
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

syst-tex
syst-gen
syst-ext
syst-new

\swapdimens
\swapmacros

Simple but effective are the next two macros. There name exactly states their purpose. The `\scratchdimen` and `\!!stringa` can only be swapped when being the first argument.

93
```
\def\swapdimens#1#2%
  {\scratchdimen=#1\relax
   #1=#2\relax
   #2=\scratchdimen}
```

94
```
\def\swapmacros#1#2%
  {\let\!!stringa=#1\relax
   \let#1=#2\relax
   \let#2=\!!stringa\relax}
```

\setlocalhsize

Sometimes we need to work with the `\hsize` that is corrected for indentation and left and right skips. The corrected value is available in `\localhsize`, which needs to be calculated with `\setlocalhsize` first.

```
   \setlocalhsize        \hbox to \localhsize{...}
   \setlocalhsize[-1em]  \hbox to \localhsize{...}
   \setlocalhsize[.5ex]  \hbox to \localhsize{...}
```

These examples show us that an optional can be used. The value provided is added to `\localhsize`.

95
```
\newdimen\localhsize
```

96
```
\def\complexsetlocalhsize[#1]%
  {\localhsize=\hsize
   \advance\localhsize by -\parindent
   \advance\localhsize by -\leftskip
   \advance\localhsize by -\rightskip
   \advance\localhsize by #1\relax}
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

syst-tex
syst-gen
syst-ext
syst-new

97
```
\def\simplesetlocalhsize%
  {\complexsetlocalhsize[\!!zeropoint]}
```

98
```
\definecomplexorsimple\setlocalhsize
```

\processtokens

We fully agree with (most) typogaphers that inter–letter spacing is only permitted in fancy titles, we provide a macro that can be used to do so. Because this is (definitely and fortunately) no feature of TEX, we have to step through the token list ourselves.

```
   \processtokens {before} {between} {after} {space} {tokens}
```

An example of a call is:

```
   \processtokens {[} {+} {]} {\space} {hello world}
```

This results in:

[h+e+l+l+o+ +w+o+r+l+d]

The list of tokens may contain spaces, while \\, {} and \ are handled as space too.

99
```
\def\dodoprocesstokens%
  {\ifx\next\lastcharacter
     \after
     \let\next=\relax
   \else
     \expandafter\if\space\next
       \before\white
     \else
       \before\next
     \fi
     \let\before=\between
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
     \let\next=\doprocesstokens
   \fi
   \next}
```

100
```
\def\doprocesstokens% the space after = is essential
  {\afterassignment\dodoprocesstokens\let\next= }
```

101
```
\def\processtokens#1#2#3#4#5%
  {\bgroup
   \def\lastcharacter{\lastcharacter}%
   \def\space{ }%
   \let\\=\space
   \def\before{#1}%
   \def\between{#2}%
   \def\after{#3}%
   \def\white{#4}%
   \doprocesstokens#5\lastcharacter%
   \egroup}
```

\doifvalue
\doifnotvalue
\doifelsevalue
\doifnothing
\doifsomething
\doifelsenothing
\doifvaluenothing
\doifvaluesomething
\doifelsevaluenothing

These long named `\if` commands can be used to access macros (or variables) that are normally accessed by using `\getvalue`. Using these alternatives safes us three tokens per call. Anyone familiar with the not–values ones, can derive their meaning from the definitions.

```
      \def\doifvalue#1{\doif{\getvalue{#1}}}
   \def\doifnotvalue#1{\doifnot{\getvalue{#1}}}
  \def\doifelsevalue#1{\doifelse{\getvalue{#1}}}

    \def\doifnothing#1{\doif{#1}{}}
  \def\doifsomething#1{\doifnot{#1}{}}
\def\doifelsenothing#1{\doifelse{#1}{}}

    \def\doifvaluenothing#1{\doif{\getvalue{#1}}{}}
  \def\doifvaluesomething#1{\doifnot{\getvalue{#1}}{}}
\def\doifelsevaluenothing#1{\doifelse{\getvalue{#1}}{}}
```

\DOIF
\DOIFELSE
\DOIFNOT

TEX is case sensitive. When comparing arguments, this feature sometimes is less desirable, for instance when we compare filenames. The next three alternatives upcase their arguments before comparing them.

```
\DOIF     {string1} {string2} {...}
\DOIFNOT  {string1} {string2} {...}
\DOIFELSE {string1} {string2} {then ...}{else ...}
```

We have to use a two–step implementation, because the expansion has to take place outside `\uppercase`.

105
```
\def\p!DOIF#1#2#3%
  {\uppercase{\ifinstringelse{$#1$}{$#2$}}%
     #3%
   \fi}
```

syst-tex
syst-gen
syst-ext
syst-new

syst-tex
syst-gen
syst-ext
syst-new

106
```
\def\p!DOIFNOT#1#2#3%
  {\uppercase{\ifinstringelse{$#1$}{$#2$}}%
   \else
     #3%
   \fi}
```

107
```
\def\p!DOIFELSE#1#2#3#4%
  {\uppercase{\ifinstringelse{$#1$}{$#2$}}%
     #3%
   \else
     #4%
   \fi}
```

108
```
\def\DOIF     {\ExpandBothAfter\p!DOIF}
\def\DOIFNOT  {\ExpandBothAfter\p!DOIFNOT}
\def\DOIFELSE {\ExpandBothAfter\p!DOIFELSE}
```

\stripcharacters
\stripspaces

The next command was needed first when we implemented the CONTEXT interactivity macros. When we use labeled destinations, we often cannot use all the characters we want. We therefore strip some of the troublemakers, like spaces, from the labels before we write them to the DVI–file, which passes them to for instance a PostScript file.

```
   \stripspaces\from\one\to\two
```

Both the old string `\one` and the new one `\two` are expanded. This command is a special case of:

```
   \stripcharacter\char\from\one\to\two
```

As we can see below, spaces following a control sequence are to enclosed in `{}`.

109
```
\def\stripcharacter#1\from#2\to#3%
  {\def\dostripcharacter##1#1##2\end%
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

syst-tex
syst-gen
syst-ext
syst-new

```
      {\edef\p!strippedstring{\p!strippedstring##1}%
       \doifemptyelse{##2}
         {\let\next=\relax}
         {\def\next{\dostripcharacter##2\end}}%
       \next}%
   \let\p!strippedstring=\empty
   \edef\!!stringa{#2}%
   \@EA\dostripcharacter\!!stringa#1\end
   \let#3=\p!strippedstring}
```

110
```
\def\stripspaces\from#1\to#2%
  {\stripcharacter{ }\from#1\to#2}
```

`\executeifdefined`

CONTEXT uses one auxiliary file for all data concerning tables of contents, references, two–pass optimizations, sorted lists etc. This file is loaded as many times as needed. During such a pass we skip the commands thate are of no use at that moment. Because we don't want to come into trouble with undefined auxiliary commands, we call the macros in a way similar to `\getvalue`. The next macro take care of such executions and when not defined, gobbles the unwanted arguments.

```
   \executeifdefined{name}\gobbleoneargument
```

We can of course globble more arguments using the appropriate globbling command.

111
```
\def\executeifdefined#1#2%
  {\ifundefined{#1}%
     \def\next{#2}%
   \else
     \def\next{\getvalue{#1}}%
   \fi
   \next}
```

syst-tex
syst-gen
syst-ext
syst-new

We considered an alternative imlementation accepting commands directly, like:

```
\executeifdefined\naam\gobblefivearguments
```

For the moment we don't need this one, so we stick to the faster one. The more versatile alternative is:

```
\def\executeifdefined#1#2%
  {\setnameofcommand{#1}%
   \@EA\ifundefined\@EA{\nameofcommand}
     \def\next{#2}%
   \else
     \def\next{\getvalue{\nameofcommand}}%
   \fi
   \next}
```

\doifsomespaceelse

The next command checks a string on the presence of a space and executed a command accordingly.

```
\doifsomespaceelse {tekst} {then ...} {else ...}
```

We use this command in CONTEXT for determing if an argument must be broken into words when made interactive. Watch the use of `\noexpand`.

112

```
\long\def\doifsomespaceelse#1#2#3%
  {\def\c!doifsomespaceelse##1 ##2##3\war%
     {\if\noexpand##2@%
        #3%
      \else
        #2%
      \fi}%
   \c!doifsomespaceelse#1 @ @\war}
```

\adaptdimension
\balancedimensions

Again we introduce some macros that are closely related to an interface aspect of CONTEXT. The first command can be used to adapt a ⟨*dimension*⟩.

```
  \adaptdimension {dimension} {value}
```

When the value is preceed by a + or minus, the dimension is advanced accordingly, otherwise it gets the value.

113
```
\def\doadaptdimension#1#2\\#3\\%
  {\if#1+%
     \dodoglobal\advance#3 by #1#2\relax
   \else\if##1-%
     \dodoglobal\advance#3 by #1#2\relax
   \else
     \dodoglobal#3=#1#2\relax
   \fi\fi}
```

114
```
\def\adaptdimension#1#2%
  {\expandafter\doadaptdimension#2\\#1\\}
```

A second command takes two ⟨*dimensions*⟩. Both are adapted, depending on the sign of the given value. maat. This time we take the value as it is, and don't look explicitly at the preceding sign.

```
  \balancedimensions {dimension 1} {dimension 2} {value}
```

When a positive value is given, the first dimension is incremented, the second ond is decremented. A negative value has the opposite result.

115
```
\def\balancedimensions#1#2#3%
  {\scratchdimen=#3\relax
   \redoglobal\advance#1 by \scratchdimen\relax
   \dodoglobal\advance#2 by -\scratchdimen\relax}
```

syst-tex
syst-gen
syst-ext
syst-new

Both commands can be preceded by `\doglobal`. Here we use `\redo` first, because `\dodo` resets the global character.

\processconcanatedlist

Maybe a bit late, but here is a more general version of the `\processcommalist` command. This time we don't handle nesting but accept arbitrary seperators.

```
\processconcanatedlist[list][separator]\command
```

One can think of things like:

```
\processconcanatedlist[alfa+beta+gamma][+]\message
```

116

```
\def\processconcanatedlist[#1][#2]#3%
  {\def\doprocessconcanatedlist##1##2#2%
     {\if]##1%
        \let\next=\relax
      \else\if]##2%
        \let\next=\relax
      \else\ifx\blankspace##2%
        #3{##1}%
        \let\next=\doprocessconcanatedlist
      \else
        #3{##1##2}%
        \let\next=\doprocessconcanatedlist
      \fi\fi\fi
      \next}%
   \doprocessconcanatedlist#1#2]#2}
```

\processassignlist

Is possible to combine an assignment list with one containing keywords. Assignments are treated accordingly, keywords are treated by `\command`.

```
\processassignlist[...=...,...=...,...]\commando
```

This command can be integrated in `\getparameters`, but we decided best not to do so.

117

```
\def\processassignlist#1[#2]#3%
  {\def\p!dodogetparameter[##1=##2=##3]%
     {\doifnot{##3}{\relax}{#3{##1}}}%
   \def\p!dogetparameter##1%
     {\p!dodogetparameter[##1==\relax]}%
   \processcommalist[#2]\p!dogetparameter}
```

\DoAfterFi
\DoAfterFiFi

Sometimes `\fi`'s can get into the way. We can reach over such a troublemaker with:

```
\DoAfterFi{some commands}
\DoAfterFiFi{some commands}
```

It saves us a `\next` construction. Skipping `\else...\fi` is more tricky, so this one is not provided.

118

```
\def\DoAfterFi#1\fi{\fi#1}
\def\DoAfterFiFi#1\fi#2\fi{\fi\fi#1}
```

\untextargument untexc..

When manipulating data(bases) and for instance generating index entries, the next three macros can be of help:

```
\untextargument{...}\to\name
\untexcommand  {...}\to\name
```

They remove braces and backslashes and give us something to sort.

syst-tex
syst-gen
syst-ext
syst-new

119
```
\def\untexsomething%
  {\bgroup
   \catcode`\{=\@@ignore
   \catcode`\}=\@@ignore
   \escapechar=-1
   \dountexsomething}
```

120
```
\long\def\dountexsomething#1#2\to#3%
  {\doglobal#1#2\to\untexedargument
   \egroup
   \let#3=\untexedargument}
```

121
```
\def\untexargument%
  {\untexsomething\convertargument}
```

122
```
\def\untexcommand%
  {\untexsomething\convertcommand}
```

\ScaledPointsToBigPoints
\ScaledPointsToWholeBi..

One characteristic of POSTSCRIPT and PDF is that both used big points (TeX's bp). The next macros convert points and scaled points into big points.

```
   \ScaledPointsToBigPoints      {number} \target
   \ScaledPointsToWholeBigPoints {number} \target
```

The magic factor 72/72.27 can be found in most TeX related books.

123
```
\def\ScaledPointsToBigPoints#1#2%
  {\scratchdimen=#1sp\relax
   \scratchdimen=.996264\scratchdimen
   \edef#2{\withoutpt{\the\scratchdimen}}}
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

syst-tex
syst-gen
syst-ext
syst-new

124
```
\def\ScaledPointsToWholeBigPoints#1#2%
  {\scratchdimen=#1sp\relax
   \scratchdimen=.996264\scratchdimen
   \scratchcounter=\scratchdimen
   \advance\scratchcounter by \!!medcard
   \divide\scratchcounter by \!!maxcard
   \edef#2{\the\scratchcounter}}
```

\PointsToReal Points can be stripped from their suffix by using `\withoutpt`. The next macro enveloppes this macro.

```
   \PointsToReal {dimension} \target
```

125
```
\def\PointsToReal#1#2%
  {\scratchdimen=#1%
   \edef#2{\withoutpt{\the\scratchdimen}}}
```

\dontleavehmode Sometimes when we enter a paragraph with some command, the first token gets the whole first line. We can prevent this by saying:

```
   \dontleavehmode
```

This command is used in for instance the language module `lang-ini`.

126
```
\def\dontleavehmode{\ifmmode\else$ $\fi}
```

127
```
\protect
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

syst-tex
syst-gen
syst-ext
syst-new

syst-tex
syst-gen
syst-ext
syst-new

2.4 [to be documented: syst-new]

This module is not yet fully documented.

syst-tex
syst-gen
syst-ext
syst-new

contents register context syst ▲ mult supp lang font colo spec core cont m s exit go back

3 Multilingual Interface

3.1 Initialization

mult-ini
mult-sys
mult-con
mult-com

This module implements the multi–lingual interface to CONTEXT. These capabilities concern messages, commands and parameters.

```
\writestatus{loading}{Context Multilingual Macros / Initialization}
```

```
\unprotect
```

\v! \c! \s! \e! \m! \r! \f! \p! \x! \y!

In the system modules we introduced some prefixed constants, variables (both macros) and registers. Apart from a tremendous saving in terms of memory and a gain in speed we use from now on prefixes when possible for just another reason: consistency and multi–linguality. Systematically using prefixed macros enables us to implement a multi–lingual user interface. Redefining these next set of prefixes therefore can have desastrous results.

prefix	meaning	application
`\v!prefix!`	v!	variable
`\c!prefix!`	c!	constant
`\s!prefix!`	s!	system
`\e!prefix!`	e!	element
`\m!prefix!`	m!	message
`\r!prefix!`	r!	reference
`\f!prefix!`	f!	file
`\p!prefix!`	p!	procedure
`\x!prefix!`	x!	setup constant
`\y!prefix!`	y!	setup variable

In the single–lingual version we used `!`, `!!`, `!!!` and `!!!!`.

contents register context syst mult supp lang font colo spec core cont m s exit go back

3
```
\def\v!prefix!{v!} \def\c!prefix!{c!} \def\s!prefix!{s!}
\def\e!prefix!{e!} \def\m!prefix!{m!} \def\r!prefix!{r!}
\def\f!prefix!{f!} \def\p!prefix!{p!} \def\x!prefix!{x!}
\def\y!prefix!{y!}
```

\@@
\??

Variables generated by the system can be recognized on their prefix `@@`. They are composed of a command (class) specific tag, which can be recognized on `??`, and a system constant, which has the prefix `c!`. We'll se some more of this.

4
```
\def\??prefix  {??}
\def\@@prefix  {@@}
```

Just to be complete we repeat some of the already defined system constants here. Maybe their prefix `\s!` now falls into place.

5
```
\def\s!next    {next}           \def\s!default {default}
\def\s!dummy   {dummy}          \def\s!unknown {unknown}
```

6
```
\def\s!do      {do}             \def\s!dodo    {dodo}
```

7
```
\def\s!complex {complex}        \def\s!start   {start}
\def\s!simple  {simple}         \def\s!stop    {stop}
```

8
```
\def\!!width   {width}          \def\!!plus    {plus}
\def\!!height  {height}         \def\!!minus   {minus}
\def\!!depth   {depth}
```

\defineinterfaceconstant
\defineinterfacevariable
\defineinterfaceelement
\definesystemvariable
\definesystemconstant
\definemessageconstant
\definereferenceconstant
\definefileconstant

mult-ini
mult-sys
mult-con
mult-com

The first part of this module is dedicated to dealing with multi–lingual constants and variables. When CONTEXT grew bigger and bigger in terms of bytes and used string space, we switched to predefined constants. At the cost of more hash table entries, the macros not only becase more compact, they became much faster too. Maybe an even bigger advantage was that mispelling could no longer lead to problems. Even a multi–lingual interface became possible.

Constants — we'll introduce the concept of variables later on — are preceded by a type specific prefix, followed by a `!`. To force consistency, we provide a few commands for defining such constants.

```
\defineinterfaceconstant {name} {meaning}
\defineinterfacevariable {name} {meaning}
\defineinterfaceelement  {name} {meaning}
```

Which is the same as:

```
\def\c!name{meaning}
\def\v!name{meaning}
\def\e!name{meaning}
```

9
```
\def\defineinterfaceconstant #1#2{\setvalue{\c!prefix!#1}{#2}}
\def\defineinterfacevariable #1#2{\setvalue{\v!prefix!#1}{#2}}
\def\defineinterfaceelement  #1#2{\setvalue{\e!prefix!#1}{#2}}
```

Next come some interface independant constants:

```
\definereferenceconstant {name} {meaning}
\definefileconstant      {name} {meaning}
```

10
```
\def\definereferenceconstant #1#2{\setvalue{\r!prefix!#1}{#2}}
\def\definefileconstant      #1#2{\setvalue{\f!prefix!#1}{#2}}
```

And finaly we have the one argument, space saving constants

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
   \definesystemconstant    {name}
   \definemessageconstant   {name}
```

11
```
\def\definesystemconstant    #1{\setvalue{\s!prefix!#1}{#1}}
\def\definemessageconstant   #1{\setvalue{\m!prefix!#1}{#1}}
```

In a parameter driven system, some parameters are shared by more system components. In CONTEXT we can distinguish parameters by a unique prefix. Such a prefix is defined with:

```
   \definesystemvariable    {name}
```

12
```
\def\definesystemvariable    #1{\setvalue{\??prefix#1}{\@@prefix#1}}
```

\selectinterface

With **`\selectinterface`** we specify the language we are going to use. The system asks for the language wanted, and defaults to **`\currentinterface`** when we just give **`enter`**. By default the message system uses the current interface language, but **`\currentresponses`** can specify another language too.

13
```
\def\defaultinterface{dutch}
\def\currentinterface{dutch}
\def\currentresponses{dutch}
```

14
```
\def\selectinterface%
  {\def\docommando##1##2%
     {\bgroup
      \endlinechar=-1
      \global\read16 to ##1
      \egroup
      \doif{\currentinterface}{}{\let##1=##2}%
      \doifundefined{\s!prefix!##1}{\let##1=##2}}%
   \docommando\currentinterface\defaultinterface
```

```
    \writestatus{interface}{defining \currentinterface\space interface}%
    \writeline
    \docommando\currentresponses\currentinterface
    \writestatus{interface}{using \currentresponses\space messages}%
    \writeline}
```

\startinterface

Sometimes we want to define things only for specific interface languages. This can be done by means of the selector:

```
    \startinterface language

    language specific definitions & commands

    \stopinterface
```

```
\long\def\startinterface #1 #2\stopinterface%
  {\doifelse{#1}{\currentinterface}
     {\long\def\next{#2}}
     {\let\next=\relax}%
   \next}
```

\startmessages
\getmessage
\showmessage
\makemessage

A package as large as CONTEXT can hardly function without a decent message mechanism. Due to its multi–lingual interface, the message subsystem has to be multi–lingual too. A major drawback of this feature is that we have to code messages. As a result, the source becomes less self documented. On the other hand, consistency will improve.

Because the overhead in terms of entries in the (already exhausted) hash table has to be minimal, messages are packed in libraries. We can extract a message from such a library in three ways:

```
    \getmessage  {library} {tag}
    \showmessage {library} {tag} {data}
```

```
\makemessage {library} {tag} {data}
```

The first command gets the message `tag` from the `library` specified. The other commands take an extra argument: a list of items to be inserted in the message text. While `\showmessage` shows the message at the terminal, the other commands generate the message as text. Before we explain the `data` argument, we give an example of a library.

```
\startmessages  english  library: alfa
  title: something
      1: first message
      2: second (--) message --
\stopmessages
```

The first message is a simple one and can be shown with:

```
\showmessage {alfa} {1} {}
```

The second message on the other hand needs some extra data:

```
\showmessage {alfa} {2} {and last,to you}
```

This message is shown as:

```
something : second (and last) message to you
```

As we can see, the title entry is shown with the message. The data fields are comma separated and are specified in the message text by `--`.

It is not required to define all messages in a library at once. We can add messages to a library in the following way:

```
\startmessages  english  library: alfa
     10: tenth message
```

```
   \stopmessages
```

Because such definitions can take place in different modules, the system gives a warning when a tag occurs more than once. The first occurrence takes preference over later ones, so we had better use a save offset, as shown in the example. As we can see, the title field is specified only the first time!

Because we want to check for duplicate tags, the macros are a bit more complicated than neccessary. The `<newline>` token is used as message separator.

16
```
\def\findinterfacemessage#1#2%
  {\def#2{}%
   \def\dofindinterfacemessage##1 #1: ##2\relax##3\end%
     {\def#2{##2}}%
   \edef\!!stringa{\getvalue{@@ms\currentmessagelibrary} #1: \relax}%
   \expandafter\dofindinterfacemessage\!!stringa\end}
```

17
```
\def\composemessagetext#1--#2--#3--#4--#5--#6\\%
  {\def\docomposemessagetext##1,##2,##3,##4,##5,##6\\%
     {\edef\currentmessagetext{#1##1#2##2#3##3#4##4#5##5}}%
   \docomposemessagetext}
```

18
```
\unexpanded\def\getmessage#1#2%
  {\def\currentmessagelibrary{#1}%
   \findinterfacemessage{#2}\currentmessagetext
   \currentmessagetext}
```

19
```
\unexpanded\def\makemessage#1#2#3%
  {\def\currentmessagelibrary{#1}%
   \findinterfacemessage{#2}\currentmessagetext
   \@EA\composemessagetext\currentmessagetext----------\\#3,,,,,\\%
   \currentmessagetext}
```

mult-ini
mult-sys
mult-con
mult-com

20
```
\def\showmessage#1#2#3%
  {\def\currentmessagelibrary{#1}%
   \findinterfacemessage{#2}\currentmessagetext
   \findinterfacemessage{title}\currentmessagetitle
   \doifelse{\currentmessagetext}{}
     {\def\currentmessagetext{<unknown message #2>}}
     {\@EA\composemessagetext\currentmessagetext----------\\#3,,,,,\\}%
   \@EA\writestatus\@EA{\currentmessagetitle}{\currentmessagetext}}
```

21
```
\def\doaddinterfacemessage#1#2%
  {\findinterfacemessage{#1}\currentmessagetext
   \doifelse{\currentmessagetext}{}
     {\setxvalue{@@ms\currentmessagelibrary}%
        {\getvalue{@@ms\currentmessagelibrary} #1: #2\relax}}
     {\debuggerinfotrue % we consider this an important error
      \debuggerinfo
        {message}
        {duplicate tag #1
         in library \currentmessagelibrary\space
         of interface \currentresponses}
      \wait}%
   \futurelet\next\getinterfacemessage}
```

22
```
\bgroup
\obeylines
\gdef\addinterfacemessage#1: #2
  {\doaddinterfacemessage{#1}{#2}}%
\egroup
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

23
```
\def\getinterfacemessage%
  {\ifx\next\stopmessages
     \def\next##1{\egroup}%
   \else
     \let\next=\addinterfacemessage
   \fi
   \next}
```

24
```
\gdef\startmessages #1 library: #2
  {\bgroup
   \obeylines
   \doifinsetelse{#1}{\currentresponses,all}
     {\def\next%
        {\def\currentmessagelibrary{#2}%
         \doifundefined{@@ms\currentmessagelibrary}
           {\setgvalue{@@ms\currentmessagelibrary}{}}%
         \futurelet\next\getinterfacemessage}}
     {\long\def\next##1\stopmessages{\egroup}}%
   \next}
```

\dosetvalue
\dosetevalue
\docopyvalue
\doresetvalue
\dogetvalue

We already defined these auxiliary macros in the system modules. Starting with this module however, we have to take multi–linguality a bit more serious.

First we show a well–defined alternative:

```
   \def\dosetvalue#1#2#3%
     {\doifdefinedelse{\c!prefix!#2}
        {\setvalue{#1\getvalue{\c!prefix!#2}}{#3}}
        {\setvalue{#1#2}{#3}}}

   \def\docopyvalue#1#2#3%
```

mult-ini
mult-sys
mult-con
mult-com

```
     {\doifdefinedelse{\c!prefix!#3}
        {\setvalue{#1\getvalue{\c!prefix!#3}}%
           {\getvalue{#2\getvalue{\c!prefix!#3}}}}
        {\setvalue{#1#3}%
           {\getvalue{#2#3}}}}

   \def\dogetvalue#1#2%
     {\getvalue{#1\getvalue{\c!prefix!#2}}}
```

These macros are called upon quite often and so we optimized them a bit.

25
```
\def\dosetvalue#1#2#3%
  {\p!doifundefined{\c!prefix!#2}%
     \let\donottest=\doprocesstest
     \@EA\def\csname#1#2\endcsname{#3}%
   \else
     \let\donottest=\doprocesstest
     \@EA\def\csname#1\csname\c!prefix!#2\endcsname\endcsname{#3}%
   \fi}
```

26
```
\def\dosetevalue#1#2#3%
  {\p!doifundefined{\c!prefix!#2}%
     \let\donottest=\doprocesstest
     \@EA\edef\csname#1#2\endcsname{#3}%
   \else
     \let\donottest=\doprocesstest
     \@EA\edef\csname#1\csname\c!prefix!#2\endcsname\endcsname{#3}%
   \fi}
```

27
```
\def\docopyvalue#1#2#3%
  {\p!doifundefined{\c!prefix!#3}%
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
    \let\donottest=\doprocesstest
    \@EA\def\csname#1#3\endcsname%
      {\csname#2#3\endcsname}%
  \else
    \let\donottest=\doprocesstest
    \@EA\def\csname#1\csname\c!prefix!#3\endcsname\endcsname%
      {\csname#2\csname\c!prefix!#3\endcsname\endcsname}%
  \fi}
```

28

```
\def\doresetvalue#1#2%
  {\dosetvalue{#1}{#2}{}}
```

29

```
\def\dogetvalue#1#2%
  {\csname#1\csname\c!prefix!#2\endcsname\endcsname}
```

Although maybe bot clearly visible, there is a considerable profit in further optimalization. By expanding the embedded `\csname` we can reduce the format file by about 5% (60 KB out of 1.9 MB).

30

```
\def\docopyvalue#1#2#3%
  {\p!doifundefined{\c!prefix!#3}%
    \let\donottest=\doprocesstest
    \@EA\@EA\@EA\def\@EA\csname\@EA#1\@EA#3\@EA\endcsname
      \@EA{\csname#2#3\endcsname}%
  \else
    \let\donottest=\doprocesstest
    \@EA\@EA\@EA\def\@EA
        \csname
          \@EA#1\@EA\csname\@EA\c!prefix!\@EA#3\@EA\endcsname\@EA
        \endcsname
      \@EA{\csname#2\csname\c!prefix!#3\endcsname\endcsname}%
  \fi}
```

mult-ini
mult-sys
mult-con
mult-com

We take this opportunity of redefining to adapt an assignment macro. The change has to do with the fact that the generated error message must be multi–lingual. We can not define the message yet, because we still have to select the interface language.

31
```
\def\p!doassign#1[#2][#3=#4=#5]%
  {\ifx\empty#3\else  % and definitely not \ifx#3\empty
     \ifx\relax#5%
       \showmessage{check}{1}{#3,\the\inputlineno}%
     \else
       #1{#2}{#3}{#4}%
     \fi
   \fi}
```

32
```
\def\dogetargument#1#2#3#4%
  {\doifnextcharelse{#1}
     {\let\expectedarguments=\noexpectedarguments
      #3\dodogetargument}
     {\ifnum\expectedarguments>\noexpectedarguments
        \showmessage{check}{2}{\expectedarguments,\the\inputlineno}%
      \fi
      \let\expectedarguments=\noexpectedarguments
      #4\dodogetargument#1#2}}
```

33
```
\def\dogetgroupargument#1#2%
  {\def\nextnext%
     {\ifx\next\bgroup
        \let\expectedarguments=\noexpectedarguments
        \def\next{#1\dodogetargument}%
      \else\ifx\next\lineending
        \def\next{\bgroup\def\\ {\egroup\dogetgroupargument#1#2}\\}%
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

mult-ini
mult-sys
mult-con
mult-com

```
        \else\ifx\next\blankspace
          \def\next{\bgroup\def\\ {\egroup\dogetgroupargument#1#2}\\}%
        \else
          \ifnum\expectedarguments>\noexpectedarguments
            \showmessage{check}{2}{\expectedarguments,\the\inputlineno}%
          \fi
          \let\expectedarguments=\noexpectedarguments
          \def\next{#2\dodogetargument{}}%
        \fi\fi\fi
        \next}%
    \futurelet\next\nextnext}
```

34
```
\def\checkdefined#1#2#3%
  {\doifdefined{#3}
     {\showmessage{check}{3}{#2,#3}}}
```

CONTEXT is a parameter driven package. This means that users instruct the system by means of variables, values and keywords. These instructions take the form:

```
   \setupsomething[some variable=some value, another one=a keyword]
```

or by keyword only:

```
   \dosomething[this way, that way, no way]
```

Because the same variables can occur in more than one setup command, we have to be able to distinguish them. This is achieved by assigning them a unique prefix.

Imagine a setup command for boxed text, that enables us to specify the height and width of the box. Behide the scenes the command

```
   \setupbox [width=12cm, height=3cm]
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

results in something like

```
\<box><width>   {12cm}
\<box><height>  {3cm}
```

while a similar command for specifying the page dimensions of an A4 page results in:

```
\<page><width>  {21.0cm}
\<page><height> {27.9cm}
```

The prefixes `<box>` and `<page>` are hidden from users and can therefore be language independant. Variables on the other hand, differ for each language:

```
\<box><color>   {<blue>}
\<box><kleur>   {<blauw>}
\<box><couleur> {<blue>}
```

In this example we can see that the assigned values or keywords are language dependant too. This will be a complication when defining multi–lingual setup files.

A third phenomena is that variables and values can have a similar meaning.

```
\<pagenumber><location> {<left>}
\<skip><left>           {12cm}
```

A (minor) complication is that where in english we use `<left>`, in dutch we find both `<links>` and `<linker>`. This means that when we use some sort of translation table, we have to distinguish between the variables at the left side and the fixed values at the right.

The same goes for commands that are composed of different user supplied and/or language specific elements. In english we can use:

```
\<empty><figure>
```

```
\<empty><intermezzo>
```

But in dutch we have the following:

```
\<lege><figuur>
\<leeg><intermezzo>
```

These subtle differences automatically lead to a solution where variables, values, elements and other components have a similar logical name (used in macro's) but a different meaning (supplied by the user).

Our solution is one in which the whole system is programmed in terms of identifiers with language specific meanings. In such an implementation, each fixed variable is available as:

```
\<prefix><variable>
```

This means that for instance:

```
\setupbox[width=12cm]
```

expands to something like:

```
\def\boxwidth{12cm}
```

because we don't want to recode the source, a setup command in another language has to expand to this variable, so:

```
\stelblokin[breedte=12cm]
```

has to result in the definition of `\boxwidth` too. This method enables us to build compact, fast and readable code.

An alternative method, which we considered using, uses a more indirect way. In this case, both calls generate a different variable:

```
\def\boxwidth   {12cm}
\def\boxbreedte {12cm}
```

And because we don't want to recode those megabytes of already developed code, this variable has to be called with something like:

```
\valueof\box\width
```

where `\valueof` takes care of the translation of `width` or `breedte` to `width` and combining this with `box` to `\boxwidth`.

One advantage of this other scheme is that, within certain limits, we can implement an interface that can be switched to another language at will, while the current approach fixes the interface at startup. There are, by the way, other reasons too for not choosing this scheme. Switching user generated commands is for instance impossible and a dual interface would therefore give a strange mix of languages.

Now let's work out the first scheme. Although the left hand of the assignment is a variable from the users point of view, it is a constant in terms of the system. Both `width` and `breedte` expand to `width` because in the source we only encounter `width`. Such system constants are presented as

```
\c!width
```

This constant is always equivalent to `width`. As we can see, we use `c!` to mark this one as constant. Its dutch counterpart is:

```
\c!breedte
```

When we interpret a setup command each variable is translated to it's `c!` counterpart. This means that `breedte` and `width` expand to `\c!breedte` and `\c!width` which both expand to `width`. That way user variables become system constants.

The interpretation is done by means of a general setup command `\getparameters` that we introduced in the system module. Let us define some simple setup command:

```
\def\setupbox[#1]%
  {\getparameters[\??bx][#1]}
```

This command can be used as:

```
\setupbox [width=3cm, height=1cm]
```

Afterwards we have two variables `\@@bxwidth` and `\@@bxheight` which have the values `3cm` and `1cm` assigned. These variables are a combinatiom of the setup prefix `\??bx`, which expands to `@@bx` and the translated user supplied variables `width` and `height` or `breedte` and `hoogte`, depending on the actual language. In dutch we just say:

```
\stelblokin [breedte=3cm, hoogte=1cm]
```

and get ourselves `\@@bxwidth` and `\@@bxheight` too. In the source of CONTEXT, we can recognize constants and variables on their leading `c!`, `v!` etc., prefixes on `??` and composed variables on `@@`.

We already saw that user supplied keywords need some special treatment too. This time we don't translate the keyword, but instead use in the source a variable which meaning depends on the interface language.

```
\v!left
```

Which can be used in macro's like:

```
\processaction
  [\@@bxlocation]
  [  \v!left=>\dosomethingontheleft,
   \v!middle=>\dosomthinginthemiddle,
```

mult-ini
mult-sys
mult-con
mult-com

```
       \v!right=>\dosomethingontheright]
```

Because variables like `\@@bxlocation` can have a lot of meanings, including tricky expandable tokens, we cannot translate this meaning when we compare. This means that `\@@bxlocation` can be `left` of `links` of whatever meaning suits the language. But because `\v!left` also has a meaning that suits the language, we are able to compare.

Although we know it sounds confusing we want to state two important characteristics of the interface as described:

user variables become system constants

and

user constants (keywords) become system variables

\startconstants
\startvariables

It's time to introduce the macro's that are responsible for this translations process, but first we show how constants and variables are defined. We only show two languages and a few words.

```
   \startconstants  english    dutch

             width:  width      breedte
            height:  height     hoogte

   \stopconstants
```

Keep in mind that what users see as variables, are constants for the system.

```
   \startvariables  english    dutch

          location:  left       links
              text:  text       tekst
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
   \stopvariables
```

The macro's responsible for interpreting these setups are shared. They take care of empty lines and permit a more or less free format. All setups accept the keyword `all` which equals every language.

35
```
\def\nointerfaceobject{-}
```

36
```
\def\startinterfaceobjects#1#2%
  {\!!counta=1
   \let\dogetinterfaceobject=\dogetinterfacetemplate
   \let\dowithinterfaceelement=#1%
   \def\dodogetinterfaceobjects%
      {\ifx\next#2%
         \def\next####1%
           {}%
       \else\ifx\next\par
         \long\def\next####1%
           {\dogetinterfaceobjects}%
       \else\ifx\next\empty
         \def\next####1%
           {\dogetinterfaceobjects}%
       \else
         \def\next####1 %
           {\dogetinterfaceobject[####1:\relax]%
            \dogetinterfaceobjects}%
       \fi\fi\fi
       \next}%
   \def\dogetinterfaceobjects%
     {\futurelet\next\dodogetinterfaceobjects}%
   \dogetinterfaceobjects}
```

mult-ini
mult-sys
mult-con
mult-com

37
```
\def\dogetinterfacetemplate[#1:#2]%
  {\doifinsetelse{#1}{\currentinterface,all}
     {\let\dogetinterfaceobject=\doskipinterfaceobject}
     {\advance\!!counta by 1\relax}}
```

38
```
\def\doskipinterfaceobject[#1:#2#3]%
  {\if#2:%
     \let\dogetinterfaceobject=\dogetinterfaceelement
     \dogetinterfaceobject[#1:#2#3]%
   \fi}
```

39
```
\def\dogetinterfaceelement[#1:#2#3]%
  {\ifx#2:%
     \!!countb=0
     \def\!!stringa{#1}%
   \else
     \advance\!!countb by 1
     \ifnum\!!countb=\!!counta
       \@EA\dowithinterfaceelement\@EA{\!!stringa}{#1}%
       \let\dogetinterfaceobject=\doskipinterfaceobject
     \fi
   \fi}
```

The constants and variables are defined as described. When `\interfacetranslation` is `true`, we also generate a reverse translation. Because we don't want to put too big a burden on TEX's hash table, this is no default behavior. Reverse translation is used in the commands that generate the quick reference cards. We are going to define the real CONTEXT commands in an abstract way and generate those reference cards for each language without further interference.

40
```
\def\setinterfaceconstant#1#2%
  {\setvalue{\c!prefix!#1}{#1}%
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
    \doifelse{#2}{\nointerfaceobject}
      {\debuggerinfo{constant}{#1 defined as #1 by default}}
      {\debuggerinfo{constant}{#1 defined as #2}%
       \ifinterfacetranslation
         \setvalue{\x!prefix!#1}{#2}%
       \fi
       \setvalue{\c!prefix!#2}{#1}}}
```

41
```
\def\setinterfacevariable#1#2%
  {\doifelse{#2}{\nointerfaceobject}
      {\debuggerinfo{variable}{#1 defined as #1 by default}%
       \setvalue{\v!prefix!#1}{#1}}
      {\debuggerinfo{variable}{#1 defined as #2}%
       \setvalue{\v!prefix!#1}{#2}}}
```

42
```
\def\startvariables%
  {\startinterfaceobjects\setinterfacevariable\stopvariables}
```

43
```
\def\startconstants%
  {\startinterfaceobjects\setinterfaceconstant\stopconstants}
```

\startinterfacesetupco..

The next command, `\startinterfacesetupconstant`, which behavior also depends on the boolean, is used for constants that are only needed in these quick reference macro's. The following, more efficient approach does not work here, because it sometimes generates spaces.

```
    \def\setinterfacesetupconstant%
      {\ifinterfacetranslation
         \expandafter\setinterfaceconstant
       \fi}
```

We therefore use the more redundant but robust method:

mult-ini
mult-sys
mult-con
mult-com

44
```
\def\setinterfacesetupvariable#1#2%
  {\ifinterfacetranslation
     \doifelse{#2}{\nointerfaceobject}
       {\setvalue{\y!prefix!#1}{#1}}
       {\setvalue{\y!prefix!#1}{#2}}%
   \fi}
```

45
```
\def\startsetupvariables%
  {\startinterfaceobjects\setinterfacesetupvariable\stopsetupvariables}
```

\startelements

Due to the object oriented nature of CONTEXT, we also need to define the elements that are used to build commands:

```
   \startelements  english    dutch

         beginof:  beginof    beginvan
          eindof:  endof      eindvan
           start:  start      start
            stop:  stop       stop

   \stopelements
```

Such elements sometimes are the same in diferent languages, but mostly they differ. Things can get even confusing when we look at for instance the setup commands. In english we say `\setup<something>`, but in dutch we have: `\stel<iets>in`. Such split elements are no problem, because we just define two elements. When no second part is needed, we use a `-`:

```
   \startelements  english    dutch

          setupa:  setup      stel
          setupb:  -          in
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

mult-ini
mult-sys
mult-con
mult-com

```
   \stopelements
```

Element translation is realized by means of:

46
```
\def\setinterfaceelement#1#2%
  {\doifelse{#2}{\nointerfaceobject}
     {\debuggerinfo{element}{#1 defined as <empty>}%
      \setvalue{\e!prefix!#1}{}}
     {\doifdefinedelse{\e!prefix!#1}
       {\doifnot{\getvalue{\e!prefix!#1}}{#2}
          {\debuggerinfo{element}{#1 redefined as #2}%
           \setvalue{\e!prefix!#1}{#2}}}
       {\debuggerinfo{element}{#1 defined as #2}%
        \setvalue{\e!prefix!#1}{#2}}}}
```

47
```
\def\startelements%
  {\startinterfaceobjects\setinterfaceelement\stopelements}
```

\startcommands The last setup has to do with the commands themselve. Commands are defined as:

```
   \startcommands  english     dutch

      starttekst:  starttext   starttekst
       stoptekst:  stoptext    stoptekst
         omlijnd:  framed      omlijnd
     margewoord:  marginword  margewoord

   \stopcommands
```

Here we also have to take care of the optional translation needed for reference cards.

48
```
\def\setinterfacecommand#1#2%
  {\doifelse{#2}{\nointerfaceobject}
     {\debuggerinfo{command}{no link to #1}%
      \setinterfacesetupvariable{#1}{#1}}
     {\doifelse{#1}{#2}
        {\debuggerinfo{command}{#1 remains #1}}
        {\doifdefinedelse{#2}
           {\debuggerinfo{command}{core command #2 redefined as #1}}%
           {\debuggerinfo{command}{#2 defined as #1}}%
         \@EA\@EA\@EA\def\@EA\csname\@EA#2\@EA\endcsname
           \@EA{\csname#1\endcsname}}%
      \setinterfacesetupvariable{#1}{#2}}}
```

49
```
\def\startcommands%
  {\startinterfaceobjects\setinterfacecommand\stopcommands}
```

\getinterfaceconstant
\getinterfacevariable

Generating the interface translation macro's that are used in the reference lists, is enabled by setting the boolean:

```
   \interfacetranslationtrue
```

Keep in mind that enabling interfacetranslation costs a bit of hash space.

50
```
\newif\ifinterfacetranslation
```

51
```
\def\getinterfaceconstant#1%
  {\ifinterfacetranslation
     \doifdefinedelse{\x!prefix!#1}
       {\getvalue{\x!prefix!#1}}
       {#1}%
   \else
```

mult-ini
mult-sys
mult-con
mult-com

```
    #1%
  \fi}
```

52
```
\def\getinterfacevariable#1%
  {\ifinterfacetranslation
     \doifdefinedelse{\y!prefix!#1}
       {\getvalue{\y!prefix!#1}}
       {#1}%
   \else
     #1%
   \fi}
```

When a reference list is generated, one does not need to generate a new format. Just reloading the relevant definition files suits:

```
\interfacetranslationtrue
\input mult-con
\input mult-com
```

\interfaced The setup commands translate the constants automatically. When we want to translate 'by hand' we can use the simple but effective command:

```
\interfaced {something}
```

Giving `\interfaced{breedte}` results in `width` or, when not defined, in `breedte` itself.

53
```
\def\interfaced#1%
  {\expandafter\ifx\csname\c!prefix!#1\endcsname\relax
     #1%
   \else
     \csname\c!prefix!#1\endcsname
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
    \fi}
```

So much for the basic multi–lingual interface commands. The macro's can be enhanced with more testing facilities, but for the moment they suffice.

54
```
\protect
```

mult-ini
mult-sys
mult-con
mult-com

contents register context syst mult supp lang font colo spec core cont m s exit go back

3.2 System

mult-ini
mult-sys
mult-con
mult-com

In boring module we define a lot of obscure but useful system constants. By doing so we save lots of memory while at the same time we prevent ourself from typing errors.

```
\writestatus{loading}{Context Multilingual Macros / System}
```

```
\unprotect
```

The constants are grouped in such a way that there is a minimal change of conflicts.

```
   \definesystemconstants {word}
   \definemessageconstant {word}
```

This commands generate `\s!word` and `\m!word`.

First we define some system constants used for both the multi–lingual interface and multi–linguag typesetting.

```
\definesystemconstant  {dutch}
\definesystemconstant  {english}
\definesystemconstant  {french}
\definesystemconstant  {german}
\definesystemconstant  {spanish}
```

As the name of their define command states, the next set of constants is used in the message macro's.

```
\definemessageconstant {colors}
\definemessageconstant {columns}
\definemessageconstant {figures}
\definemessageconstant {floatblocks}
\definemessageconstant {fonts}
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
\definemessageconstant {interactions}
\definemessageconstant {layouts}
\definemessageconstant {linguals}
\definemessageconstant {references}
\definemessageconstant {specials}
\definemessageconstant {structures}
\definemessageconstant {systems}
\definemessageconstant {textblocks}
\definemessageconstant {versions}
```

The word `height` takes 6 token memory cells. The control sequence `\height` on the other hand uses only one. Knowing this, we can improve the performance of TEX, both is terms of speed and memory usage, by using control sequences instead of the words written in full.

Where in the ASCII file the second lines takes nine extra characters, TEX saves us 13 tokens.

```
   \hrule width 10pt height 2pt depth 1pt
   \hrule \!!width 10pt \!!height 2pt \!!depth 1pt
```

One condition is that we have defined `\!!height`, `\!!width` and `\!!depth` as respectively `height`, `width` and `depth`. Using this scheme therefore only makes sense when a token sequence is used more than once. Savings like this should of course be implemented in english, just because TEX is english.

5
```
\def\!!width  {width}
\def\!!height {height}
\def\!!depth  {depth}
```

6
```
\def\!!plus   {plus}
\def\!!minus  {minus}
```

mult-ini
mult-sys
mult-con
mult-com

7
```
\def\!!fill   {fill}
```

The same goes for some CONTEXT constants, used in the definition of private commands:

8
```
\definesystemconstant  {next}
\definesystemconstant  {pickup}
```

9
```
\definesystemconstant  {default}
\definesystemconstant  {unknown}
```

10
```
\definesystemconstant  {action}
\definesystemconstant  {compare}
```

11
```
\definesystemconstant  {do}
\definesystemconstant  {dodo}
```

12
```
\definesystemconstant  {complex}
\definesystemconstant  {simple}
```

13
```
\definesystemconstant  {start}
\definesystemconstant  {stop}
```

14
```
\definesystemconstant  {dummy}
```

15
```
\definesystemconstant  {local}
\definesystemconstant  {global}
```

16
```
\definesystemconstant  {done}
```

A more experienced TEX user will recognize the next four constants. We need these because font-definitions are partially english.

contents register context syst mult supp lang font colo spec core cont m s exit go back

mult-ini
mult-sys
mult-con
mult-com

17
```
\definesystemconstant  {fam}
\definesystemconstant  {text}
\definesystemconstant  {script}
\definesystemconstant  {scriptscript}
```

Just to be complete we define the standard TEX units.

18
```
\definesystemconstant  {cm}
\definesystemconstant  {em}
\definesystemconstant  {ex}
\definesystemconstant  {mm}
\definesystemconstant  {pt}
\definesystemconstant  {sp}
\definesystemconstant  {in}
```

These constants are used for internal and utility commands.

19
```
\definesystemconstant  {check}
\definesystemconstant  {reset}
\definesystemconstant  {set}
```

20
```
\definesystemconstant  {entrya}
\definesystemconstant  {entryb}
\definesystemconstant  {entryc}
\definesystemconstant  {entry}
\definesystemconstant  {see}
\definesystemconstant  {page}
\definesystemconstant  {line}
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

mult-ini
mult-sys
mult-con
mult-com

21
```
\definesystemconstant  {synonym}
```

22
```
\definesystemconstant  {reference}
\definesystemconstant  {main}
```

23
```
\definesystemconstant  {list}
```

24
```
\definesystemconstant  {item}
\definesystemconstant  {itemcount}
```

25
```
\definesystemconstant  {number}
\definesystemconstant  {references}
\definesystemconstant  {between}
\definesystemconstant  {format}
\definesystemconstant  {old}
```

26
```
\definesystemconstant  {thisisblock}
\definesystemconstant  {thiswasblock}
```

27
```
\definesystemconstant  {figurepreset}
```

Some CONTEXT commands take a two–pass aproach to optimize the typesetting. Each two–pass object has its own tag.

28
```
\definesystemconstant  {pass}
```

29
```
\definesystemconstant  {float}
\definesystemconstant  {list}
\definesystemconstant  {page}
\definesystemconstant  {subpage}
\definesystemconstant  {margin}
\definesystemconstant  {profile}
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

mult-ini
mult-sys
mult-con
mult-com

```
\definesystemconstant  {versionbegin}
\definesystemconstant  {versionend}
\definesystemconstant  {cross}
\definesystemconstant  {paragraph}
```

A lot of macros use tags to distinguish between different objects, e.g. lists and registers.

30
```
\definesystemconstant  {prt}  % part (deel)
\definesystemconstant  {chp}  % chapter (hoofdstuk)
\definesystemconstant  {sec}  % section (paragraaf)
\definesystemconstant  {tit}  % title (titel)
\definesystemconstant  {sub}  % subject (onderwerp)
\definesystemconstant  {mar}  % margin (marge)
\definesystemconstant  {num}  % number (doornummeren)
\definesystemconstant  {def}  % definition (doordefinieren)
\definesystemconstant  {for}  % formula (formule)
\definesystemconstant  {fnt}  % footnote (voetnoot)
\definesystemconstant  {ind}  % index (register)
\definesystemconstant  {lin}  % linked index
\definesystemconstant  {lst}  % list (opsomming)
\definesystemconstant  {flt}  % float (plaatsblok)
\definesystemconstant  {pag}  % page (pagina)
\definesystemconstant  {txt}  % text (tekst)
\definesystemconstant  {ref}  % reference (verwijzing)
\definesystemconstant  {lab}  % label (label)
\definesystemconstant  {aut}  % automatic (inhoud, index)
```

31
```
\definesystemconstant  {kop}  % kop  % still dutch
```

Reference labels can be tagged by users, for instance by means of `tag:`. The reference mechanism itself uses some tags too. These are definitely not to be used by users. Here they are:

32
```
\definereferenceconstant {cross}   {:c:} % cross reference
\definereferenceconstant {view}    {:v:} % view reference
\definereferenceconstant {viewa}   {:a:} % view reference test a
\definereferenceconstant {viewb}   {:b:} % view reference test b
\definereferenceconstant {page}    {:p:} % page referece
\definereferenceconstant {list}    {:l:} % list reference
\definereferenceconstant {exec}    {:e:} % execution reference
```

33
```
\definereferenceconstant {from}    {:f:} % from list reference
\definereferenceconstant {to}      {:t:} % to list reference
```

When we use numbers and dimensions the same applies as with the keywords like `width` and `plus` mentioned earlier.

34
```
\def\!!ten             {10}
\def\!!twelve          {12}
\def\!!hundred        {100}
\def\!!thousand      {1000}
\def\!!tenthousand  {10000}
\def\!!maxcard      {65536}
\def\!!medcard      {32768}
```

35
```
\def\!!zeropoint    {0pt}
\def\!!onepoint     {1pt}
\def\!!twopoint     {2pt}
\def\!!threepoint   {3pt}
\def\!!fourpoint    {4pt}
\def\!!fivepoint    {5pt}
\def\!!sixpoint     {6pt}
\def\!!sevenpoint   {7pt}
\def\!!eightpoint   {8pt}
```

```
\def\!!ninepoint    {9pt}
\def\!!tenpoint    {10pt}
\def\!!elevenpoint {11pt}
\def\!!twelvepoint {12pt}
```

Variables are composed of a command specific tag and a user supplied variable (system constant). The first tag `ag` for instance is available as `\??ag` and expands to `@@ag` in composed variables.

36

```
\definesystemvariable {ag}   % AchterGrond
\definesystemvariable {al}   % ALinea's
\definesystemvariable {am}   % interActieMenu
\definesystemvariable {ba}   % synchronisatieBAlk
\definesystemvariable {be}   % startstop (BeginEnd)
\definesystemvariable {bj}   % BlokJe
\definesystemvariable {bk}   % Blokken (floats)
\definesystemvariable {bl}   % BLanko
\definesystemvariable {bs}   % SelecteerBlokken
\definesystemvariable {bt}   % BuTton
\definesystemvariable {bu}   % BUffer
\definesystemvariable {bv}   % Brieven
\definesystemvariable {by}   % Per
\definesystemvariable {ci}   % CItaat
\definesystemvariable {cl}   % kleur (CoLor setup)
\definesystemvariable {co}   % COmbinaties
\definesystemvariable {cr}   % kleur (ColoR)
\definesystemvariable {cv}   % ConVersie
\definesystemvariable {dd}   % DoorDefinieren
\definesystemvariable {de}   % DEel
\definesystemvariable {dl}   % DunneLijnen
\definesystemvariable {dn}   % DoorNummeren
```

```
\definesystemvariable {do}   % DefinieerOpmaak
\definesystemvariable {ds}   % DoorSpringen
\definesystemvariable {ef}   % ExternFiguur
\definesystemvariable {ep}   % ExternfiguurPreset
\definesystemvariable {ex}   % ExterneFiguren
\definesystemvariable {fg}   % FiGuurmaten
\definesystemvariable {fi}   % FIle
\definesystemvariable {fl}   % Floats
\definesystemvariable {fm}   % ForMules
\definesystemvariable {ft}   % FonTs
\definesystemvariable {fp}   % FilegroeP
\definesystemvariable {ia}   % Interactie
\definesystemvariable {ib}   % InteractieBalk
\definesystemvariable {id}   % Index
\definesystemvariable {ih}   % InHoudsopgave
\definesystemvariable {ii}   % stelIndexIn
\definesystemvariable {il}   % stelInvulRegelsin
\definesystemvariable {im}   % InMarge
\definesystemvariable {in}   % INspringen
\definesystemvariable {is}   % Items
\definesystemvariable {iv}   % stelInvulLijnenin
\definesystemvariable {ka}   % KAntlijn
\definesystemvariable {kd}   % KaDerteksten
\definesystemvariable {kj}   % KopJes (floats)
\definesystemvariable {kl}   % KoLommen
\definesystemvariable {km}   % KenMerk
\definesystemvariable {ko}   % KOp(pen)
\definesystemvariable {kp}   % KopPelteken
\definesystemvariable {kr}   % KoRps
```

```
\definesystemvariable {kt}   % KonTakten
\definesystemvariable {kw}   % KontaktWaarde
\definesystemvariable {la}   % LAnguage
\definesystemvariable {lg}   % taal (LanGuage)
\definesystemvariable {li}   % LIjst
\definesystemvariable {ln}   % LijNen
\definesystemvariable {lo}   % LOgos
\definesystemvariable {lt}   % LiTeratuur
\definesystemvariable {ly}   % LaYout
\definesystemvariable {ma}   % MargeAchtergrond
\definesystemvariable {mb}   % MargeBlokken
\definesystemvariable {mk}   % MarKering
\definesystemvariable {nm}   % Nummering
\definesystemvariable {np}   % NaastPlaatsen
\definesystemvariable {nr}   % Nummeren
\definesystemvariable {oi}   % OmlijndInstellingen
\definesystemvariable {ol}   % OmLijnd
\definesystemvariable {oo}   % OpsOmmingen
\definesystemvariable {op}   % OPsomming
\definesystemvariable {os}   % OffSet
\definesystemvariable {pa}   % PAlet
\definesystemvariable {pb}   % PuBlicatie
\definesystemvariable {pf}   % ProFiel
\definesystemvariable {pg}   % KoppelPagina
\definesystemvariable {pn}   % PaginaNummer
\definesystemvariable {pp}   % PaPier
\definesystemvariable {pr}   % PRogrammas
\definesystemvariable {ps}   % PoSitioneren
\definesystemvariable {rf}   % ReFereren
```

```
\definesystemvariable {rg}   % ReGel
\definesystemvariable {rl}   % ReferentieLijst
\definesystemvariable {rn}   % RegelNummer
\definesystemvariable {ro}   % ROteren
\definesystemvariable {rs}   % RaSters
\definesystemvariable {rt}   % RoosTers
\definesystemvariable {rv}   % ReserVeerfiguur
\definesystemvariable {sb}   % SectieBlok
\definesystemvariable {sc}   % SCherm
\definesystemvariable {se}   % SEctie
\definesystemvariable {sk}   % SectieKop
\definesystemvariable {sl}   % SmalLer
\definesystemvariable {sm}   % SynonieMen
\definesystemvariable {sn}   % SubNummer
\definesystemvariable {so}   % SOrteren
\definesystemvariable {sp}   % SelecteerPapier
\definesystemvariable {ss}   % Symbool
\definesystemvariable {st}   % STickers
\definesystemvariable {sv}   % SysteemVariabelen
\definesystemvariable {sy}   % SYnchronisatie
\definesystemvariable {ta}   % TAb
\definesystemvariable {tb}   % TekstBlokken
\definesystemvariable {ti}   % TabelInstellingen
\definesystemvariable {tk}   % Teksten
\definesystemvariable {tl}   % TekstLijnen
\definesystemvariable {tp}   % TyPen
\definesystemvariable {tu}   % TabUlatie
\definesystemvariable {ty}   % TYpe
\definesystemvariable {ve}   % VErsie
```

mult-ini
mult-sys
mult-con
mult-com

contents register context syst mult supp lang font colo spec core cont m s exit go back

mult-ini
mult-sys
mult-con
mult-com

```
\definesystemvariable {vn}   % VoetNoten
\definesystemvariable {wr}   % WitRuimte
\definesystemvariable {za}   % ZetspiegelAanpassing
```

Next we define some language independant one letter variables and keywords.

37
```
\defineinterfaceconstant {x} {x}  % x offset
\defineinterfaceconstant {y} {y}  % y offset
\defineinterfaceconstant {w} {w}  % width
\defineinterfaceconstant {h} {h}  % height
\defineinterfaceconstant {s} {s}  % size
\defineinterfaceconstant {t} {t}  % title
\defineinterfaceconstant {c} {c}  % creator
\defineinterfaceconstant {e} {e}  % extension
\defineinterfaceconstant {f} {f}  % file
```

38
```
\defineinterfaceconstant {a} {a}  % kunnen weg
\defineinterfaceconstant {b} {b}  % kunnen weg
\defineinterfaceconstant {c} {c}  % kunnen weg
\defineinterfaceconstant {d} {d}  % kunnen weg
\defineinterfaceconstant {e} {e}  % kunnen weg
```

39
```
\defineinterfaceconstant {r} {r}
\defineinterfaceconstant {g} {g}
\defineinterfaceconstant {b} {b}
\defineinterfaceconstant {c} {c}
\defineinterfaceconstant {m} {m}
\defineinterfaceconstant {y} {y}
\defineinterfaceconstant {k} {k}
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

40
```
\defineinterfaceconstant {t} {t}
\defineinterfaceconstant {h} {h}
\defineinterfaceconstant {b} {b}
```

41
```
\defineinterfaceconstant {rgb}  {rgb}
\defineinterfaceconstant {cmyk} {cmyk}
```

42
```
\defineinterfacevariable {rgb}  {rgb}
\defineinterfacevariable {cmyk} {cmyk}
\defineinterfacevariable {gray} {k}
```

The names of files and their extensions are fixed. CONTEXT uses as less files as possible. Utility files can be recognized by the first two characters of the extension: `tu`.

43
```
\definefileconstant {utilityfilename}    {texutil}
```

44
```
\definefileconstant {blockextension}     {tub}
\definefileconstant {figureextension}    {tuf}
\definefileconstant {inputextension}     {tui}
\definefileconstant {outputextension}    {tuo}
\definefileconstant {temporaryextension} {tmp}
\definefileconstant {patternsextension}  {pat}
\definefileconstant {hyphensextension}   {hyp}
```

These files are loaded at start–up. They may contain system specific setups (or calls to other files), old macro's, to garantee compatibility and new macro's noy yet present in the format.

45
```
\definefileconstant {systemfilename}     {cont-sys}
\definefileconstant {oldfilename}        {cont-old}
\definefileconstant {newfilename}        {cont-new}
```

The setup files for the language, font, color and special subsystems have a common prefix. This means that we have at most three characters for unique filenames.

46
```
\definefileconstant {languageprefix}     {lang-}
\definefileconstant {fontprefix}         {font-}
\definefileconstant {colorprefix}        {colo-}
\definefileconstant {specialprefix}      {spec-}
```

CONTEXT follows different strategies for finding files. The macros that are responsible for this 'clever' searching make use of two (very important) path specifiers.

47
```
\definefileconstant {currentpath}        {.}
\definefileconstant {parentpath}         {..}
```

The way fonts are defined and called upon is language independant. We follow the scheme laid down by Knuth in Plain TEX. We'll explain their meaning later.

48
```
\defineinterfaceconstant {tf} {tf}
\defineinterfaceconstant {bf} {bf}
\defineinterfaceconstant {bs} {bs}
\defineinterfaceconstant {bi} {bi}
\defineinterfaceconstant {sl} {sl}
\defineinterfaceconstant {it} {it}
\defineinterfaceconstant {sc} {sc}
\defineinterfaceconstant {rm} {rm}
\defineinterfaceconstant {ss} {ss}
\defineinterfaceconstant {tt} {tt}
\defineinterfaceconstant {hw} {hw}
\defineinterfaceconstant {cg} {cg}
\defineinterfaceconstant {os} {os}
\defineinterfaceconstant {mm} {mm}
```

```
\defineinterfaceconstant {i}  {i}
```

49
```
\defineinterfaceconstant {x}  {x}
\defineinterfaceconstant {xx} {xx}
```

50
```
\defineinterfaceconstant {mi} {mi}
\defineinterfaceconstant {sy} {sy}
\defineinterfaceconstant {ex} {ex}
\defineinterfaceconstant {mr} {mr}
```

51
```
\defineinterfaceconstant {ma} {ma}
\defineinterfaceconstant {mb} {mb}
\defineinterfaceconstant {mc} {mc}
```

Finally we need:

52
```
\defineinterfaceconstant {tif}  {tif}
\defineinterfaceconstant {eps}  {eps}
```

A careful reader will have noticed that in the module `mult-ini` we defined `\selectinterface`. We were not yet able to actually select an interface, because we still had to define the constants and variables. Now we've done so, selection is permitted.

53
```
\selectinterface
```

And only after this selection is done, we can define messages, otherwise the default language is in use.

54
```
\startmessages  dutch  library: check
  title: controle
      1: '=' ontbreekt na '--' in regel --
      2: -- argument(en) verwacht in regel --
      3: -- -- vervangt een macro, gebruik HOOFDLETTERS!
\stopmessages
```

55
```
\startmessages  english library: check
  title: check
      1: missing '=' after '--' in line --
      2: -- argument(s) expected in line --
      3: -- -- replaces a macro, use CAPITALS!
\stopmessages
```

56
```
\startmessages  german library: check
  title: check
      1: Fehlendes '=' nach '--' in Zeile --
      2: -- Argument(e) in Zeile -- erwartet
      3: -- -- ersetzt ein Makro, verwende VERSALIEN!
\stopmessages
```

Ok, here are some more, because we've got ouselves some extensions to CONTEXT.

57
```
\definemessageconstant {addresses}
\definemessageconstant {documents}
```

58
```
\protect
```

mult-ini
mult-sys
mult-con
mult-com

3.3 Constants

In this rather large definition file we are going to tell CONTEXT which constants, variables and elements we use.

```
\writestatus{loading}{Context Multilingual Macros / Constants}
```

CONTEXT supports language specific typesetting (such as hyphenation). Switching from one to another language as well as specifying for instance language dependant labels is done by means of similar keywords and commands. For to historical reasons we support more than one naming scheme.

```
\startconstants      dutch                english           german

                nl:  nl                   nl                nl
                en:  en                   en                en
                fa:  fa                   fr                fr
                du:  du                   ge                de
                sp:  sp                   sp                sp

\stopconstants
```

By defining them as constants, we can use them in the left hand part of an assignment.

Next come some variables. These are used as keywords and therefore need a different treatment.

```
\startvariables      dutch                english            german

              een:   een                  one                eins
             twee:   twee                 two                zwei
             drie:   drie                 three              drei
             vier:   vier                 four               vier
             vijf:   vijf                 five               fuenf

\stopvariables
```

Of course we need the names of the months.

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
\startvariables  dutch       english     german

        january:  januari     January     Januar
       february:  februari    February    Februar
          march:  maart       March       Maerz
          april:  april       April       April
            may:  mei         May         Mai
           june:  juni        June        Juni
           july:  juli        July        Juli
         august:  augustus    August      August
      september:  september   September   September
        october:  oktober     October     Oktober
       november:  november    November    November
       december:  december    December    Dezember

\stopvariables
```

User defined commands are language specific, so we have to use variable when defining them. First we define some general structuring variables:

```
\startvariables       dutch                english             german

           sectie:    sectie               section             abschnitt

             deel:    deel                 part                teil
        hoofdstuk:    hoofdstuk            chapter             kapitel
        onderwerp:    onderwerp            subject             thema
        paragraaf:    paragraaf            paragraph           absatz
            titel:    titel                title               titel

           inhoud:    inhoud               content             inhalt
         inhouden:    inhouden             contents            inhalte

          bijlage:    bijlage              appendix            anhang
         bijlagen:    bijlagen             appendices          anhaenge
       hoofdtekst:    hoofdtekst           maintext            haupttext
     hoofdteksten:    hoofdteksten         maintexts           haupttexte
        inleiding:    inleiding            introduction        einleitung
      inleidingen:    inleidingen          introductions       einleitungen
```

```
       uitleiding:  uitleiding         extroduction        epilog
     uitleidingen:  uitleidingen       extroductions       epiloge

         voetnoot:  voetnoot           footnote            fussnote

          systeem:  systeem            system              systeme

\stopvariables

\startvariables     dutch              english             german

            typen:  typen              typing              tippen
             file:  file               file                datei

\stopvariables
```

As we can see below, there are some more variables needed, for instance for the definition of macro's for handling floating bodies.

```
\startvariables     dutch              english             german

          formule:  formule            formula             formel
         formules:  formules           formulas            formeln

           figuur:  figuur             figure              abbildung
          figuren:  figuren            figures             abbildungen
            tabel:  tabel              table               tabelle
         tabellen:  tabellen           tables              tabellen
          grafiek:  grafiek            graphic             grafik
        grafieken:  grafieken          graphics            grafiken
       intermezzo:  intermezzo         intermezzo          intermezzo
      intermezzos:  intermezzos        intermezzos         intermezzi

            index:  index              index               index
          indices:  indices            indices             indizies

        afkorting:  afkorting          abbreviation        abkuerzung
      afkortingen:  afkortingen        abbreviations       abkuerzungen
             logo:  logo               logo                logo
            logos:  logos              logos               logos
```

mult-ini
mult-sys
mult-con
mult-com

```
           eenheid:  eenheid             unit                einheit
          eenheden:  eenheden            units               einheiten

             regel:  regel               line                zeile
            regels:  regels              lines               zeilen

\stopvariables
```

The setup commands can take a lot of different arguments, often in the form `variable=value`. Here we define the variable part. Keep in mind that for the system, user defined variables have a constant character.

```
\startconstants      dutch               english             german

      leftquotation:  linkercitaat        leftquotation       linkerzitat
     rightquotation:  rechtercitaat       rightquotation      rechterzitat
          leftquote:  linkerciteer        leftquote           linkerzitieren
         rightquote:  rechterciteer       rightquote          rechterzitieren
       leftsentence:  linkerzin           leftsentence        linkersatz
      rightsentence:  rechterzin          rightsentence       rechtersatz
    leftsubsentence:  linkersubzin        leftsubsentence     linkersubsatz
   rightsubsentence:  rechtersubzin       rightsubsentence    rechtersubsatz

              datum:  datum               date                datum

                aan:  aan                 to                  zu
         aanduiding:  aanduiding          indicator           indikator
        achtergrond:  achtergrond         background          hintergrund
   achtergrondkleur:  achtergrondkleur    backgroundcolor     hintergrundfarbe
  achtergrondoffset:  achtergrondoffset   backgroundoffset    hintergrundoffset
    achtergrondhoek:  achtergrondhoek     backgroundangle     hintergrundwinkel
  achtergrondstraal:  achtergrondstraal   backgroundradius    hintergrundradius
  achtergronddiepte:  achtergronddiepte   backgrounddepth     hintergrundtiefe
  achtergrondraster:  achtergrondraster   backgroundscreen    hintergrundraster
              adres:  adres               address             adresse
           afmeting:  afmeting            size                abmessung
          afsluiter:  afsluiter           stopper             abschnitttrenner
            afstand:  afstand             distance            abstand
        assenstelsel:  assenstelsel       axis                achsen
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

mult-ini
mult-sys
mult-con
mult-com

auteur:	auteur	author	autor
balanceren:	balanceren	balance	ausgleichen
bfactor:	bfactor	wfactor	bfaktor
binnen:	binnen	inner	innen
bladzijde:	bladzijde	page	seite
blanko:	blanko	blank	blanko
blokkade:	blokkade	obstruction	gesperrt
blokwijze:	blokwijze	blockway	blockauf
boven:	boven	top	oben
bovenafstand:	bovenafstand	topdistance	obenabstand
bovenkader:	bovenkader	topframe	obenrahmen
bovenoffset:	bovenoffset	topoffset	obenoffset
bovenstatus:	bovenstatus	topstatus	statusoben
breedte:	breedte	width	breite
bron:	bron	source	quelle
commando:	commando	command	befehl
commandos:	commandos	commands	befehle
conversie:	conversie	conversion	konversion
correctie:	correctie	correction	korrektur
criterium:	criterium	criterium	kriterium
datum:	datum	date	datum
default:	default	default	norm
diepte:	diepte	depth	tiefe
dikte:	dikte	thickness	dicke
doorgaan:	doorgaan	continue	fortsetzten
dubbelzijdig:	dubbelzijdig	doublesided	doppelseitig
dummy:	dummy	dummy	dummy
eenheid:	eenheid	unit	einheit
contrastkleur:	contrastkleur	contrastcolor	--farbe
expansie:	expansie	expansion	expansion
factor:	factor	factor	faktor
file:	file	file	datei
formaat:	formaat	size	groesse
gebied:	gebied	directory	verzeichnis
groot:	groot	big	gross
haal:	haal	get	hole
hang:	hang	hang	haengend
hfactor:	hfactor	hfactor	hfaktor

contents register context syst mult supp lang font colo spec core cont m s exit go back

hoek:	hoek	angle	winkel
hoffset:	hoffset	hoffset	hoffset
hokjes:	hokjes	frames	umrahmen
hoofd:	hoofd	header	kopfzeile
hoofdafstand:	hoofdafstand	headerdistance	kopfzeilenabstand
hoofdstatus:	hoofdstatus	headerstatus	kopfzeilenstatus
hoogte:	hoogte	height	hoehe
huidige:	huidige	current	aktuell
in:	in	in	in
interactie:	interactie	interaction	interaktion
inspringen:	inspringen	indenting	einziehen
items:	items	items	posten
kader:	kader	frame	rahmen
kaderkleur:	kaderkleur	framecolor	rahmenfarbe
kaderoffset:	kaderoffset	frameoffset	rahmenoffset
kaderdiepte:	kaderdiepte	framedepth	rahmentiefe
kaderhoek:	kaderhoek	frameangle	rahmenwinkel
kaderstraal:	kaderstraal	frameradius	rahmenradius
kantlijn:	kantlijn	marginedge	marginal
kantlijntekst:	kantlijntekst	marginedgetext	marginaltext
klein:	klein	small	klein
kleur:	kleur	color	farbe
kolom:	kolom	column	spalte
kopkleur:	kopkleur	headcolor	kopffarbe
kopletter:	kopletter	headstyle	kopfschrift
kopna:	kopna	afterhead	nachkopf
kopoffset:	kopoffset	topoffset	kopfoffset
koppeling:	koppeling	coupling	verknuepfung
kopvoor:	kopvoor	beforehead	vorkopf
kopwit:	kopwit	topspace	kopfspatium
korps:	korps	corps	fliesstext
label:	label	label	label
letter:	letter	style	schrift
lijn:	lijn	line	linie
lijndikte:	lijndikte	linethickness	liniendicke
lijst:	lijst	list	liste
linker:	linker	left	linker
linkerbreedte:	linkerbreedte	leftwidth	linkerbreite

mult-ini
mult-sys
mult-con
mult-com

contents register context syst mult supp lang font colo spec core cont m s exit go back

mult-ini
mult-sys
mult-con
mult-com

linkerkader:	linkerkader	leftframe	linkerrahmen
linkerkleur:	linkerkleur	leftcolor	linkerfarbe
linkerletter:	linkerletter	leftstyle	linkerschrift
linkermarge:	linkermarge	leftmargin	linkerrand
linkermargeafstand:	linkermargeafstand	leftmargindistance	linkerrandabstand
linkeroffset:	linkeroffset	leftoffset	linkeroffset
linkerrand:	linkerrand	leftedge	linkekante
linkerrandafstand:	linkerrandafstand	leftedgedistance	linkerkantenabstand
linkertekst:	linkertekst	lefttext	linkertext
links:	links	left	links
logo:	logo	logo	logo
logos:	logos	logos	logos
lokaal:	lokaal	local	lokal
lokale:	lokale	local	lokal
lokatie:	lokatie	location	position
marge:	marge	margin	marginalie
margeafstand:	margeafstand	margindistance	marginalabstand
margetekst:	margetekst	margintext	marginaltext
markering:	markering	mark	beschriftung
markleur:	markleur	marcolor	beschrfarbe
marletter:	marletter	marstyle	beschrschrift
menu:	menu	menu	menue
methode:	methode	method	methode
midden:	midden	middle	mittig
middentekst:	middentekst	middletext	mittigertext
min:	min	min	min
monster:	monster	sample	muster
na:	na	after	nach
naam:	naam	name	name
nacommando:	nacommando	commandafter	zumbefehl
nboven:	nboven	ntop	noben
niveau:	niveau	level	niveau
niveaus:	niveaus	levels	niveaus
nonder:	nonder	nbottom	nunten
norm:	norm	norm	norm
nregels:	nregels	nlines	zzeile
nummer:	nummer	number	nummer
nummercommando:	nummercommando	numbercommand	nummerbefehl

mult-ini
mult-sys
mult-con
mult-com

nummeren:	nummeren	numbering	nummerierung
nummerkleur:	nummerkleur	numbercolor	nummernfarbe
nummerletter:	nummerletter	numberstyle	nummernschrift
nummerscheider:	nummerscheider	numberseparator	nummernseperator
offset:	offset	offset	offset
omvang:	omvang	size	umfang
onbekendeverwijzing:	verwijzing	unknownreference	unbekantereferenz
onder:	onder	bottom	unten
onderafstand:	onderafstand	bottomdistance	untenabstand
onderkader:	onderkader	bottomframe	untenrahmen
onderoffset:	onderoffset	bottomoffset	untenoffset
onderstatus:	onderstatus	bottomstatus	untenstatus
op:	op	at	bei
optie:	optie	option	option
pagina:	pagina	page	seite
paginacommando:	paginacommando	pagecommand	seitenbefehl
paginakleur:	paginakleur	pagecolor	seitenfarbe
paginaletter:	paginaletter	pagestyle	seitenschrift
paginanummer:	paginanummer	pagenumber	seitennummer
paginaovergangen:	paginaovergangen	pageboundaries	seitenbegrenzung
papier:	papier	paper	papier
plaats:	plaats	location	platz
plaatsafsluiter:	plaatsafsluiter	placestopper	setzetrenner
plaatsen:	plaatsen	place	plaziere
plaatskop:	plaatskop	placehead	setzekopf
prefix:	prefix	prefix	prefix
preset:	preset	preset	voreinstellung
preview:	preview	preview	preview
punt:	punt	period	punkt
rand:	rand	edge	kante
randafstand:	randafstand	edgedistance	kantenabstand
raster:	raster	screen	raster
rechter:	rechter	right	rechter
rechterbreedte:	rechterbreedte	rightwidth	rechterbreite
rechterkader:	rechterkader	rightframe	rechterrahmen
rechterkleur:	rechterkleur	rightcolor	rechterfarbe
rechterletter:	rechterletter	rightstyle	rechterschrift
rechtermarge:	rechtermarge	rightmargin	rechterrand

contents register context syst mult supp lang font colo spec core cont m s exit go back

rechtermargeafstand:	rechtermargeafstand	rightmargindistance	rechterrandabstand
rechteroffset:	rechteroffset	rightoffset	rechterabstand
rechterrand:	rechterrand	rightedge	rechtekante
rechterrandafstand:	rechterrandafstand	rightedgedistance	rechterkantenabstand
rechtertekst:	rechtertekst	righttext	rechtertext
rechts:	rechts	right	rechts
refereren:	refereren	referencing	referieren
regels:	regels	lines	zeilen
resolutie:	resolutie	resolution	aufloesung
rotatie:	rotatie	rotation	rotation
rugoffset:	rugoffset	backoffset	rumpfabstand
rugwit:	rugwit	backspace	rumpfspatium
schaal:	schaal	scale	format
scheider:	scheider	separator	seperator
scope:	scope	scope	bereich
sectie:	sectie	section	abschnitt
sectienummer:	sectienummer	sectionnumber	abschnittsnummer
soort:	soort	family	familie
spatie:	spatie	space	spatium
spatiering:	spatiering	spacing	spatiumausgleich
stap:	stap	step	schritt
start:	start	start	start
status:	status	state	status
stop:	stop	stop	stop
straal:	straal	radius	radius
sub:	sub	sub	sub
symbool:	symbool	symbol	symbol
symkleur:	symkleur	symcolor	symfarbe
symletter:	symletter	symstyle	symschrift
synoniem:	synoniem	synonym	synonym
synoniemkleur:	synoniemkleur	synonymcolor	synonymfarbe
synoniemletter:	synoniemletter	synonymstyle	synonymschrift
tab:	tab	tab	tab
teken:	teken	sign	zeichen
tekst:	tekst	text	text
tekstcommando:	tekstcommando	textcommand	textbefehl
tekstformaat:	tekstformaat	textsize	textgroesse
tekstkleur:	tekstkleur	textcolor	textfarbe

mult-ini
mult-sys
mult-con
mult-com

tekstletter:	tekstletter	textstyle	textschrift
tekstscheider:	tekstscheider	textseparator	textseparator
tekststatus:	tekststatus	textstatus	textstatus
subtitel:	subtitel	subtitle	untertitel
titel:	titel	title	titel
titeluitlijnen:	titeluitlijnen	aligntitle	titelausrichten
tolerantie:	tolerantie	tolerance	toleranz
tussen:	tussen	inbetween	zwischen
type:	type	type	tippen
uitlijnen:	uitlijnen	align	ausrichtung
van:	van	from	von
variant:	variant	alternative	alternative
verhoog:	verhoog	increment	erhoehen
verhoognummer:	verhoognummer	incrementnumber	nummererhoehen
verwijzing:	verwijzing	reference	referenz
voet:	voet	footer	fusszeile
voetafstand:	voetafstand	footerdistance	fusszeilenabstand
voetstatus:	voetstatus	footerstatus	fusszeilenstatus
voffset:	voffset	voffset	voffset
volgende:	volgende	next	folgende
voor:	voor	before	vor
voorcommando:	voorcommando	commandbefore	vorigerbefehl
vorige:	vorige	previous	vorige
vorigenummer:	vorigenummer	previousnumber	vorigenummer
wijze:	wijze	way	art
wit:	wit	white	weiss
xmax:	xmax	xmax	xmax
xoffset:	xoffset	xoffset	xoffset
xstap:	xstap	xstep	xschritt
ymax:	ymax	ymax	ymax
yoffset:	yoffset	yoffset	yoffset
ystap:	ystap	ystep	yschritt
zij:	zij	side	objektabstand
zelfdepagina:	zelfdepagina	samepage	--seite

contents register context syst mult supp lang font colo spec core cont m s exit go back

mult-ini
mult-sys
mult-con
mult-com

33 \stopconstants

The keywords of values are very language specific and therefore variables for the systems. This list shows soem overlap with the previous one.

34 \startvariables

35

\startvariables	dutch	english	german
aan:	aan	on	an
aanelkaar:	aanelkaar	serried	kleinerabstand
aansluitend:	aansluitend	joinedup	keinabstand
absoluut:	absoluut	absolute	absolut
afsluiter:	afsluiter	stopper	trenner
achtergrond:	achtergrond	background	hintergrund
alles:	alles	all	alles
altijd:	altijd	always	immer
beide:	beide	both	beide
binnen:	binnen	inner	innen
blanko:	blanko	blank	blanko
blokkeer:	blokkeer	disable	sperren
boven:	boven	top	oben
breed:	breed	wide	breit
breedte:	breedte	width	breite
buiten:	buiten	outer	aussen
Cijfers:	Cijfers	Numbers	Ziffern
cijfers:	cijfers	numbers	ziffern
commandos:	commandos	commands	befehle
commando:	commando	command	befehl
concept:	concept	concept	konzept
dag:	dag	day	tag
datum:	datum	date	datum
diepte:	diepte	depth	tiefe
definitief:	definitief	final	endfassung
dubbelzijdig:	dubbelzijdig	doublesided	doppelseitig
eerste:	eerste	first	erste
elk:	elk	each	jede
enkelzijdig:	enkelzijdig	singlesided	einzelseitig
even:	even	even	gerade
flexibel:	flexibel	flexible	flexibel

contents register context syst mult supp lang font colo spec core cont m s exit go back

forceer:	forceer	force	zwinge
gebruikt:	gebruikt	used	verwende
geen:	geen	none	kein
geenwit:	geenwit	nowhite	keinweiss
globaal:	globaal	global	global
groot:	groot	big	gross
grotevoorkeur:	grotevoorkeur	bigpreference	grosszuegig
handhaaf:	handhaaf	keep	behalte
herstel:	herstel	fix	stellewiederher
hier:	hier	here	hier
hoofd:	hoofd	header	kopfzeile
hoog:	hoog	high	hoch
hoogte:	hoogte	height	hoehe
horizontaal:	horizontaal	horizontal	horizontal
inlinker:	inlinker	inleft	imlinken
inmarge:	inmarge	inmargin	imrand
inrechter:	inrechter	inright	imrechten
intekst:	intekst	intext	imtext
intro:	intro	intro	intro
ja:	ja	yes	ja
jaar:	jaar	year	jahr
kader:	kader	frame	rahmen
kantlijn:	kantlijn	marginedge	marginal
kap:	kap	cap	kap
kapitaal:	kapitaal	capital	grossbuchstabe
kenmerk:	kenmerk	referral	merkmal
klein:	klein	small	klein
kleinnormaal:	kleinnormaal	smallnormal	kleinnormal
kleinschuin:	kleinschuin	smallslanted	kleingeneigt
kleinschuinvet:	kleinschuinvet	smallslantedbold	kleingeneigtfett
kleintype:	kleintype	smalltype	kleintippen
kleinvet:	kleinvet	smallbold	kleinfett
kleinvetschuin:	kleinvetschuin	smallboldslanted	kleinfettgeneigt
kleur:	kleur	color	farbe
kolommen:	kolommen	columns	spalten
kop:	kop	head	kopf
label:	label	label	label
laag:	laag	low	tief

mult-ini
mult-sys
mult-con
mult-com

laatste:	laatste	last	letzte
lang:	lang	tall	lang
leeg:	leeg	empty	leer
Letter:	Letter	Character	Buchstabe
letter:	letter	character	buchstabe
Letters:	Letters	Characters	Buchstaben
letters:	letters	characters	buchstaben
lijn:	lijn	line	linie
linker:	linker	left	linker
linkermarge:	linkermarge	leftmargin	linkerrand
linkerrand:	linkerrand	leftedge	linkekante
links:	links	left	links
lokaal:	lokaal	local	lokal
maand:	maand	month	monat
MAAND:	MAAND	MONTH	MONAT
mar:	mar	mar	mar
marge:	marge	margin	marginalie
max:	max	max	max
mediaeval:	mediaeval	mediaeval	mittelalterlich
middel:	middel	medium	mittel
midden:	midden	middle	mittig
naam:	naam	name	name
naast:	naast	opposite	gegenueber
nee:	nee	no	nein
niet:	niet	not	nicht
nooit:	nooit	never	nie
normaal:	normaal	normal	normal
nummer:	nummer	number	nummer
onbekend:	onbekend	unknown	unbekannt
onder:	onder	bottom	unten
oneven:	oneven	odd	ungerade
opelkaar:	opelkaar	packed	kleinerdurchschuss
opmaak:	opmaak	markup	umbruch
opmarge:	opmarge	atmargin	amrand
overlay:	overlay	overlay	overlay
pagina:	pagina	page	seite
paginanummer:	paginanummer	pagenumber	seitennummer
passend:	passend	fit	passend

per:	per	by	pro
postscript:	postscript	postscript	postscript
punt:	punt	dot	punkt
rand:	rand	edge	kante
raster:	raster	screen	raster
recht:	recht	right	rechts
rechter:	rechter	right	rechter
rechtermarge:	rechtermarge	rightmargin	rechterrand
rechterrand:	rechterrand	rechterrand	rechterkante
rechts:	rechts	right	rechts
regel:	regel	line	zeile
relatief:	relatief	relative	relativ
reset:	reset	reset	zuruecksetzten
Romeins:	Romeins	Romannumerals	Roemischezahlen
romeins:	romeins	romannumerals	roemischezahlen
rond:	rond	round	rund
ruim:	ruim	broad	breit
schuin:	schuin	slanted	geneigt
schuinvet:	schuinvet	slantedbold	geneigtfett
sectienummer:	sectienummer	sectionnumber	abschnittsnummer
smal:	smal	tall	schmall
soepel:	soepel	tolerant	tolerant
standaard:	standaard	standard	standard
start:	start	start	start
stop:	stop	stop	stop
streng:	streng	rigged	streng
sub:	sub	sub	sub
sym:	sym	sym	sym
its:	its	its	its
symbool:	symbool	symbol	symbol
tekst:	tekst	text	text
terug:	terug	backward	rueckwaerts
test:	test	test	test
type:	type	type	tippen
uit:	uit	off	aus
vast:	vast	fixed	fest
verder:	verder	continue	fortsetzten
vertikaal:	vertikaal	vertical	vertikal

mult-ini
mult-sys
mult-con
mult-com

vet:	vet	bold	fett
vetschuin:	vetschuin	boldslanted	fettgeneigt
viertal:	viertal	quadruple	viertel
voet:	voet	footer	fusszeile
volgende:	volgende	next	folgende
voorkeur:	voorkeur	preference	einstellung
voorlopig:	voorlopig	temporary	temporaer
vorige:	vorige	previous	vorig
waarde:	waarde	value	wert
wit:	wit	white	weiss
zeer:	zeer	very	sehr
zeersoepel:	zeersoepel	verytolerant	sehrtolerant
zeerstreng:	zeerstreng	veryrigged	sehrstreng

36 `\stopvariables`

The next setup shows the use of the keyword **all**. These constants are the same for all languages.

37 `\startconstants        all`

38

dx:	dx
dy:	dy
nx:	nx
ny:	ny
n:	n
vfil:	vfil
hfil:	hfil
strut:	strut
reset:	reset
set:	set

39 escape: escape

40 apa: apa

41 `\stopconstants`

We need some font family switching names both as constant and as variable.

contents register context syst mult supp lang font colo spec core cont m s exit go back

mult-ini
mult-sys
mult-con
mult-com

```
\startconstants       dutch               english         german

        calligrafie:  calligrafie         calligraphy     kalligraphie
        handschrift:  handschrift         handwritten     handschrift
        schreefloos:  schreefloos         sansserif       grotesk
             romaan:  romaan              roman           antiqua
           teletype:  teletype            teletype        fernschreiber
               type:  type                type            tippen

\stopconstants

\startvariables       dutch               english         german

        calligrafie:  calligrafie         calligraphy     kalligraphie
        handschrift:  handschrift         handwritten     handschrift
        schreefloos:  schreefloos         sansserif       grotesk
             romaan:  romaan              roman           antiqua
           teletype:  teletype            teletype        fernschreiber
               type:  type                type            tippen

\stopvariables
```

All relevent commands of CONTEXT are specified in a structured way that enables the generation of reference cards. This specification is setup in a language independant way. The next category of variables is only used in this context.

```
\startsetupvariables  dutch               english         german

      doornummering:  doornummering       enumeration     nummerierung
      doordefinitie:  doordefinitie       description     beschreibung
         doorsprong:  doorsprong          indentation     einzug
          doorlabel:  doorlabel           labeling        beschriften
 samengesteldelijst:  samengesteldelijst  combinedlist    kombiniereliste
             sectie:  sectie              section         abschnitt
           register:  register            register        register
           synoniem:  synoniem            synonym         synonym
         synoniemen:  synoniemen          synonyms        synonyme
            sorteer:  sorteer             sort            sortiere
           sorteren:  sorteren            sorts           sortieren
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

mult-ini
mult-sys
mult-con
mult-com

```
        naam:  naam                 name                 name
        blok:  blok                 block                block
     blokken:  blokken              blocks               bloecke
      alinea:  alinea               paragraphs           absaetze
\stopsetupvariables
```

The number of elements used for composing user defined commands is rather small. We use a - for empty elements.

```
\startelements        dutch                english              german

     beginvan:  beginvan             beginof              beginvon
      eindvan:  eindvan              endof                endevon
   gekoppelde:  gekoppelde           coupled              verknuepft
           in:  in                   -                    in        % why not in English?
         leeg:  leeg                 empty                leer
         lege:  lege                 empty                leer
         laad:  laad                 load                 laden
        lijst:  lijst                list                 auflisten
     lijstmet:  lijstmet             listof               auflistenvon
       opmaak:  opmaak               makeup               umbruch
       plaats:  plaats               place                setzten
    reserveer:  reserveer            reserve              reservieren
        start:  start                start                start
         stel:  stel                 setup                einstellen
         stop:  stop                 stop                 stop
        tekst:  tekst                text                 text
      verhoog:  verhoog              increment            erhoehen
     volgende:  volgende             next                 folgende
    volledige:  volledige            complete             vollende
       vorige:  vorige               previous             vorige
          zie:  zie                  see                  sieh
\stopelements
```

Last we define some constants and variables that are used in the PRAGMA extensions of CONTEXT.

contents register context syst mult supp lang font colo spec core cont m s exit go back

mult-ini
mult-sys
mult-con
mult-com

54
```
\startconstants         all
                bet:    bet
                dat:    dat
                ken:    ken
                ref:    ref
\stopconstants
```

55
```
\startvariables         all
            formeel:    formeel
          informeel:    informeel
           rekening:    rekening
            sticker:    sticker
              sheet:    sheet
              brief:    brief
                 ls:    ls
              avery:    avery
\stopvariables
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

mult-ini
mult-sys
mult-con
mult-com

3.4 Commands

In this module we define the commands. A more than quick glance at this list leans that it's incomplete. This is due to the fact that the system generated a lot of commands by means of the elements specified somewhere else.

Because the original interface to CONTEXT is dutch, the words left of the : are in dutch. In the near future english will be the core language.

```
\writestatus{loading}{Context Multilingual Macros / Commands}
```

The commands are grouped according to their functionality. Although the `\stop` counterpart of the `\start`–commands could be generated automatically, we've chosen do define it explicitly.

One complication of the english commands, is that we don't want them to overrule or conflict with Plain TEX. The names therefore are not always optimal.

```
\startcommands                    dutch                        english
                                  german

                      language:   taal                         language
                                  sprache
                  mainlanguage:   hoofdtaal                    mainlanguage
                                  hauptsprache
                     translate:   vertaal                      translate
                                  uebersetzten
               installlanguage:   installeertaal               installlanguage
                                  installieresprache

                    showmakeup:   toonopmaak                   showmakeup
                                  zeigeumbruch

                   usespecials:   gebruikspecials              usespecials
                                  benutzespezielles
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

6	setuptype:	steltypein stelletipein	setuptype
	type:	type tippen	type
	typ:	typ tip	typ
	arg:	arg arg	arg
	tex:	tex tex	tex
	definetyping:	definieertypen definieretippen	definetyping
	setuptyping:	steltypenin stelletippenein	setuptyping
	typefile:	typefile tippedatei	typefile
7	defineaccent:	definieeraccent definiereakzent	defineaccent
	definecharacter:	definieerkarakter definierezeichen	definecharacter
	definecommand:	definieercommando definierebefehl	definecommand
	startcoding:	startcodering startkodierung	startcoding
	stopcoding:	stopcodering stopkodierung	stopcoding
	definecorpsenvironment:	definieerkorpsomgeving definierefliesstextumgebung	definecorpsenvironment
	definecorps:	definieerkorps definierefliesstext	definecorps
8	kap:	kap kap	kap
	KAP:	KAP KAP	KAP
	nokap:	nokap nokap	nokap

	Kap:	Kap	Kap
		Kap	
	Kaps:	Kaps	Kaps
		Kaps	
	WORD:	WOORD	WORD
		WORT	
	WORDS:	WOORDEN	WORD
		WORT	
	Word:	Woord	Word
		Wort	
	Words:	Woorden	Words
		Woerter	
	stretched:	opgerekt	stretched
		–	
	overstrike:	doorstreep	overstrike
		ueberstrichen	
	overstrikes:	doorstrepen	overstrikes
		ueberstreichen	
	underbar:	onderstreep	underbar
		unterstrichen	
	underbars:	onderstrepen	underbars
		unterstreichen	
9	setupcorps:	stelkorpsin	setupcorps
		stellefliesstextein	
	switchtocorps:	switchnaarkorps	switchtocorps
		wechselezumfliesstext	
	showcorps:	toonkorps	showcorps
		zeigefliesstext	
	showcorpsenvironment:	toonkorpsomgeving	showcorpsenvironment
		zeigefliesstextumgebung	
10	gebruikmodule:	gebruikmodule	usemodule
		verwendemodul	
	gebruikmodules:	gebruikmodules	usemodules
		verwendemodule	
11	starttekst:	starttekst	starttext
		starttext	

	stoptekst:	stoptekst stoptext	stoptext
12	margetitel:	margetitel marginaltitel	margintitle
	margewoord:	margewoord marginalwort	marginword
	inlinker:	inlinker imlinken	inleft
	inmarge:	inmarge inmarginalie	inmargin
	inanderemarge:	inanderemarge inanderermarginale	inothermargin
	inrechter:	inrechter imrechten	inright
	startmargeblok:	startmargeblok startmarginalblock	startmarginblock
	stopmargeblok:	stopmargeblok stopmarginalblock	stopmarginblock
	stelinmargein:	stelinmargein stelleinmarginalieein	setupinmargin
	stelmargeblokkenin:	stelmargeblokkenin stellemarginalblockein	setupmarginblocks
13	inleftside:	inlinkerrand imlinkenrand	inleftside
	inleftmargin:	inlinkermarge inlinkermarginale	inleftmargin
	inrightmargin:	inrechtermarge inrechtermarginale	inrightmargin
	inrightside:	inrechterrand imrechtenrand	inrightside
14	woordrechts:	woordrechts wortrechts	wordright
15	blokje:	blokje rechteck	blackrule
	blokjes:	blokjes	blackrules

mult-ini
mult-sys
mult-con
mult-com

		rechtecke	
	stelblokjesin:	stelblokjesin	setupblackrules
		stellerechteckein	
16	blanko:	blanko	blank
		blanko	
	stelblankoin:	stelblankoin	setupblank
		stelleblankoein	
	corrigeerwitruimte:	corrigeerwitruimte	correctwhitespace
		korrigierezwischenraum	
	vastespaties:	vastespaties	fixedspaces
		festesspatium	
	geenspatie:	geenspatie	nospace
		keinspatium	
	spatie:	spatie	
		spatium	
	geenwitruimte:	geenwitruimte	nowhitespace
		keinzwischenraum	
	opelkaar:	opelkaar	packed
		kleinerdurchschuss	
	startopelkaar:	startopelkaar	startpacked
		startkleinerdurchschuss	
	stopopelkaar:	stopopelkaar	stoppacked
		stopkleinerdurchschuss	
	startvanelkaar:	startvanelkaar	startunpacked
		startgrosserdurchschuss	
	stopvanelkaar:	stopvanelkaar	stopunpacked
		stopkleinerdurchschuss	
	startregelcorrectie:	startregelcorrectie	startlinecorrection
		startzeilenkorrektur	
	stopregelcorrectie:	stopregelcorrectie	stoplinecorrection
		stopzeilenkorrektur	
	omlaag:	omlaag	godown
		nachunten	
	witruimte:	witruimte	whitespace
		zwischenraum	

contents register context syst mult supp lang font colo spec core cont m s exit go back

17	nietinspringen:	nietinspringen	noindenting
		nichteinziehen	
	inspringen:	inspringen	indenting
		einziehen	
	stelinspringenin:	stelinspringenin	setupindenting
		stelleeinziehenein	
18	startuitlijnen:	startuitlijnen	startalignment
		startausrichtung	
	stopuitlijnen:	stopuitlijnen	stopalignment
		stopausrichtung	
19	startregels:	startregels	startlines
		startzeilen	
	stopregels:	stopregels	stoplines
		stopzeilen	
	stelregelnummerenin:	stelregelnummerenin	setuplinenumbering
		stellezeilennummerierungein	
	startregelnummeren:	startregelnummeren	startlinenumbering
		startzeilennummerierung	
	stopregelnummeren:	stopregelnummeren	stoplinenumbering
		stopzeilennummerierung	
	startregel:	startregel	startline
		startzeile	
	stopregel:	stopregel	stopline
		stopzeile	
	eenregel:	eenregel	someline
		einezeile	
	inregel:	inregel	inline
		inzeile	
	crlf:	crlf	crlf
		crlf	
	stelregelsin:	stelregelsin	setuplines
		stellezeilenein	
20	startsmaller:	startsmaller	startnarrower
		startenger	
	stopsmaller:	stopsmaller	stopnarrower

		stopenger	
	stelsmallerin:	stelsmallerin	setupnarrower
		stelleengerein	
21	starttabel:	starttabel	starttable
		starttabelle	
	stoptabel:	stoptabel	stoptable
		stoptabelle	
	steltabellenin:	steltabellenin	setuptables
		stelletabelleein	
22	pagina:	pagina	page
		seite	
	koppelpagina:	koppelpagina	couplepage
		doppelseite	
	soortpagina:	soortpagina	pagetype
		seitentyp	
	verwerkpagina:	verwerkpagina	processpage
		bearbeiteseite	
	koppelpapier:	koppelpapier	couplepaper
		doppelseitigespapier	
	selecteerpapier:	selecteerpapier	selectpaper
		waehlepapieraus	
	scherm:	scherm	screen
		bildschirm	
23	definieersectie:	definieersectie	definesection
		definiereabschnitt	
	definieersectieblok:	definieersectieblok	definesectionblock
		definiereabschnittsblock	
	stelsectieblokin:	stelsectieblokin	setupsectionblock
		stelleabschnittsblockein	
	stelsectiein:	stelsectiein	setupsection
		stelleabschnittein	
24	geenbovenenonderregels:	geenbovenenonderregels	notopandbottomlines
		keinzeilenobenundunten	
	geenhoofdenvoetregels:	geenhoofdenvoetregels	noheaderandfooterlines
		keinekopfundfusszeilen	

```
              stelbovenin:  stelbovenin                  setuptop
                            stellenobenein
        stelboventekstenin: stelboventekstenin           setuptoptexts
                            stelletextobenein
              stelhoofdin:  stelhoofdin                  setupheader
                            stellekopfzeileein
        stelhoofdtekstenin: stelhoofdtekstenin           setupheadertexts
                            stellekopfzeilentextein
          stelnummeringin:  stelnummeringin              setuppagenumbering
                            stelleseitennummeriernungein
              stelonderin:  stelonderin                  setupbottom
                            stelleuntenein
        stelondertekstenin: stelondertekstenin           setupbottomtexts
                            stelletextuntenein
               stelvoetin:  stelvoetin                   setupfooter
                            stellefusszeileein
         stelvoettekstenin: stelvoettekstenin            setupfootertexts
                            stellefusszeilentextein
        stelpaginanummerin: stelpaginanummerin           setuppagenumber
                            stelleseitennummerein
     stelsubpaginanummerin: stelsubpaginanummerin        setupsubpagenumber
                            stelleunterseitennummerein
              steltekstin:  steltekstin                  setuptext
                            stelletextein
        steltekstteskstenin: steltekstteskstenin          setuptexttexts
                            stelletexttexteein

25                  items:  items                        items
                            posten
              stelitemsin:  stelitemsin                  setupitems
                            stellepostenein

26             regellinks:  regellinks                   leftlined
                            zeilelinks  % better: linksbuendig
              regelmidden:  regelmidden                  middlelined
                            zeilemitte  % better: zentriert
              regelrechts:  regelrechts                  rightlined
                            zeilerechts % better: rechtsbuendig
```

27	startkolommen:	startkolommen	startcolumns
		startspalten	
	stopkolommen:	stopkolommen	stopcolumns
		stopspalten	
	stelkolommenin:	stelkolommenin	setupcolumns
		stellespaltenein	
	kolom:	kolom	column
		spalte	
28	kop:	kop	head
		kopf	
	but:	but	but
		but	
	som:	som	item
		pos	
	nop:	nop	nop
		nop	
	mar:	mar	mar
		mar	
	sym:	sym	sym
		sym	
	its:	its	its
		its	
	startopsomming:	startopsomming	startitemize
		startaufzaehlung	
	stelopsommingin:	stelopsommingin	setupitemize
		stelleaufzaehlungein	
	stopopsomming:	stopopsomming	stopitemize
		stopaufzaehlung	
29	definieerkop:	definieerkop	definehead
		definierekopf	
	stelkopin:	stelkopin	setuphead
		stellekopfein	
	stelkoppenin:	stelkoppenin	setupheads
		stellekoepfeein	
	stelkopnummerin:	stelkopnummerin	setupheadnumber
		stellekopfzahlein	

mult-ini
mult-sys
mult-con
mult-com

contents register context syst mult supp lang font colo spec core cont m s exit go back

mult-ini
mult-sys
mult-con
mult-com

	kopnummer:	kopnummer kopfnummer	headnumber
30	resetmarkering:	resetmarkering ruecksetztenbeschriftung	resetmarking
	stelmarkeringin:	stelmarkeringin stellebeschriftungein	setupmarking
	definieermarkering:	definieermarkering definierebeschriftung	definemarking
	geenmarkering:	geenmarkering keinebeschriftung	nomarking
	markeer:	markeer beschriftung	marking
	ontkoppelmarkering:	ontkoppelmarkering entknuepfebeschriftung	decouplemarking
	koppelmarkering:	koppelmarkering verknuepfebeschriftung	couplemarking
	haalmarkering:	haalmarkering holebeschriftung	getmarking
31	stellayoutin:	stellayoutin stellelayoutein	setuplayout
	paslayoutaan:	paslayoutaan passelayoutan	adaptlayout
	steloffsetin:	steloffsetin stelleoffsetein	setupoffset
	tooninstellingen:	tooninstellingen zeigeeinstellungen	showsetups
	toonkader:	toonkader zeigerahmen	showframe
	toonopbouw:	toonopbouw zeigeaufbau	showbuildup
	toonlayout:	toonlayout zeigelayout	showlayout
	toonstruts:	toonstruts zeigestruts	showstruts
	definieerpapierformaat:	definieerpapierformaat definierepapierformat	definepapersize

contents register context syst mult supp lang font colo spec core cont m s exit go back

mult-ini
mult-sys
mult-con
mult-com

	stelpapierformaatin:	stelpapierformaatin stellepapierformatin	setuppapersize
32	versie:	versie version	version
33	achtergrond:	achtergrond hintergrund	background
	startachtergrond:	startachtergrond starthintergrund	startbackground
	stelachtergrondenin:	stelachtergrondenin stellehintergruendeein	setupbackgrounds
	stelachtergrondin:	stelachtergrondin stellehintergrundein	setupbackground
	stopachtergrond:	stopachtergrond stophintergrund	stopbackground
34	steluitlijnenin:	steluitlijnenin stelleausrichtungein	setupalign
	stelwitruimtein:	stelwitruimtein stellezwischenraumein	setupwhitespace
35	stelinteractiein:	stelinteractiein stelleinteraktionein	setupinteraction
	stelinteractiemenuin:	stelinteractiemenuin stelleinteraktionsmenueein	setupinteractionmenu
	definieerinteractiemenu:	definieerinteractiemenu definiereinteraktionsmenue	defineinteractionmenu
	startinteractiemenu:	startinteractiemenu startinteraktionsmenue	startinteractionmenu
	blokkeerinteractiemenu:	blokkeerinteractiemenu inaktiviereinteraktionsmenue	disableinteractionmenu
	interactiebuttons:	interactiebuttons interaktionsknopfe	interactionbuttons
	interactiebalk:	interactiebalk interaktionsbalken	interactionbar
	stelinteractiebalkin:	stelinteractiebalkin stelleinteraktionsbalkenein	setupinteractionbar
	stelinteractieschermin:	stelinteractieschermin	setupinteractionscreen

mult-ini
mult-sys
mult-con
mult-com

		stelleinteraktionsbildschirmein	
36	definieerbeeldmerk:	definieerbeeldmerk definierelogo	definelogo
	plaatsbeeldmerken:	plaatsbeeldmerken platzierelogo	placelogos
37	definecolor:	definieerkleur definierefarbe	definecolor
	definecolorgroup:	definieerkleurgroep definierefarbengruppe	definecolorgroup
	definepalet:	definieerpalet definierepalette	definepalet
	gray:	grijs grau	gray
	color:	kleur farbe	color
	startcolor:	startkleur startfarbe	startcolor
	stopcolor:	stopkleur stopfarbe	stopcolor
	setupcolors:	stelkleurenin stellefarbenein	setupcolors
	setupcolor:	stelkleurin stellefarbeein	setupcolor
	setuppalet:	stelpaletin stellepaletteein	setuppalet
	showcolor:	toonkleur zeigefarbe	showcolor
	showcolorgroup:	toonkleurgroep zeigefarbengruppe	showcolorgroup
	showpalet:	toonpalet zeigepalette	showpalet
	comparecolorgroup:	vergelijkkleurgroep vergleichefarbengruppe	comparecolorgroup
	comparepalet:	vergelijkpalet vergleichepalette	comparepalet
	colorvalue:	kleurwaarde	colorvalue

contents register context syst mult supp lang font colo spec core cont m s exit go back

mult-ini
mult-sys
mult-con
mult-com

	grayvalue:	farbewert grijswaarde grauwert	grayvalue
38	startraster:	startraster startraster	startraster
	stopraster:	stopraster stopraster	stopraster
39	definieerblok:	definieerblok definiereblock	defineblock
	gebruikblokken:	gebruikblokken verwendeblock	useblocks
	geenblokkenmeer:	geenblokkenmeer keinebloeckemehr	nomoreblocks
	handhaafblokken:	handhaafblokken behaltebloecke	keepblocks
	selecteerblokken:	selecteerblokken waehlebloeckeaus	selectblocks
	stelblokin:	stelblokin stelleblockein	setupblock
	verbergblokken:	verbergblokken verbergebloecke	hideblocks
40	definieerlijst:	definieerlijst definiereliste	definelist
	definieersamengesteldelijst:	definieersamengesteldelijst definierezusammengestellteliste	definecombinedlist
	plaatslijst:	plaatslijst plaziereliste	placelist
	schrijfnaarlijst:	schrijfnaarlijst schreibezurliste	writetolist
	schrijftussenlijst:	schrijftussenlijst schreibezwischenliste	writebetweenlist
	stellijstin:	stellijstin stellelisteein	setuplist
	stelsamengesteldelijstin:	stelsamengesteldelijstin stellezusammengestelltelisteein	setupcombinedlist

contents register context syst mult supp lang font colo spec core cont m s exit go back

41	definieerreferentielijst:	definieerreferentielijst definierereferenzliste	definereferencelist
	plaatsreferentielijst:	plaatsreferentielijst plazierereferenzliste	placereferencelist
	schrijfnaarreferentielijst:	schrijfnaarreferentielijst schreibezurreferenzliste	writetoreferencelist
	stelreferentielijstin:	stelreferentielijstin stellereferenzlisteein	setupreferencelist
42	definieerplaatsblok:	definieerplaatsblok definieregleitobjekt	definefloat
	stelplaatsblokin:	stelplaatsblokin stellegleitobjektein	setupfloat
	stelplaatsblokkenin:	stelplaatsblokkenin stellegleitobjekteein	setupfloats
	startcombinatie:	startcombinatie startkombination	startcombination
	stopcombinatie:	stopcombinatie stopkombination	stopcombination
	stelblokkopjein:	stelblokkopjein stellebildunterschriftein	setupcaption
	stelblokkopjesin:	stelblokkopjesin stellebilderunterschriftein	setupcaptions
	stelcombinatiesin:	stelcombinatiesin stellekombinationein	setupcombinations
43	startoverlay:	startoverlay startoverlay	startoverlay
	stopoverlay:	stopoverlay stopoverlay	stopoverlay
	defineoverlay:	definieeroverlay definiereoverlay	defineoverlay
44	definieerregister:	definieerregister definiereregister	defineregister
	koppelregister:	koppelregister verknuepfregister	coupleregister
	stelregisterin:	stelregisterin	setupregister

mult-ini
mult-sys
mult-con
mult-com

contents register context syst mult supp lang font colo spec core cont m s exit go back

mult-ini
mult-sys
mult-con
mult-com

		stelleregisterein	
	schrijfnaarregister:	schrijfnaarregister	writetoregister
		schreibezumregister	
	plaatsregister:	plaatsregister	placeregister
		plaziereregister	
45	definieersorteren:	definieersorteren	definesorting
		definieresortieren	
	definieersynoniemen:	definieersynoniemen	definesynonyms
		definieresynonyme	
	stelsorterenin:	stelsorterenin	setupsorting
		stellesortierenein	
	stelsynoniemin:	stelsynoniemin	setupsynonym
		stellesynonymein	
46	startsynchronisatie:	startsynchronisatie	startsynchronization
		startsynchronisation	
	stopsynchronisatie:	stopsynchronisatie	stopsynchronization
		stopsynchronisation	
	stelsynchronisatiebalkin:	stelsynchronisatiebalkin	setupsynchronizationbar
		stellesynchronisationsbalkenein	
	stelsynchronisatiein:	stelsynchronisatiein	setupsynchronization
		stellesynchronisationein	
	synchronisatiebalk:	synchronisatiebalk	synchronizationbar
		synchronisationsbalken	
	synchroniseer:	synchroniseer	synchronize
		synchronisieren	
47	gebruikexterndocument:	gebruikexterndocument	useexternaldocument
		verwendeexteresdokument	
48	stelprogrammasin:	stelprogrammasin	setupprograms
		stelleprogrammein	
	definieerprogramma:	definieerprogramma	defineprogram
		definiereprogrammein	
	programma:	programma	program
		programm	

contents register context syst mult supp lang font colo spec core cont m s exit go back

mult-ini
mult-sys
mult-con
mult-com

49	definieerprofiel:	definieerprofiel definiereprofil	defineprofile
	definieerversie:	definieerversie definiereversion	defineversion
	markeerversie:	markeerversie beschrifteversion	markversion
	selecteerversie:	selecteerversie waehleversionaus	selectversion
	startprofiel:	startprofiel startprofil	startprofile
	startversie:	startversie startversion	startversion
	stelprofielenin:	stelprofielenin stelleprofilein	setupprofiles
	stelversiesin:	stelversiesin stelleversionein	setupversions
	stopprofiel:	stopprofiel stopprofil	stopprofile
	stopversie:	stopversie stopversion	stopversion
	volgprofiel:	volgprofiel folgeprofil	followprofile
	volgprofielversie:	volgprofielversie folgeprofilversion	followprofileversion
	volgversie:	volgversie folgeversion	followversion
50	doordefinieren:	doordefinieren definierebeschreibung	definedescription
	doorlabelen:	doorlabelen definierelabel	definelabel
	doornummeren:	doornummeren definierenummerierung	defineenumeration
	doorspringen:	doorspringen definiereeinzug	defineindenting
	steldoordefinierenin:	steldoordefinierenin definierebeschreibungen	setupdescriptions
	steldoornummerenin:	steldoornummerenin	setupenumerations

contents register context syst mult supp lang font colo spec core cont m s exit go back

mult-ini
mult-sys
mult-con
mult-com

		stellebeschreibungein	
	steldoorspringenin:	steldoorspringenin	setupindentations
		stelleeinzuegein	
	steldunnelijnenin:	steldunnelijnenin	setupthinrules
		stelleduennerumrissein	
51	steltabin:	steltabin	setuptab
		stelletabein	
	tab:	tab	tab
		tab	
52	stelexternefigurenin:	stelexternefigurenin	setupexternalfigures
		stelleexterneabbildungenein	
	toonexternefiguren:	toonexternefiguren	showexternalfigures
		zeigeexterneabbildungen	
	externalfigure:	externfiguur	externalfigure
		externeabbildung	
	toonexternfiguur:	toonexternfiguur	showexternalfigure
		zeigeexterneabbildung	
	gebruikexternfiguur:	gebruikexternfiguur	useexternalfigure
		verwendeexterneabbildung	
53	startfiguur:	startfiguur	startfigure
		startabbildung	
	stopfiguur:	stopfiguur	stopfigure
		stopabbildung	
	refereer:	refereer	referring
		referieren	
	markeer:	markeer	marking
		beschriftung	
54	dunnelijn:	dunnelijn	thinrule
		duennelinie	
	dunnelijnen:	dunnelijnen	thinrules
		duennerumriss	
	haarlijn:	haarlijn	hairline
		haarlinie	
	invullijnen:	invullijnen	fillinrules
		gefuelltesrechteck	

contents register context syst mult supp lang font colo spec core cont m s exit go back

mult-ini
mult-sys
mult-con
mult-com

invulregel:	invulregel	fillinline
	gefuelltezeile	
kantlijn:	kantlijn	marginrule
	marginallinie	
startkantlijn:	startkantlijn	startmarginrule
	startmarginallinie	
stelinvullijnenin:	stelinvullijnenin	setupfillinrules
	stellegefuelltesrechteckein	
stelinvulregelsin:	stelinvulregelsin	setupfillinlines
	stellegefuelltezeileein	
stelkantlijnin:	stelkantlijnin	setupmarginrules
	stellemarginallinieein	
stellijnenin:	stellijnenin	setuprules
	stelleumrissein	
steltekstlijnenin:	steltekstlijnenin	setuptextrules
	stelletextumrissein	
stopkantlijn:	stopkantlijn	stopmarginrule
	stopmarginallinie	
tekstlijn:	tekstlijn	textline
	textlinie	
vl:	vl	vl
	vl	
hl:	hl	hl
	hl	

55

omlijnd:	omlijnd	framed
	umrahmt	
inlijnd:	inlijnd	inframed
	imumriss	
stelomlijndin:	stelomlijndin	setupframed
	stelleumrahmtein	
startkadertekst:	startkadertekst	startframedtext
	startumrahmtertext	
stopkadertekst:	stopkadertekst	stopframedtext
	stopumrahmtertext	
stelkadertekstenin:	stelkadertekstenin	setupframedtexts
	stelleumrahmtetexteein	
stelrastersin:	stelrastersin	setupscreens

contents register context syst mult supp lang font colo spec core cont m s exit go back

mult-ini
mult-sys
mult-con
mult-com

		stellerasterein	
56	rooster:	rooster gitter	grid
57	button:	button knopf	button
	menubutton:	menubutton menueknopf	menubutton
	stelbuttonsin:	stelbuttonsin stelleknopfein	setupbuttons
58	gebruikreferenties:	gebruikreferenties verwendereferenzen	usereferences
	reflijst:	reflijst refliste	reflist
	paginareferentie:	paginareferentie seitenreferenz	pagereference
	referentie:	referentie referenz	reference
	stelreferentielijstin:	stelreferentielijstin stellereferenzlisteein	setupreferencelist
	stelrefererenin:	stelrefererenin stellereferenzierenein	setupreferencing
	tekstreferentie:	tekstreferentie textreferenz	textreference
	uit:	uit von	from
	in:	in in	in
	op:	op bei	at
	naar:	naar zu	goto
	naarbox:	naarbox zurbox	gotobox
59	startformule:	startformule startformel	startformula

contents register context syst mult supp lang font colo spec core cont m s exit go back

mult-ini
mult-sys
mult-con
mult-com

stopformule:	stopformule stopformel	stopformula
plaatsformule:	plaatsformule plaziereformel	placeformula
plaatssubformule:	plaatssubformule plaziereunterformel	placesubformula
stelformulesin:	stelformulesin stelleformelnein	setupformulas
startgegeven:	startgegeven startgegeben	startgiven
stopgegeven:	stopgegeven stopgegeben	stopgiven
startlegenda:	startlegenda startlegende	startlegend
stoplegenda:	stoplegenda stoplegende	stoplegend
60 mathematics:	wiskunde mathematik	mathematics
dimension:	dimensie dimension	dimension
nodimension:	geendimensie keindimension	nodimension
61 startomgeving:	startomgeving startumgebung	startenvironment
stopomgeving:	stopomgeving stopumgebung	stopenvironment
startdeelomgeving:	startdeelomgeving startlokaleumgebung	startlocalenvironment
startonderdeel:	startonderdeel startkomponente	startcomponent
stoponderdeel:	stoponderdeel stopkomponente	stopcomponent
startprodukt:	startprodukt startprodukt	startproduct
stopprodukt:	stopprodukt stopprodukt	stopproduct

contents register context syst mult supp lang font colo spec core cont m s exit go back

mult-ini
mult-sys
mult-con
mult-com

	startproject:	startproject startprojekt	startproject
	stopproject:	stopproject stopprojekt	stopproject
62	project:	project projekt	project
	onderdeel:	onderdeel komponente	component
	produkt:	produkt produkt	product
	omgeving:	omgeving umgebung	environment
	geenfilesmeer:	geenfilesmeer keinedateienmehr	nomorefiles
63	haalbuffer:	haalbuffer holebuffer	getbuffer
	startbuffer:	startbuffer startbuffer	startbuffer
	stopbuffer:	stopbuffer stopbuffer	stopbuffer
	stelbufferin:	stelbufferin stellebufferein	setupbuffer
	typebuffer:	typebuffer tippebuffer	typebuffer
64	definieersymbool:	definieersymbool definieresymbol	definesymbol
	symbool:	symbool symbol	symbol
	definieerconversie:	definieerconversie definierekonversion	defineconversion
	Numbers:	Cijfers Ziffern	Numbers
	numbers:	cijfers ziffern	numbers
	romannumerals:	romeins	romannummerals

contents register context syst mult supp lang font colo spec core cont m s exit go back

		roemischezahlen	
	Romannumerals:	Romeins	Romannummerals
		Roemischezahlen	
	character:	letter	character
		buchstabe	
	Character:	Letter	Character
		Buchstabe	
	characters:	letters	characters
		buchstaben	
	Characters:	Letters	Characters
		Buchstaben	
	maand:	maand	–
		monat	
	MAAND:	MAAND	–
		MONAT	
65	betekenis:	betekenis	united
		bedeutung	
	voluit:	voluit	infull
		volleswort	
66	citaat:	citaat	quotation
		zitat	
	citeer:	citeer	quote
		zitieren	
	startcitaat:	startcitaat	startquotation
		startzitat	
	stopcitaat:	stopcitaat	stopquotation
		stopzitat	
	stelciterenin:	stelciterenin	setupquote
		stellezitierenein	
67	definieer:	definieer	define
		definieren	
	herhaal:	herhaal	redo
		wiederhohlen	
	gebruikcommandos:	gebruikcommandos	usecommands
		verwendebefehl	

	definieerstartstop:	definieerstartstop definierestartstop	definestartstop
	startlokaal:	startlokaal startlokal	startlocal
	stoplokaal:	stoplokaal stoplokal	stoplocal
	naam:	naam name	name
68	definieeropmaak:	definieeropmaak definiereumbruch	definemakeup
	stelopmaakin:	stelopmaakin stelleumbruchein	setupmakeup
69	gebruikexternefile:	gebruikexternefile verwendeexternedatei	useexternalfile
	gebruikexternefiles:	gebruikexternefiles verwendeexternedateien	useexternalfiles
70	huidigedatum:	huidigedatum heutigesdatum	currentdate
	kenmerk:	kenmerk verweis	referral
	kenmerkdatum:	kenmerkdatum verweisdatum	referraldate
71	hoog:	hoog hoch	high
	laho:	laho hoti	lohi
	laag:	laag tief	low
72	startuitstellen:	startuitstellen startverschieben	startpostponing
	stopuitstellen:	stopuitstellen stopverschieben	stoppostponing
	startverbergen:	startverbergen startverbergen	starthiding

	stopverbergen:	stopverbergen stopverbergen	stophiding
73	koptekst:	koptekst kopftext	headtext
	labeltekst:	labeltekst labeltext	labeltext
	stelkoptekstin:	stelkoptekstin stellekopftextein	setupheadtext
	stellabeltekstin:	stellabeltekstin stellelabeltextein	setuplabeltext
74	stelvoetnotenin:	stelvoetnotenin stellefussnotenein	setupfootnotes
	noot:	noot notiz	note
	voetnoot:	voetnoot fussnote	footnote
75	breuk:	breuk bruch	fraction
	chem:	chem chem	chem
76	startnaast:	startnaast startgegenueber	startopposite
	stopnaast:	stopnaast stopgegenueber	stopopposite
	stelnaastplaatsenin:	stelnaastplaatsenin stellegegenueberplazierenein	setupoppositeplacing
77	startpositioneren:	startpositioneren startpositionieren	startpositioning
	stoppositioneren:	stoppositioneren stoppositionieren	stoppositioning
	positioneer:	positioneer position	position
	stelpositionerenin:	stelpositionerenin stellepositionierenein	setuppositioning

78	roteer:	roteer drehen	rotate
	stelroterenin:	stelroterenin stelledrehenein	setuprotate
79	stelnummerenin:	stelnummerenin stellenummerierungein	setupnumbering
	reset:	reset ruecksetzten	reset
80	stelpublicatiesin:	stelpublicatiesin stellepublikationein	setuppublications
	publicatie:	publicatie publikation	publication
81	definieerhbox:	definieerhbox definierehbox	definehbox
82	toelichting:	toelichting bemerkung	remark
83	toevoegen:	toevoegen zusatz	adding
84	punten:	punten punkt	periods
85	stelkoppeltekenin:	stelkoppeltekenin stellebindestrichein	setuphyphenmark
	stelinterliniein:	stelinterliniein stellezeilenabstandein	setupinterlinespace
	stelspatieringin:	stelspatieringin stellespatiumein	setupspacing
	steltolerantiein:	steltolerantiein stelletoleranzein	setuptolerance
	stelsysteemin:	stelsysteemin stellesystemein	setupsystem
86	definieeralineas:	definieeralineas definiereabsaetz	defineparagraphs

mult-ini
mult-sys
mult-con
mult-com

stelalineasin:	stelalineasin stelleabsaetzein	setupparagraphs
87 geentest:	geentest keintest	donttest

88 `\stopcommands`

There are a lot of variables that users can use in setups and dedicated macros. (*I still have to check the english names.*)

89 `\startcommands`	dutch german	english
90 bovenhoogte:	bovenhoogte –	topheight
bovenafstand:	bovenafstand –	topdistance
hoofdhoogte:	hoofdhoogte –	headerheight
hoofdafstand:	hoofdafstand –	headerdistance
teksthoogte:	teksthoogte –	textheight
voetafstand:	voetafstand –	footerdistance
voethoogte:	voethoogte –	footerheight
onderafstand:	onderafstand –	bottomdistance
onderhoogte:	onderhoogte –	bottomheight
margebreedte:	margebreedte –	marginwidth
linkermargebreedte:	linkermargebreedte –	leftmarginwidth
rechtermargebreedte:	rechtermargebreedte –	rightmarginwidth

contents register context syst mult supp lang font colo spec core cont m s exit go back

margeafstand:	margeafstand	–	margindistance
randbreedte:	randbreedte	–	edgewidth
linkerrandbreedte:	linkerrandbreedte	–	leftedgewidth
rechterrandbreedte:	rechterrandbreedte	–	rightedgewidth
randafstand:	randafstand	–	edgedistance
tekstbreedte:	tekstbreedte	–	textwidth
zetbreedte:	zetbreedte	–	makeupwidth
zethoogte:	zethoogte	–	makeupheight
kopwit:	kopwit	–	topspace
rugwit:	rugwit	–	backspace

```
\stopcommands
```

At PRAGMA we use an extended version of CONTEXT. The commands below are part of this. Beware of conflicts when defining your own.

\startcommands	dutch	german	english
startdocument:	startdocument	startdokument	startdocument
stopdocument:	stopdocument	stopdokument	stopdocument
startoverzicht:	startoverzicht	startueberblick	startoverview
stopoverzicht:	stopoverzicht	stopueberblick	stopoverview
stelbrievenin:	stelbrievenin		setupcorrespondence

	stellekorrespondenzein	
brieven:	brieven	letters
	briefe	
brief:	brief	letter
	brief	
label:	label	label
	label	
sheet:	sheet	sheet
	blatt	
stelstickersin:	stelstickersin	setupstickers
	stellestickerein	
stelsheetsin:	stelsheetsin	setupsheets
	stelleblattein	
labels:	labels	labels
	labels	
woonplaats:	woonplaats	domicile
	wohnort	

```
94 \stopcommands
```

4 General Support

4.1 Missing (For Generic Use)

Some support modules are more or less independant. This module, which is not part of plain CONTEXT, provides the missing macros and declarations of registers.

\ifnocontextobject

First we take care of redundant defining. The next set of macros are a bit complicated by the fact that Plain TEX defines the `\new`–macros as being outer. Furthermore nested `\if`'s can get us into trouble.

1
```
\def\definecontextobject%
  {\iftrue}
```

2
```
\def\gobblecontextobject%
  {\setbox0=\hbox
     \bgroup
     \long\def\gobblecontextobject##1\fi{\egroup}%
     \expandafter\gobblecontextobject\string}
```

3
```
\def\ifnocontextobject#1\do%
  {\ifx#1\undefined
     \let\next=\definecontextobject
   \else
     %\writestatus{system}{beware of conflicting \string#1}%
     \let\next=\gobblecontextobject
   \fi
   \next}
```

\writestatus

We start each module with a message. Normally the output is formatted, but here we keep things simple.

```
\ifnocontextobject \writestatus \do

  \def\writestatus#1#2%
    {\immediate\write16{#1 : #2}}

\fi
```

Lets see if it works.

```
\writestatus{loading}{Context Support Macros / Missing}
```

\protect
\unprotect

Next we present a poor mans alternative for `\protect` and `\unprotect`, two commands that enable us to use the characters `@`, `!` and `?` in macro names.

```
\ifnocontextobject \protect \do

  \let\protect=\relax

\fi

\ifnocontextobject \unprotect \do

  \def\unprotect%
    {\catcode`@=11
     \catcode`!=11
     \catcode`?=11
     \let\normalprotect=\protect
     \edef\protect%
       {\catcode`@=\the\catcode`@\relax
        \catcode`!=\the\catcode`!\relax
```

```
          \catcode`?=\the\catcode`?\relax
          \let\protect=\normalprotect}}
```

13

```
\fi
```

We start using this one it at once.

14

```
\unprotect
```

\scratch...
\if...
\next...

We need some scratch registers. Users are free to use them, but can never be sure of their value once another macro is called. We only allocate things when they are yet undefined. This way we can't mess up other macro packages, but of course previous definitions can mess up our modules.

15

```
\ifnocontextobject \scratchcounter       \do \newcount  \scratchcounter   \fi
\ifnocontextobject \scratchdimen         \do \newdimen  \scratchdimen     \fi
\ifnocontextobject \scratchskip          \do \newskip   \scratchskip      \fi
\ifnocontextobject \scratchmuskip        \do \newmuskip \scratchmuskip    \fi
\ifnocontextobject \scratchbox           \do \newbox    \scratchbox       \fi
\ifnocontextobject \scratchread          \do \newread   \scratchread      \fi
\ifnocontextobject \scratchwrite         \do \newwrite  \scratchread      \fi
```

16

```
\ifnocontextobject \nextbox              \do \newbox    \nextbox          \fi
```

17

```
\ifnocontextobject \nextdepth            \do \newdimen  \nextdepth        \fi
```

18

```
\ifnocontextobject \ifCONTEXTtrue        \do \newif\ifCONTEXT             \fi
\ifnocontextobject \ifdonetrue           \do \newif\ifdone                \fi
\ifnocontextobject \ifeightbitcharacters \do \newif\ifeightbitcharacters \fi
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

\@@... We use symbolic name for ⟨*catcodes*⟩. They can only be used when we are in unprotected state.

19
```
\ifnocontextobject \@@escape             \do \chardef\@@escape     =  0  \fi
\ifnocontextobject \@@begingroup         \do \chardef\@@begingroup =  1  \fi
\ifnocontextobject \@@endgroup           \do \chardef\@@endgroup   =  2  \fi
\ifnocontextobject \@@ignore             \do \chardef\@@ignore     =  9  \fi
\ifnocontextobject \@@space              \do \chardef\@@space      = 10  \fi
\ifnocontextobject \@@letter             \do \chardef\@@letter     = 11  \fi
\ifnocontextobject \@@other              \do \chardef\@@other      = 12  \fi
\ifnocontextobject \@@active             \do \chardef\@@active     = 13  \fi
\ifnocontextobject \@@comment            \do \chardef\@@comment    = 14  \fi
```

\everyline \EveryLine \EveryPar In CONTEXT we use `\everypar` for special purposes and provide `\EveryPar` as an alternative. The same goes for `\everyline` and `\EveryLine`.

20
```
\ifnocontextobject \everyline            \do \newtoks\everyline          \fi
\ifnocontextobject \EveryPar             \do \let\EveryPar  = \everypar  \fi
\ifnocontextobject \EveryLine            \do \let\EveryLine = \everyline \fi
```

\!!... We reserve ourselves some scratch strings (i.e. macros).

21
```
\ifnocontextobject \!!stringa            \do \def\!!stringa     {}       \fi
\ifnocontextobject \!!stringb            \do \def\!!stringb     {}       \fi
\ifnocontextobject \!!stringc            \do \def\!!stringc     {}       \fi
\ifnocontextobject \!!stringd            \do \def\!!stringd     {}       \fi
```

\!!... The next set of definitions speed up processing a bit. Furthermore it saves memory.

22
```
\ifnocontextobject \!!zeropoint          \do \def\!!zeropoint   {0pt}    \fi
\ifnocontextobject \!!tenthousand        \do \def\!!tenthousand {10000}  \fi
```

23
```
\ifnocontextobject \!!width              \do \def\!!width        {width}  \fi
\ifnocontextobject \!!height             \do \def\!!height       {height} \fi
\ifnocontextobject \!!depth              \do \def\!!depth        {depth}  \fi
```

24
```
\ifnocontextobject \!!plus               \do \def\!!plus         {plus}   \fi
\ifnocontextobject \!!minus              \do \def\!!minus        {minus}  \fi
```

\smashbox The system modules offer a range of smashing macros, of which we only copied \smashbox.

25
```
\ifnocontextobject \smashbox \do
```

26
```
  \def\smashbox#1%
    {\wd#1=\!!zeropoint
     \ht#1=\!!zeropoint
     \dp#1=\!!zeropoint}
```

27
```
\fi
```

\dowithnextbox Also without further comment, we introduce a macro that gets the next box and does something usefull with it. Because the \afterassignment is executed inside the box, we have to use a \aftergroup too.

28
```
\ifnocontextobject \dowithnextbox \do
```

29
```
  \def\dowithnextbox#1%
    {\def\dodowithnextbox{#1}%
     \afterassignment\dododowithnextbox
     \setbox\nextbox}
```

30
```
  \def\dododowithnextbox%
    {\aftergroup\dodowithnextbox}
```

31
```
\fi
```

\setvalue
\getvalue

The next two macros expand their argument to **\argument**. The first one is used to define macro's the second one executes them.

```
\ifnocontextobject \setvalue \do

  \def\setvalue#1{\expandafter\def\csname#1\endcsname}
  \def\getvalue#1{\csname#1\endcsname}

\fi
```

\protected

The next command can be used as prefixed for commands that need protection during tests and writing to files. This is a very CONTEXT specific one.

```
\ifnocontextobject \unexpanded \do

  \let\unexpanded=\relax

\fi
```

\convertargument

The original one offers a bit more, like global assignment, the the next implementation is however a bit more byte saving.

```
\ifnocontextobject \convertargument \do

  \def\doconvertargument#1>{}

  \long\def\convertargument#1\to#2%
    {\long\def\convertedargument{#1}%
     \edef#2{\expandafter\doconvertargument\meaning\convertedargument}}

\fi
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

\forgetall Sometimes we have to disable interference of whatever kind of skips and mechanisms. The next macro resets some.

```
\ifnocontextobject \forgetall \do

  \def\forgetall%
    {\parskip\!!zeropoint
     \leftskip\!!zeropoint
     \parindent\!!zeropoint
     \everypar{}}

\fi
```

\withoutpt TEX lacks some real datastructure. We can however use ⟨*dimensions*⟩. This kind of trickery is needed when we want TEX to communicate with the outside world (by means of `\specials`).

```
\ifnocontextobject \withoutpt \do

  {\catcode`\.=\@@other
   \catcode`\p=\@@other
   \catcode`\t=\@@other
   \gdef\WITHOUTPT#1pt{#1}}

  \def\withoutpt#1%
    {\expandafter\WITHOUTPT#1}

  \def\ScaledPointsToBigPoints#1#2%
    {\scratchdimen=#1sp\relax
     \scratchdimen=.996264\scratchdimen
     \edef#2{\withoutpt{\the\scratchdimen}}}

\fi
```

\doprocessfile This macro takes three arguments: the file number, the filename and a macro that handles the content of a read line.

```
\ifnocontextobject \doprocessfile \do
```

```
  \def\doprocessfile#1#2#3%
    {\openin#1=#2\relax
     \def\doprocessline%
       {\ifeof#1%
          \def\doprocessline{\closein#1}%
        \else
          \read#1 to \fileline
          #3\relax
        \fi
        \doprocessline}%
     \doprocessline}
```

```
\fi
```

\uncatcodespecials This one is taken from the TEX book. The CONTEXT alternative is a bit different, but we hope this one works here.

```
\ifnocontextobject \uncatcodespecials \do
```

```
  \def\uncatcodespecials%
    {\def\do##1{\catcode`##1=12 }\dospecials}
```

```
\fi
```

That's it. Please forget this junk and take a look at how it should be done.

```
\protect
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

4.2 Verbatim

Because this module is quite independant of system macros, it can be used as a stand–alone verbatim environment.

1
```
\ifx \undefined \writestatus \input supp-mis.tex \fi
```

Verbatim typesetting, especially of TEX sources, is a non–trivial task. This is a direct results of the fact that characters can have ⟨*catcodes*⟩ other than 11 and such characters needs a special treatment. What for instance is TEX supposed to do when it encounters a `$` or an `#`? This module deals with these matters.

2
```
\writestatus{loading}{Context Support Macros / Verbatim}
```

The verbatim environment has some features, like coloring TEX text, seldom found in other environments. Especially when the output of TEX is viewed on an electronic medium, coloring has a positive influence on the readability of TEX sources, so we found it very acceptable to dedicate half of this module to typesetting TEX specific character sequences in color. In this module we'll also present some macro's for typesetting inline, display and file verbatim. The macro's are capable of handling `<tab>` too.

This module shows a few tricks that are often overseen by novice, like the use of the TEX primitive `\meaning`. First I'll show in what way the users are confronted with verbatim typesetting. Because we want to be able to test for symmetry and because we hate the method of closing down the verbatim mode with some strange active character, we will use the following construction for display verbatim:

```
\starttyping
The Dutch word 'typen' stands for 'typing', therefore in the Dutch version
one will not find the word 'verbatim'.
\stoptyping
```

In CONTEXT files can be typed with `\typefile` and inline verbatim can be accomplished with `\type`. This last command comes in many flavors:

```
We can say \type<<something>> or \type{something}. The first one is a bit
longer but also supports slanted typing, which accomplished by typing
\type<<a <<slanted>> word>>. We can also use commands to enhance the text
\type<<with <</bf boldfaced>> text>>. Just to be complete, we decided
to accept also \LaTeX\ alike verbatim, which means that \type+something+
and \type|something| are valid commands too. Of course we want the grouped
alternatives to process \type{hello {\bf big} world}} with braces.
```

In the core modules, we will build this support on top of this module. There these commands can be tuned with accompanying setup commands. There we can enable commands, slanted typing, control spaces, `<tab>`–handling and (here we are:) coloring. We can also setup surrounding white space and indenting. Here we'll only show some examples.

3
```
\unprotect
```

`\verbatimfont` When we are typesetting verbatim we use a non–proportional (mono spaced) font. Normally this font is available by calling `\tt`. In CONTEXT this command does a complete font–style switch. There we could have stuck with `\tttf`.

4
```
\ifx \undefined \verbatimfont \def\verbatimfont {\tt} \fi
```

\obeyedspace
\obeyedtab
\obeyedline
\obeyedpage

We have followed Knuth in naming macros that make `<space>`, `<newline>` and `<newpage>` active and assigning them `\obeysomething`, but first we set some default values.

```
\def\obeyedspace {\hbox{ }}
\def\obeyedtab   {\obeyedspace}
\def\obeyedline  {\par}
\def\obeyedpage  {\vfill\eject}
```

\controlspace
\setcontrolspaces

First we define `\obeyspaces`. When we want visible spaces (control spaces) we only have to adapt the definition of `\obeyedspace` to:

```
\def\controlspace {\hbox{\char32}}
```

```
\bgroup
\catcode`\ =\@@active
\gdef\obeyspaces{\catcode`\ =\@@active\def {\obeyedspace}}
\gdef\setcontrolspaces{\catcode`\ =\@@active\def {\controlspace}}
\egroup
```

\obeytabs
\obeylines
\obeypages
\ignoretabs
\ignorelines
\ignorepages

Next we take care of `<newline>` and `<newpage>` and because we want to be able to typeset listings that contain `<tab>`, we have to handle those too. Because we have to redefine the `<newpage>` character locally, we redefine the meaning of this (often already) active character.

```
\catcode`\^^L=\@@active \def^^L{\par}
```

```
\bgroup
```

```
\catcode`\^^I=\@@active
\catcode`\^^M=\@@active
\catcode`\^^L=\@@active
```

11
```
\gdef\obeytabs    {\catcode`\^^I=\@@active\def^^I{\obeyedtab}}
\gdef\obeylines   {\catcode`\^^M=\@@active\def^^M{\obeyedline}}
\gdef\obeypages   {\catcode`\^^L=\@@active\def^^L{\obeyedpage}}
```

12
```
\gdef\ignoretabs  {\catcode`\^^I=\@@active\def^^I{\obeyedspace}}
\gdef\ignorelines {\catcode`\^^M=\@@active\def^^M{\obeyedspace}}
\gdef\ignorepages {\catcode`\^^L=\@@active\def^^L{\obeyedline}}
```

13
```
\egroup
```

`\obeycharacters` We also predefine `\obeycharacters`, which will enable us to implement character–specific behavior, like colored verbatim.

14
```
\let\obeycharacters=\relax
```

`\settabskips` The macro `\settabskip` can be used to enable tab handling. Processing tabs is sometimes needed when one processes a plain ASCII listing. Tab handling slows down verbatim typesetting considerably.

15
```
\bgroup
```

16
```
\catcode`\^^I=\@@active
```

17
```
\gdef\settabskips%
  {\let\processverbatimline=\doprocesstabskipline
   \catcode`\^^I=\@@active
   \let^^I=\doprocesstabskip}
```

18
```
\egroup
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

\processinlineverbatim

Although the inline verbatim commands presented here will be extended and embedded in the core modules of CONTEXT, they can be used separately. Both grouped and character alternatives are provided but `<<` and nested braces are implemented in the core module. This commands takes one argument: the closing command.

```
\processinlineverbatim{\closingcommand}
```

One can define his own verbatim commands, which can be very simple:

```
\def\Verbatim {\processinlineverbatim\relax}
```

or a bit more more complex:

```
\def\GroupedVerbatim%
  {\bgroup
   \dosomeusefullthings
   \processinlineverbatim\egroup}
```

Before entering inline verbatim mode, we take care of the unwanted `<tab>`, `<newline>` and `<newpage>` characters and turn them into `<space>`. We need the double `\bgroup` construction to keep the closing command local.

19
```
\def\setupinlineverbatim%
  {\verbatimfont
   \let\obeytabs=\ignoretabs
   \let\obeylines=\ignorelines
   \let\obeypages=\ignorepages
   \setupcopyverbatim}
```

20
```
\def\doprocessinlineverbatim%
  {\ifx\next\bgroup
     \setupinlineverbatim
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

```
      \catcode`\{=\@@begingroup
      \catcode`\}=\@@endgroup
      \def\next{\let\next=}%
    \else
      \setupinlineverbatim
      \def\next##1{\catcode`##1=\@@endgroup}%
    \fi
    \next}
```

21
```
\def\processinlineverbatim#1%
  {\bgroup
   \localcatcodestrue % TeX processes paragraph's
   \def\endofverbatimcommand{#1\egroup}%
   \bgroup
   \aftergroup\endofverbatimcommand
   \futurelet\next\doprocessinlineverbatim}
```

\processdisplayverbatim

The closing command is executed afterwards as an internal command and therefore should not be given explicitly when typesetting inline verbatim.

We can define a display verbatim environment with the command `\processdisplayverbatim` in the following way:

```
   \processdisplayverbatim{\closingcommand}
```

For instance, we can define a simple command like:

```
   \def\BeginVerbatim {\processdisplayverbatim{EndVerbatim}}
```

But we can also do more advance things like:

```
   \def\BeginVerbatim {\bigskip \processdisplayverbatim{\EndVerbatim}}
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
  \def\EndVerbatim   {\bigskip}
```

When we compare these examples, we see that the backslash in the closing command is optional. One is free in actually defining a closing command. If one is defined, the command is executed after ending verbatim mode.

22
```
\def\processdisplayverbatim#1%
  {\par
   \bgroup
   \escapechar=-1
   \xdef\verbatimname{\string#1}%
   \egroup
   \def\endofdisplayverbatim{\csname\verbatimname\endcsname}%
   \bgroup
   \parindent\!!zeropoint
   \ifdim\lastskip<\parskip
     \removelastskip
     \vskip\parskip
   \fi
   \parskip\!!zeropoint
   \processingverbatimtrue
   \linepartrue
   \expandafter\let\csname\verbatimname\endcsname=\relax
   \edef\endofverbatimcommand{\csname\verbatimname\endcsname}%
   \edef\endofverbatimcommand{\meaning\endofverbatimcommand}%
   \verbatimfont
   \setupcopyverbatim
   \let\doverbatimline=\relax
   \copyverbatimline}
```

We save the closing sequence in \endofverbatimcommand in such a way that it can be compared on a line by line basis. For the conversion we use \meaning, which converts the line to non–expandable tokens. We reset \parskip, because we don't want inter–paragraph skips to creep into the verbatim source. Furthermore we \relax the line–processing macro while getting the rest of the first line. The initialization command \setupcopyverbatim does just what we expect it to do: it assigns all characters ⟨*catcode*⟩ 11. Next we switch to french spacing and call for obeyance.

23
```
\def\setupcopyverbatim%
  {\uncatcodecharacters
   \frenchspacing
   \obeyspaces
   \obeytabs
   \obeylines
   \obeycharacters}
```

\ifeightbitcharacters
\setcatcodes
\uncatcodespecials
\uncatcodecharacters

As its name says, \uncatcodecharacters resets the ⟨*catcode*⟩ of characters. When we use an upper bound of 127 or 255, depending in \ifeightbitcharacters. By counting down, we only have to use one counter. The macro \setcatcodes can be uses to set alternative values. The macro \resetspecialcharacters resets characters with special meanings. This macro is not used in the verbatim macros, but is best defined in this module.

24
```
\def\doprocesscatcodes#1%
  {\ifeightbitcharacters
     \scratchcounter=255
   \else
     \scratchcounter=127
   \fi
   \loop
     \savecatcode
     #1\relax
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
      \advance\scratchcounter by -1
      \ifnum\scratchcounter>-1
    \repeat
    \let\savecatcode=\relax
    \let\restorecatcodes=\dorestorecatcodes}
```

25
```
\def\uncatcodespecials%
  {\doprocesscatcodes
     {\ifnum\catcode\scratchcounter=\@@letter\relax\else
        \catcode\scratchcounter=\@@other
      \fi}%
    \catcode`\   =\@@space
    \catcode`\^^L=\@@ignore
    \catcode`\^^M=\@@endofline
    \catcode`\^^?=\@@ignore}
```

26
```
\def\setcatcodes#1%
  {\doprocesscatcodes
     {\catcode\scratchcounter=#1}}
```

27
```
\def\uncatcodecharacters%
  {\setcatcodes\@@letter}
```

\iflocalcatcodes
\restorecatcodes
\beginrestorecatcodes
\endrestorecatcodes

We're not finished dealing ⟨*catcodes*⟩ yet. In CONTEXT we use only one auxiliary file, which deals with tables of contents, registers, two pass tracking, references etc. This file, as well as files concerning graphics, is processed when needed, which can be in the mid of typesetting verbatim. However, when reading in data in verbatim mode, we should temporary restore the normal ⟨*catcodes*⟩, and that's exactly what the next macros do. Saving the catcodes can be disabled by saying `\localcatcodestrue`.

The previous macros call for `\savecatcode`, which is implemented as:

28
```
\newif\iflocalcatcodes
```

29
```
\def\savecatcode%
  {\iflocalcatcodes \else
     \expandafter\edef\csname @@cc@@\the\scratchcounter\endcsname%
       {\the\catcode\scratchcounter}%
   \fi}
```

It's counterpart is:

30
```
\def\restorecatcode%
  {\expandafter\catcode\expandafter\scratchcounter\expandafter=
     \csname @@cc@@\the\scratchcounter\endcsname}
```

When we want to restore ⟨*catcodes*⟩ we call for `\restorecatcodes`, which default to `\relax`

31
```
\let\restorecatcodes=\relax
```

or when we've saves things calls for:

32
```
\def\dorestorecatcodes%
  {\iflocalcatcodes \else
     \doprocesscatcodes\restorecatcode
   \fi}
```

We also provide an alternative, that forces grouping when needed. An application of this macros can be found in buffering data.

33
```
\def\beginrestorecatcodes%
  {\ifx\restorecatcodes\relax
     \let\endrestorecatcodes=\relax
   \else
     \bgroup
     \let\beginrestorecatcodes=\bgroup
     \let\endrestorecatcodes=\egroup
   \fi}
```

The main copying routine of display verbatim does an ordinary string–compare on the saved closing command and the current line. The space after `#1` in the definition of `\next` is essential! As a result of using `\obeylines`, we have to use `%`'s after each line but none after the first `#1`.

34
```
{\obeylines%
 \gdef\copyverbatimline#1
   {\ifx\doverbatimline\relax% gobble rest of the first line
      \let\doverbatimline=\dodoverbatimline%
      \def\next{\copyverbatimline}%
    \else%
      \def\next{#1 }%
      \ifx\next\emptyspace%
        \def\next%
          {\doemptyverbatimline{#1}%
           \copyverbatimline}%
      \else%
        \edef\next{\meaning\next}%
        \ifx\next\endofverbatimcommand%
```

```
          \def\next%
            {\egroup\endofdisplayverbatim}%
        \else%
          \def\next%
            {\doverbatimline{#1}%
             \copyverbatimline}%
        \fi%
      \fi%
  \fi%
  \next}}
```

The actual typesetting of a line is done by a separate macro, which enables us to implement `<tab>` handling. The `\do` and `\dodo` macros take care of the preceding `\parskip`, while skipping the rest of the first line. The `\relax` is used as an signal.

`\iflinepar` A careful reader will see that `\linepar` is reset. This boolean can be used to determine if the current line is the first line in a pseudo paragraph and this boolean is set after each empty line.

35
```
\newif\iflinepar
```

36
```
\def\dodoverbatimline#1%
  {\leavevmode\the\everyline\strut\processverbatimline{#1}%
   \EveryPar{}%
   \lineparfalse
   \obeyedline\par}
```

`\obeyemptylines` Empty lines in verbatim can lead to white space on top of a new page. Because this is not what we want, we turn them into vertical skips. This default behavior can be overruled by:

```
  \obeyemptylines
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

Although it would cost us only a few lines of code, we decided not to take care of multiple empty lines. When a (display) verbatim text contains more successive empty lines, this probably suits some purpose.

37
```
\bgroup
\catcode`\^^L=\@@active  \gdef\emptypage  {^^L}
\catcode`\^^M=\@@active  \gdef\emptyline  {^^M}
                         \gdef\emptyspace { }
\egroup
```

38
```
\def\doemptyverbatimline%
  {\vskip\ht\strutbox
   \vskip\dp\strutbox
   {\setbox0=\hbox{\the\everyline}}%
   \linepartrue}
```

39
```
\def\obeyemptylines%
  {\def\doemptyverbatimline{\doverbatimline}}
```

TEX does not offer `\everyline`, which is a direct result of its advanced multi–pass paragraph typesetting mechanism. Because in verbatim mode paragraphs and lines are more or less equal, we can easily implement our own simple `\everyline` support.

`\EveryPar` `\EveryLine`

In this module we've reserved `\everypar` for the things to be done with paragraphs and `\everyline` for line specific actions. In CONTEXT however, we use `\everypar` for placing side- and columnfloats, inhibiting indentation and some other purposes. In verbatim mode, every line becomes a paragraph, which means that `\everypar` is executed frequently. To be sure, the user specific use of both `\everyline` and `\everypar` is implemented by means of `\EveryLine` and `\EveryPar`.

We still have to take care of the `<tab>`. A `<tab>` takes eight spaces and a `<space>` normally has a width of 0.5 em. Because we can be halfway a tabulation, we must keep track of the position. This takes time, especially when we print complete files, therefore we `\relax` this mechanism by default.

40
```
\def\doprocesstabskip%
  {\obeyedspace % \hskip.5em  or  \hbox to .5em{}
   \ifdone
     \advance\scratchcounter by 1
     \let\next=\doprocesstabskip
     \donefalse
   \else\ifnum\scratchcounter>7\relax
     \let\next=\relax
   \else
     \advance\scratchcounter 1\relax
     \let\next=\doprocesstabskip
   \fi\fi
   \next}
```

41
```
\def\dodoprocesstabskipline#1#2\endoftabskipping%
  {\ifnum\scratchcounter>7\relax
     \scratchcounter=1\relax
     \donetrue
   \else
     \advance\scratchcounter 1\relax
     \donefalse
   \fi
   \ifx#1\relax
     \let\next=\relax
   \else
     \def\next{#1\dodoprocesstabskipline#2\endoftabskipping}%
   \fi
   \next}
```

42
```
\let\endoftabskipping    = \relax
\let\processverbatimline = \relax
```

43
```
\def\doprocesstabskipline#1%
  {\bgroup
   \scratchcounter=1\relax
   \dodoprocesstabskipline#1\relax\endoftabskipping
   \egroup}
```

\processfileverbatim

The verbatim typesetting of files is done on a bit different basis. This time we don't check for a closing command, but look for `<eof>` and when we've met, we make sure it does not turn into an empty line.

```
   \processfileverbatim{filename}
```

Typesetting a file in most cases results in more than one page. Because we don't want problems with files that are read in during the construction of the page, we set `\ifprocessingverbatim`, so the output routine can adapt its behavior. Originally we used `\scratchread`, but because we want to support nesting, we decided to use a separate input file.

44
```
\newif\ifprocessingverbatim
```

45
```
\newread\verbatiminput
```

46
```
\def\processfileverbatim#1%
  {\par
   \bgroup
   \parindent\!!zeropoint
   \ifdim\lastskip<\parskip
     \removelastskip
     \vskip\parskip
```

```
\fi
\parskip\!!zeropoint
\processingverbatimtrue
\linepartrue
\uncatcodecharacters
\verbatimfont
\frenchspacing
\obeyspaces
\obeytabs
\obeylines
\obeypages
\obeycharacters
\openin\verbatiminput=#1%
\def\doreadline%
  {\read\verbatiminput to \next
   \ifeof\verbatiminput
     % we don't want <eof> to be treated as <crlf>
   \else\ifx\next\emptyline
     \expandafter\doemptyverbatimline\expandafter{\next}%
   \else\ifx\next\emptypage
     \expandafter\doemptyverbatimline\expandafter{\next}%
   \else
     \expandafter\dodoverbatimline\expandafter{\next}%
   \fi\fi\fi
   \readline}%
\def\readline%
  {\ifeof\verbatiminput
     \let\next=\relax
   \else
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
        \let\next=\doreadline
      \fi
      \next}%
   \readline
   \closein\verbatiminput
   \egroup
   \ignorespaces}
```

These macro's can be used to construct the commands we mentioned in the beginning of this documentation. We leave this to the fantasy of the reader and only show some PLAIN TEX alternatives for display verbatim and listings. We define three commands for typesetting inline text, display text and files verbatim. The inline alternative also accepts user supplied delimiters.

```
   \type{text}

   \starttyping
   ... verbatim text ...
   \stoptyping

   \typefile{filename}
```

We can turn on the options by:

```
   \controlspacetrue
   \verbatimtabstrue
   \prettyverbatimtrue
```

Here is the implementation:

47
```
\newif\ifcontrolspace
\newif\ifverbatimtabs
```

```
\newif\ifprettyverbatim
```

48
```
\def\presettyping%
  {\ifcontrolspace
     \let\obeyspace=\setcontrolspace
   \fi
   \ifverbatimtabs
     \let\obeytabs=\settabskips
   \fi
   \ifprettyverbatim
     \let\obeycharacters=\setupprettytextype
   \fi}
```

49
```
\def\type%
  {\bgroup
   \presettyping
   \processinlineverbatim{\egroup}}
```

50
```
\def\starttyping%
  {\bgroup
   \presettyping
   \processdisplayverbatim{\stoptyping}}
```

51
```
\def\stoptyping%
  {\egroup}
```

52
```
\def\typefile#1%
  {\bgroup
   \presettyping
   \processfileverbatim{#1}%
   \egroup}
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

One can use the different `\obeysomething` commands to influence the behavior of these macro's. We use for instance `\obeycharacters` for making `/` an active character when we want to include typesetting commands.

We'll spend the remainder of this article on coloring the verbatim text. At PRAGMA we use the integrated environment TEXEDIT for editing and processing TEX documents.[2] This program also supports real time spell checking and TEX based file management. Although definitely not exclusive, the programs cooperate nicely with CONTEXT. Because TEX can be considered a tool for experts, we've tried to put as less a burden on non–technical users as possible. This is accomplished in the following ways:

- We've added some trivial symmetry checking to TEXEDIT. Sources are checked for the use of brackets, braces, begin–end and start–stop like constructions, with or without arguments.
- Although TEX is very tolerant to unformatted input, we stimulate users to make the ASCII source as clean as possible. Many sources I've seen in distribution sets look so awful, that I sometimes wonder how people get them working. In our opinion, a good–looking source leads to less errors.
- We use parameter driven setups and make the commands as tolerant as possible. We don't accept commands that don't look nice in ASCII.
- Finally —I could have added some more— we use color.

When in spell–checking–mode, the words spelled correctly are shown in *green*, the unknown or wrongly spelled words are in *red* and upto four categories of words, for instance passive verbs and nouns, become *blue* (or cyan) or *yellow*. Short and nearly always correct words are in white (on a black screen). This makes checking–on–the–fly very easy and convenient, especially because we place the accents automatically.

In TEX–mode we show TEX–specific tokens and sequences of tokens in appropriate colors and again we use four colors. We use those colors in a way that supports parameter driven setups, table typesetting and easy visual checking of symmetry. Furthermore the text becomes more readable.

[2] TEXEDIT has been operative since 1991.

color	characters that are influenced
red	`{ } $`
green	`\this \!!that \??these \@@those`
yellow	`` ‘ ’ ~ ^ _ & / + - | % ``
blue	`( ) # [ ] " < > =`

Macro–definition and style files often look quite green, because they contain many calls to macros. Pure text files on the other hand are mostly white (on the screen) and color clearly shows their structure.

When I prepared the interactive PDF manuals of CONTEXT, TEXEDIT and PPCHTEX (1995), I decided to include the original source text of the manuals as an appendix. At every chapter or (sub)section the reader can go to the corresponding line in the source, just to see how things were done in TEX. Of course, the reader can jump from the to corresponding typeset text too.

Confronted with those long (boring) sources, I decided that a colored output, in accordance with TEXEDIT would be nice. It would not only visually add some quality to the manual, but also make the sources more readable.

Apart from a lot of ⟨*catcode*⟩–magic, programming the color macros was surprisingly easy. Although the macro's are hooked into the standard CONTEXT verbatim mechanism, they are set up in a way that embedding them in another verbatim environment is possible.

We can turn on coloring by reassigning `\obeycharacters`:

```
\let\obeycharacters=\setupprettytextype
```

During pretty typesetting we can be in two states: *command* and *parameter*. The first condition becomes true if we encounter a backslash, the second state is entered when we meet a `#`.

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

53
```
\newif\ifintexcommand
\newif\ifintexparameter
```

\splittexparameters

The mechanism described here, is meant to be used with color. It is nevertheless possible to use different fonts instead of distinctive colors. When using color, it's better to end parameter mode after the `#`. When on the other hand we use a slanted typeface for the hashmark, then a slanted number looks better.

54
```
\newif\ifsplittexparameters   \splittexparameterstrue
```

\splittexcontrols

With `\splittexcontrols` we can influence the way control characters are processed in macro names. By default, the `^^` part is uncolored. When this boolean is set to false, they get the same color as the other characters.

55
```
\newif\ifsplittexcontrols     \splittexcontrolstrue
```

The next boolean is used for internal purposes only and keeps track of the length of the name. Because two–character sequences starting with a backslash are always seen as a command.

56
```
\newif\iffirstintexcommand
```

We use a maximum of four colors because more colors will distract too much. In the following table we show the logical names of the colors, their color and *rgb* values.

identifier	**color**	**r**	**g**	**b**	**bw**
texprettyone	red	0.9	0.0	0.0	0.30
texprettytwo	green	0.0	0.8	0.0	0.45
texprettythree	yellow	0.0	0.0	0.9	0.60
texprettyfour	blue	0.8	0.8	0.6	0.75

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

This following poor mans implementation of color is based on PostScript. One can of course use grayscales too. In the core modules these macros are redefined to using the color mechanism present in CONTEXT.

57
```
\def\setcolorverbatim%
  {\splittexparameterstrue
   \def\texprettyone   {.9 .0 .0 }       % red
   \def\texprettytwo   {.0 .8 .0 }       % green
   \def\texprettythree {.0 .0 .9 }       % blue
   \def\texprettyfour  {.8 .8 .6 }       % yellow
   \def\texbeginofpretty[##1]%
     {\special{ps:: \csname##1\endcsname setrgbcolor}}
   \def\texendofpretty%
     {\special{ps:: 0 0 0 setrgbcolor}}} % black
```

58
```
\def\setgrayverbatim%
  {\splittexparameterstrue
   \def\texprettyone   {.30 }            % gray
   \def\texprettytwo   {.45 }            % gray
   \def\texprettythree {.60 }            % gray
   \def\texprettyfour  {.75 }            % gray
   \def\texbeginofpretty[##1]%
     {\special{ps:: \csname##1\endcsname setgray}}
   \def\texendofpretty%
     {\special{ps:: 0 setgray}}}         % black
```

One can redefine these two commands after loading this module. When available, one can also use appropriate font–switch macro's. We default to color.

59
```
\setcolorverbatim
```

Here come the commands that are responsible for entering and leaving the two states. As we can see, they've got much in common.

60
```
\def\texbeginofcommand%
  {\texendofparameter
   \ifintexcommand
   \else
     \global\intexcommandtrue
     \global\firstintexcommandtrue
     \texbeginofpretty[texprettytwo]%
   \fi}
```

61
```
\def\texendofcommand%
  {\ifintexcommand
     \texendofpretty
     \global\intexcommandfalse
     \global\firstintexcommandfalse
   \fi}
```

62
```
\def\texbeginofparameter%
  {\texendofcommand
   \ifintexparameter
   \else
     \global\intexparametertrue
     \texbeginofpretty[texprettythree]%
   \fi}
```

63
```
\def\texendofparameter%
  {\ifintexparameter
```

```
    \texendofpretty
    \global\intexparameterfalse
  \fi}
```

We've got nine types of characters. The first type concerns the grouping characters that become red and type seven takes care of the backslash. Type eight is the most recently added one and handles the control characters starting with `^^`. In the definition part at the end of this module we can see how characters are organized by type.

64
```
\def\ifnotfirstintexcommand#1%
  {\iffirstintexcommand
     \string#1%
     \texendofcommand
   \else}
```

65
```
\def\textypeone#1%
  {\ifnotfirstintexcommand#1%
     \texendofcommand
     \texendofparameter
     \texbeginofpretty[texprettyone]\string#1\texendofpretty
   \fi}
```

66
```
\def\textypetwo#1%
  {\ifnotfirstintexcommand#1%
     \texendofcommand
     \texendofparameter
     \texbeginofpretty[texprettythree]\string#1\texendofpretty
   \fi}
```

67
```
\def\textypethree#1%
  {\ifnotfirstintexcommand#1%
```

```
    \texendofcommand
    \texendofparameter
    \texbeginofpretty[texprettyfour]\string#1\texendofpretty
  \fi}
```

68
```
\def\textypefour#1%
  {\ifnotfirstintexcommand#1%
    \texendofcommand
    \texendofparameter
    \string#1%
  \fi}
```

69
```
\def\textypefive#1%
  {\ifnotfirstintexcommand#1%
    \texbeginofparameter
    \string#1%
  \fi}
```

70
```
\def\textypesix#1%
  {\ifnotfirstintexcommand#1%
    \ifintexparameter
      \ifsplittexparameters
        \texendofparameter
        \string#1%
      \else
        \string#1%
        \texendofparameter
      \fi
    \else
      \texendofcommand
      \string#1%
    \fi
  \fi}
```

71
```
\def\textypeseven#1%
  {\ifnotfirstintexcommand#1%
     \texbeginofcommand
     \string#1%
   \fi}
```

72
```
\def\textypeeight#1#2%
  {\texendofparameter
   \ifx#1#2%
     \ifsplittexcontrols
       \ifintexcommand
         \texendofcommand
         \string#1\string#1%
         \texbeginofcommand
       \else
         \string#1\string#2%
       \fi
     \else
       \string#1\string#1%
     \fi
   \else
     \ifintexcommand
       \firstintexcommandfalse
       \string#1#2%
     \else
       \textypethree#1#2%
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

```
    \fi
  \fi}
```

73
```
\def\textypenine#1%
  {\texendofparameter
   \global\firstintexcommandfalse
   \string#1}
```

We have to take care of the control characters we mentioned before. We obey their old values but only after ending our two states.

74
```
\def\texsetcontrols%
  {\global\let\oldobeyedspace = \obeyedspace
   \global\let\oldobeyedline  = \obeyedline
   \global\let\oldobeyedpage  = \obeyedpage
   \def\obeyedspace%
     {\texendofcommand
      \texendofparameter
      \oldobeyedspace}%
   \def\obeyedline%
     {\texendofcommand
      \texendofparameter
      \oldobeyedline}%
   \def\obeyedpage%
     {\texendofcommand
      \texendofparameter
      \oldobeyedpage}%
   \let\obeytabs=\ignoretabs}
```

Next comes the tough part. We have to change the ⟨*catcode*⟩ of each character. These macro's are tuned for speed and simplicity. When viewed in color they look quite simple.

contents register context syst mult supp lang font colo spec core cont m s exit go back

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

75
```
\def\setupprettytextype%
  {\texsetcontrols
   \texsetspecialpretty
   \texsetalphabetpretty
   \texsetextrapretty}
```

When handling the lowercase characters, we cannot use lowercased macro names. This means that we have to redefine some well known macros, like `\bgroup`.

76
```
\def\texpresetcatcode%
  {\def\\##1%
     {\expandafter\catcode\expandafter`\csname##1\endcsname\@@active}}
```

77
```
\def\texsettypenine%
  {\def\\##1%
     {\def##1{\textypenine##1}}}
```

78
```
\bgroup
  \bgroup
    \gdef\texpresetalphapretty%
      {\texpresetcatcode
       \\A\\B\\C\\D\\E\\F\\G\\H\\I\\J\\K\\L\\M%
       \\N\\O\\P\\Q\\R\\S\\T\\U\\V\\W\\X\\Y\\Z}
    \texpresetalphapretty
    \gdef\texsetalphapretty%
      {\texpresetalphapretty
       \texsettypenine
       \\A\\B\\C\\D\\E\\F\\G\\H\\I\\J\\K\\L\\M%
       \\N\\O\\P\\Q\\R\\S\\T\\U\\V\\W\\X\\Y\\Z}
  \egroup
  \global\let\TEXPRESETCATCODE = \texpresetcatcode
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
  \global\let\TEXSETTYPENINE    = \texsettypenine
  \global\let\BGROUP            = \bgroup
  \global\let\EGROUP            = \egroup
  \global\let\GDEF              = \gdef
  \BGROUP
    \GDEF\TEXPRESETALPHAPRETTY%
      {\TEXPRESETCATCODE
       \\a\\b\\c\\d\\e\\f\\g\\h\\i\\j\\k\\l\\m%
       \\n\\o\\p\\q\\r\\s\\t\\u\\v\\w\\x\\y\\z}
    \TEXPRESETALPHAPRETTY
    \GDEF\TEXSETALPHAPRETTY%
      {\TEXPRESETALPHAPRETTY
       \TEXSETTYPENINE
       \\a\\b\\c\\d\\e\\f\\g\\h\\i\\j\\k\\l\\m%
       \\n\\o\\p\\q\\r\\s\\t\\u\\v\\w\\x\\y\\z}
  \EGROUP
  \gdef\texsetalphabetpretty%
    {\texsetalphapretty
     \TEXSETALPHAPRETTY}
\egroup
```

Macro names normally only may contain characters, but in unprotected state we can also use the characters `@`, `!` and `?`. Of course they are only colored (green) when they are part of a name.

79

```
\bgroup
  \gdef\texpresetextrapretty%
    {\texpresetcatcode
     \\?\\!\\@}
  \texpresetextrapretty
  \gdef\texsetextrapretty%
```

```
    {\texpresetextrapretty
     \texsettypenine
     \\?\\!\\@}
\egroup
```

Here comes the main specification routine. In this macro we also have to change the escape character to `!` and use `X`, `Y` and `Z` for grouping and ignoring, which makes the result a bit less readable. Plain TeX defines `\+` as an outer macro, so we have to redefine this one too.

```
\def\+{\tabalign}
```

```
\bgroup
  \gdef\texpresetspecialpretty%
    {\def\\##1{\catcode`##1\@@active}%
     \\\[\\\]\\\=\\\<\\\>\\\#\\\(\\\)\\\"%
     \\\$\\\{\\\}%
     \\\-\\\+\\\|\\\%\\\/\\\_\\\^\\\&\\\~\\\'\\\`%
     \\\.\\\,\\\:\\\;%
     \\\*%
     \\\1\\\2\\\3\\\4\\\5\\\6\\\7\\\8\\\9%
     \\\\}
  \catcode`\X=\the\catcode`\{
  \catcode`\Y=\the\catcode`\}
  \catcode`\Z=\the\catcode`\%
  \gdef\texsetsometypes%
    {\def\!##1##2{\def##1{##2{##1}}}}%
  XZ
   \catcode`\!=\@@escape
   !texpresetspecialpretty
   !gdef!texsetspecialpretty
```

```
     XZ
      !texpresetspecialpretty
      !texsetsometypes
      !! $ !textypeone   !! { !textypeone   !! } !textypeone
      !! [ !textypetwo   !! ] !textypetwo   !! ( !textypetwo   !! ) !textypetwo
      !! = !textypetwo   !! < !textypetwo   !! > !textypetwo   !! " !textypetwo
      !! - !textypethree !! + !textypethree !! / !textypethree
      !! | !textypethree !! % !textypethree !! ' !textypethree !! ` !textypethree
      !! _ !textypethree !! ^ !textypethree !! & !textypethree !! ~ !textypethree
      !! . !textypefour  !! , !textypefour  !! : !textypefour  !! ; !textypefour
      !! * !textypefour
      !! # !textypefive
      !! 1 !textypesix   !! 2 !textypesix   !! 3 !textypesix
      !! 4 !textypesix   !! 5 !textypesix   !! 6 !textypesix
      !! 7 !textypesix   !! 8 !textypesix   !! 9 !textypesix
      !! \ !textypeseven
      !! ^ !textypeeight
     YZ
  YZ
\egroup
```

This text was published in the MAPS of the dutch TEX users group NTG. In that article, the verbatim part of the text was set with the following commands for the examples:

```
   \def\starttypen% We simplify the \ConTeXt\ macro.
     {\bgroup
      \everypar{} % We disable some troublesome mechanisms.
      \advance\leftskip by 1em
      \processdisplayverbatim{\stoptypen}}
```

```
\def\stoptypen%
  {\egroup}
```

The implementation itself was typeset with:

```
\def\startdefinition%
  {\bgroup
   \everypar{} % Again we disable some troublesome mechanisms.
   \let\obeycharacters=\setupprettytextype
   \EveryPar{\showparagraphcounter}%
   \EveryLine{\showlinecounter}%
   \verbatimcorps
   \processdisplayverbatim{\stopdefinition}}

\def\stopdefinition%
  {\egroup}
```

And because we have both `\EveryPar` and `\EveryLine` available, we can implement a dual numbering mechanism:

```
\newcount\paragraphcounter
\newcount\linecounter

\def\showparagraphcounter%
  {\llap
     {\bgroup
      \counterfont
      \hbox to 4em
        {\global\advance\paragraphcounter by 1
         \hss \the\paragraphcounter \hskip2em}%
      \egroup
```

```
        \hskip1em}}

  \def\showlinecounter%
    {\llap
       {\bgroup
        \counterfont
        \hbox to 2em
          {\global\advance\linecounter by 1
           \hss \the\linecounter}%
        \egroup
        \hskip1em}}
```

One may have noticed that the `\EveryPar` is only executed once, because we consider each piece of verbatim as one paragraph. When one wants to take the empty lines into account, the following assignments are appropriate:

```
  \EveryLine
    {\iflinepar
       \showparagraphcounter
     \fi
     \showlinecounter}
```

In this case, nothing has to be assigned to `\EveryPar`, maybe except of just another extra numbering scheme. The macros used to typeset this documentation are a bit more complicated, because we have to take take 'long' margin lists into account. When such a list exceeds the previous pargraph we postpone placement of the paragraph number till there's room. This way so it does not clash with the margin words.

Normally such commands have to be embedded in a decent setup structure, where options can be set at will.

Now let's summarize the most important commands.

```
\processinlineverbatim{\closingcommand}
\processdisplayverbatim{\closingcommand}
\processfileverbatim{filename}
```

We can satisfy our own specific needs with the following interfacing macro's:

```
\obeyspaces  \obeytabs  \obeylines  \obeypages  \obeycharacters
```

Some needs are fulfilled already with:

```
\setcontrolspace  \settabskips  \setupprettytextype
```

lines can be enhanced with ornaments using:

```
\everypar  \everyline  \iflinepar
```

and color support is implemented by:

```
\texbeginofpretty[#1] ... \texendofpretty
```

We can influence the verbatim environment with the following macro and booleans:

```
\obeyemptylines  \splittexparameters...  \splittexcontrols...
```

The color support macro can be redefined by the user. The parameter `#1` can be one of the four 'fixed' identifiers *texprettyone*, *texprettytwo*, *texprettythree* and *texprettyfour*. We have implemented a more or less general PostScript color support mechanism, using `specials`. One can toggle between color and grayscale with:

```
\setgrayverbatim  \setcolorverbatim
```

\permitshiftedendofver..

We did not mention one drawback of the mechanism described here. The closing command must start at the first position of the line. In CONTEXT we will not have this drawback, because we can test if the end command is a substring of the current line. The testing is done by two of the support macros, which of course are not available in a stand alone application of this module.

82
```
\ifx \undefined \doifinstringelse \else
```

83
```
\def\processdisplayverbatim#1%
  {\par
   \bgroup
   \escapechar=-1
   \xdef\verbatimname{\string#1}%
   \egroup
   \def\endofdisplayverbatim{\csname\verbatimname\endcsname}%
   \bgroup
   \parindent\!!zeropoint
   \ifdim\lastskip<\parskip
     \removelastskip
     \vskip\parskip
   \fi
   \parskip\!!zeropoint
   \processingverbatimtrue
   \expandafter\let\csname\verbatimname\endcsname=\relax
   \expandafter\convertargument\csname\verbatimname\endcsname
     \to\endofverbatimcommand
   \verbatimfont
   \setupcopyverbatim
   \let\doverbatimline=\relax
   \copyverbatimline}
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

84
```
\let\doifendofverbatim=\doifelse
```

85
```
\def\permitshiftedendofverbatim%
  {\let\doifendofverbatim=\doifinstringelse}
```

86
```
{\obeylines%
 \gdef\copyverbatimline#1
   {\ifx\doverbatimline\relax% gobble rest of the first line
      \let\doverbatimline=\dodoverbatimline%
      \def\next{\copyverbatimline}%
    \else%
       \convertargument#1 \to\next%
       \ifx\next\emptyspace%
         \def\next%
           {\doemptyverbatimline{#1}%
            \copyverbatimline}%
       \else%
         \doifendofverbatim{\endofverbatimcommand}{\next}%
           {\def\next%
              {\egroup\endofdisplayverbatim}}%
           {\def\next%
              {\doverbatimline{#1}%
               \copyverbatimline}}%
       \fi%
   \fi%
   \next}}
```

87
```
\fi
```

88
```
\protect
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

4.3 Visualization

Although an integral part of CONTEXT, this module is one of the support modules. Its stand alone character permits use in PLAIN TEX or TEX based macropackages.

This module is still in development. Depending on my personal need and those of whoever uses it, the macros will be improved in terms of visualization, efficiency and compatibility.

1
```
\ifx \undefined \writestatus \input supp-mis.tex \fi
```

One of the strong points of TEX is abstraction of textual input. When macros are defined well and do what we want them to do, we will seldom need the tools present in What You See Is What You Get systems. For instance, when entering text we don't need rulers, because no manual shifting and/or alignment of text is needed. On the other hand, when we are designing macros or specifying layout elements, some insight in TEX's advanced spacing, kerning, filling, boxing and punishment abilities will be handy. That's why we've implemented a mechanism that shows some of the inner secrets of TEX.

2
```
\writestatus{loading}{Context Support Macros / Visualization}
```

In this module we are going to redefine some TEX primitives and PLAIN macro's. Their original meaning is saved in macros with corresponding names, preceded by `normal`. These original macros are (1) used to temporary restore the old values when needed and (2) used to prevent recursive calls in the macros that replace them.

3
```
\unprotect
```

\normalhbox
\normalvbox
\normalvtop

There are three types of boxes, one horizontal and two vertical in nature. As we will see later on, all three types are to be handled according to their orientation and baseline behavior. Especially `\vtop`'s need our special attention.

4
```
\let\normalhbox      = \hbox
\let\normalvbox      = \vbox
\let\normalvtop      = \vtop
\let\normalvcenter   = \vcenter
```

\normalhskip
\normalvskip

Next come the flexible skips, which come in two flavors too. Like boxes these are handled with TEX primitives.

5
```
\let\normalhskip     = \hskip
\let\normalvskip     = \vskip
```

\normalpenalty
\normalkern

Both penalties and kerns are taken care of by mode sensitive primitives. This means that when making them visible, we have to take the current mode into account.

6
```
\let\normalpenalty   = \penalty
\let\normalkern      = \kern
```

\normalhglue
\normalvglue

Glues on the other hand are macro's defined in PLAIN TEX. As we will see, their definitions make the implementation of their visible counterparts a bit more TEXnical.

7
```
\let\normalhglue     = \hglue
\let\normalvglue     = \vglue
```

\normalmkern
\normalmskip

Math mode has its own spacing primitives, preceded by `m`. Due to the relation with the current font and the way math is typeset, their unit `mu` is not compatible with other dimensions. As a result, the visual appearance of these primitives is kept primitive too.

8
```
\let\normalmkern    = \mkern
\let\normalmskip    = \mskip
```

\hfilneg
\vfilneg

Fills can be made visible quite easy. We only need some additional negation macros. Because PLAIN TEX only offers `\hfilneg` and `\vfilneg`, we define our own alternative double `ll`'ed ones.

9
```
\def\hfillneg%
  {\normalhskip\!!zeropoint \!!plus-1fill\relax}
```

10
```
\def\vfillneg%
  {\normalvskip\!!zeropoint \!!plus-1fill\relax}
```

\normalhss
\normalhfil
\normalhfill
\normalvss
\normalvfil
\normalvfill

The positive stretch primitives are used independant and in combination with `\leaders`.

11
```
\let\normalhss      = \hss
\let\normalhfil     = \hfil
\let\normalhfill    = \hfill
\let\normalvss      = \vss
\let\normalvfil     = \vfil
\let\normalvfill    = \vfill
```

\normalhfilneg
\normalhfillneg
\normalvfilneg
\normalvfillneg

Keep in mind that both **\hfillneg** and **\vfillneg** are not part of PLAIN TEX and therefore not documented in standard TEX documentation. They can nevertheless be used at will.

12
```
\let\normalhfilneg  = \hfilneg
\let\normalhfillneg = \hfillneg
\let\normalvfilneg  = \vfilneg
\let\normalvfillneg = \vfillneg
```

Visualization is not always wanted. Instead of turning this option off in those (unpredictable) situations, we just redefine a few PLAIN macros.

13
```
\def\rlap#1{\normalhbox to \!!zeropoint{#1\normalhss}}
\def\llap#1{\normalhbox to \!!zeropoint{\normalhss#1}}
```

14
```
\def~{\normalpenalty\!!tenthousand\ }
```

\makeruledbox

Ruled boxes can be typeset is many ways. Here we present just one alternative. This implementation may be a little complicated, but it supports all three kind of boxes. The next command expects a ⟨*box*⟩ specification, like:

```
\makeruledbox0
```

\baselinerule
\baselinefill
\baselinesmash

We can make the baseline of a box visible, both dashed and as a rule. Normally the line is drawn on top of the baseline, but a smashed alternative is offered too. If we want them all, we just say:

```
\baselineruletrue
\baselinefilltrue
\baselinesmashtrue
```

At the cost of some overhead these alternatives are implemented using **\if**'s:

15
```
\newif\ifbaselinerule  \baselineruletrue
\newif\ifbaselinefill  \baselinefillfalse
\newif\ifbaselinesmash \baselinesmashfalse
```

supp-mis supp-ver supp-vis supp-lan supp-pdf supp-spe supp-mps supp-tpi supp-fil supp-ini supp-box supp-mrk supp-mul supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

\iftoprule
\ifbottomrule
\ifleftrule
\ifrightrule

Rules can be turned on and off, but by default we have:

```
\topruletrue
\bottomruletrue
\leftruletrue
\rightruletrue
```

As we see below:

16
```
\newif\iftoprule        \topruletrue
\newif\ifbottomrule     \bottomruletrue
\newif\ifleftrule       \leftruletrue
\newif\ifrightrule      \rightruletrue
```

\boxrulewidth

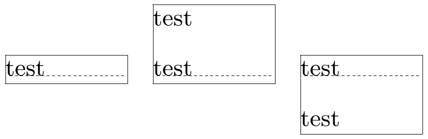
The width in the surrounding rules can be specified by assigning an apropriate value to the dimension used. This module defaults the width to:

```
\boxrulewidth=.2pt
```

Although we are already low on ⟨*dimensions*⟩ it's best to spend one here, mainly because it enables easy manipulation, like multiplication by a given factor.

17
```
\newdimen\boxrulewidth \boxrulewidth=.2pt
```

The core macro `\makeruledbox` looks a bit hefty. The manipulation at the end is needed because we want to preserve both the mode and the baseline. This means that `\vtop`'s and `\vbox`'es behave the way we expect them to do.



supp-mis supp-ver supp-vis supp-lan supp-pdf supp-spe supp-mps supp-tpi supp-fil supp-ini supp-box supp-mrk supp-mul supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

The `\cleaders` part of the macro is responsible for the visual baseline. The `\normalhfill` belongs to this primitive too. By storing and restoring the height and depth of box `#1`, we preserve the mode.

18

```
\def\makeruledbox#1%
  {\edef\ruledheight {\the\ht#1}%
   \edef\ruleddepth  {\the\dp#1}%
   \edef\ruledwidth  {\the\wd#1}%
   \setbox\scratchbox=\normalvbox
     {\dontcomplain
      \offinterlineskip
      \hrule
        \!!height\boxrulewidth
        \iftoprule\else\!!width\!!zeropoint\fi
      \normalvskip-\boxrulewidth
      \normalhbox to \ruledwidth
        {\vrule
           \!!height\ruledheight
           \!!depth\ruleddepth
           \!!width\ifleftrule\else0\fi\boxrulewidth
         \ifdim\ruledheight>\!!zeropoint \else \baselinerulefalse \fi
         \ifdim\ruleddepth>\!!zeropoint \else \baselinerulefalse \fi
         \ifbaselinerule
           \ifdim\ruledwidth<20\boxrulewidth
             \baselinefilltrue
           \fi
           \cleaders
             \ifbaselinefill
               \hrule
                 \ifbaselinesmash
```

supp-mis supp-ver supp-vis supp-lan supp-pdf supp-spe supp-mps supp-tpi supp-fil supp-ini supp-box supp-mrk supp-mul supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
                    \!!height\boxrulewidth
                  \else
                    \!!height.5\boxrulewidth
                    \!!depth.5\boxrulewidth
                  \fi
              \else
                \normalhbox
                  {\normalhskip2.5\boxrulewidth
                   \vrule
                   \ifbaselinesmash
                     \!!height\boxrulewidth
                   \else
                     \!!height.5\boxrulewidth
                     \!!depth.5\boxrulewidth
                   \fi
                   \!!width5\boxrulewidth
                   \normalhskip2.5\boxrulewidth}%
              \fi
          \fi
          \normalhfill
          \vrule
            \!!width\ifrightrule\else0\fi\boxrulewidth}%
        \normalvskip-\boxrulewidth
        \hrule
          \!!height\boxrulewidth
          \ifbottomrule\else\!!width\!!zeropoint\fi}%
    \wd#1=\!!zeropoint
    \setbox#1=\ifhbox#1\normalhbox\else\normalvbox\fi
      {\normalhbox{\box#1\lower\ruleddepth\box\scratchbox}}%
```

```
  \ht#1=\ruledheight
  \wd#1=\ruledwidth
  \dp#1=\ruleddepth}
```

Just in case one didn't notice: the rules are in fact layed over the box. This way the contents of a box cannot visually interfere with the rules around (upon) it. A more advanced version of ruled boxes can be found in one of the core modules of CONTEXT. There we take offsets, color, rounded corners, backgrounds and alignment into account too.

\ruledhbox
\ruledvbox
\ruledvtop
\ruledvcenter

These macro's can be used instead of **\hbox**, **\vbox**, **\vtop** and, when in math mode, **\vcenter**. They just do what their names state. Using an auxiliary macro would save us a few words of memory, but it would make their appearance even more obscure.

one two three four five

```
   \hbox
     {\strut
      one
      two
      \hbox{three}
      four
      five}
```

19
```
\def\ruledhbox%
  {\normalhbox\bgroup
   \dowithnextbox{\makeruledbox\nextbox\box\nextbox\egroup}%
   \normalhbox}
```

first line
second line
third line
fourth line
fifth line

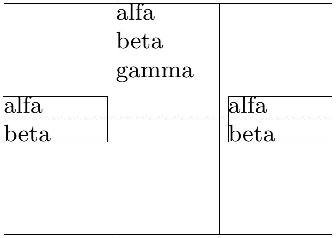
```
\vbox
  {\strut
   first line  \par
   second line \par
   third line  \par
   fourth line \par
   fifth line
   \strut }
```

20
```
\def\ruledvbox%
  {\normalvbox\bgroup
   \dowithnextbox{\makeruledbox\nextbox\box\nextbox\egroup}%
   \normalvbox}
```

first line
second line
third line
fourth line
fifth line

```
\vtop
  {\strut
   first line  \par
   second line \par
   third line  \par
   fourth line \par
   fifth line
   \strut }
```

21
```
\def\ruledvtop%
  {\normalvtop\bgroup
   \dowithnextbox{\makeruledbox\nextbox\box\nextbox\egroup}%
   \normalvtop}
```



```
\hbox
  {$\vcenter{\hsize.2\hsize
      alfa \par beta}$
   $\vcenter to 3cm{\hsize.2\hsize
      alfa \par beta \par gamma}$
   $\vcenter{\hsize.2\hsize
      alfa \par beta}$}
```

22
```
\def\ruledvcenter%
  {\normalvbox\bgroup
   \dontinterfere
   \dowithnextbox
     {\scratchdimen=.5\ht\nextbox
      \advance\scratchdimen by .5\dp\nextbox
      \ht\nextbox=\scratchdimen
      \dp\nextbox=\scratchdimen
      \ruledhbox{\box\nextbox}%
      \egroup}%
   \normalvbox}
```

\ruledbox
\setruledbox

Of the next two macros the first can be used to precede a box of ones own choice. One can for instance prefix boxes with **\ruledbox** and afterwards — when the macro satisfy the needs — let it to **\relax**.

```
\ruledbox\hbox{What rules do you mean?}
```

The macro **\setruledbox** can be used to directly rule a box.

```
\setruledbox12=\hbox{Who's talking about rules here?}
```

At the cost of some extra macros we can implement a variant that does not need the =, but we stick to:

23
```
\def\ruledbox%
  {\dowithnextbox{\makeruledbox\nextbox\box\nextbox}}
```

24
```
\def\setruledbox#1=%
  {\dowithnextbox{\makeruledbox\nextbox\setbox#1=\nextbox}}
```

\investigateskip
\investigatecount
\investigatemuskip

Before we meet the visualizing macro's, we first implement ourselves some handy utility ones. Just for the sake of efficiency and readability, we introduce some status variables, that tell us a bit more about the registers we use:

```
   \ifflexible
   \ifzero
   \ifnegative
   \ifpositive
```

These status variables are set when we call for one of the investigation macros, e.g.

```
   \investigateskip\scratchskip
```

We use some dirty trick to check stretchability of ⟨*skips*⟩. Users of these macros are invited to study their exact behavior first. The positive and negative states both include zero and are in fact non-negative (≥ 0) and non-positive (≤ 0) .

25
```
\newif\ifflexible
\newif\ifzero
\newif\ifnegative
\newif\ifpositive
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

26
```
\def\investigateskip#1%
  {\relax
   \scratchdimen=#1\relax
   \edef\!!stringa{\the\scratchdimen}%
   \edef\!!stringb{\the#1}%
   \ifx\!!stringa\!!stringb \flexiblefalse \else \flexibletrue \fi
   \ifdim#1=\!!zeropoint\relax
     \zerotrue      \else
     \zerofalse     \fi
   \ifdim#1<\!!zeropoint\relax
     \positivefalse \else
     \positivetrue  \fi
   \ifdim#1>\!!zeropoint\relax
     \negativefalse \else
     \negativetrue  \fi}
```

27
```
\def\investigatecount#1%
  {\relax
   \flexiblefalse
   \ifnum#1=0
     \zerotrue      \else
     \zerofalse     \fi
   \ifnum#1<0
     \positivefalse \else
     \positivetrue  \fi
   \ifnum#1>0
     \negativefalse \else
     \negativetrue  \fi}
```

28
```
\def\investigatemuskip#1%
  {\relax
   \edef\!!stringa{\the\scratchmuskip}%
   \edef\!!stringb{0mu}%
   \def\!!stringc##1##2\\{##1}%
   \expandafter\edef\expandafter\!!stringc\expandafter
     {\expandafter\!!stringc\!!stringa\\}%
   \edef\!!stringd{-}%
   \flexiblefalse
   \ifx\!!stringa\!!stringb
     \zerotrue
     \negativefalse
     \positivefalse
   \else
     \zerofalse
     \ifx\!!stringc\!!stringd
       \positivefalse
       \negativetrue
     \else
       \positivetrue
       \negativefalse
     \fi
   \fi}
```

`\dontinterfere` Indentation, left and/or right skips, redefinition of `\par` and assignments to `\everypar` can lead to unwanted results. We can therefore turn all those things off with `\dontinterfere`.

29
```
\def\dontinterfere%
  {\everypar   = {}%
   \let\par    = \endgraf
   \parindent = \!!zeropoint
   \parskip   = \!!zeropoint
   \leftskip  = \!!zeropoint
   \rightskip = \!!zeropoint
   \relax}
```

\dontcomplain

In this module we do a lot of box manipulations. Because we don't want to be confronted with to many over- and underfull messages we introduce `\dontcomplain`.

30

```
\def\dontcomplain%
  {\hbadness   = \!!tenthousand
   \hfuzz      = \maxdimen
   \vbadness   = \!!tenthousand
   \vfuzz      = \maxdimen}
```

Now the neccessary utility macros are defined, we can make a start with the visualizing ones. The implementation of these macros is a compromise between readability, efficiency of coding and processing speed. Sometimes we do in steps what could have been done in combination, sometimes we use a few boxes more or less then actually needed, and more than once one can find the same piece of rule drawing code twice.

\ifcenteredvcue
\normalvcue

Depending on the context, one can force visual vertical cues being centered along `\hsize` or being put at the current position. Although centering often looks better, we've chosen the second alternative as default. The main reason for doing so is that often when we don't set the `\hsize` ourselves, TEX takes the value of the surrounding box. As a result the visual cues can migrate outside the current context.

This behavior is accomplished by a small but effective auxiliary macro, which behavior can be influenced by the boolean `\centeredvcue`. By saying

```
   \centeredvcuetrue
```

supp-mis supp-ver supp-vis supp-lan supp-pdf supp-spe supp-mps supp-tpi supp-fil supp-ini supp-box supp-mrk supp-mul supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

one turns centering on. As said, we turn it off.

31
```
\newif\ifcenteredvcue  \centeredvcuefalse
```

32
```
\def\normalvcue#1%
  {\normalhbox \ifcenteredvcue to \hsize \fi {\normalhss#1\normalhss}}
```

We could have used the more robust version

```
\def\normalvcue%
  {\normalhbox \ifcenteredvcue to \hsize \fi
   \bgroup\bgroup\normalhss
   \aftergroup\normalhss\aftergroup\egroup
   \let\next=}
```

or the probably best one:

```
\def\normalvcue%
  {\hbox \ifcenteredvcue to \hsize
     \bgroup\bgroup\normalhss
     \aftergroup\normalhss\aftergroup\egroup
   \else
     \bgroup
   \fi
   \let\next=}
```

Because we don't have to preserve ⟨*catcodes*⟩ and only use small arguments, we stick to the first alternative.

\testrulewidth

We build our visual cues out of rules. At the cost of a much bigger DVI file, this is to be prefered over using characters (1) because we cannot be sure of their availability and (2) because their dimensions are fixed.

As with ruled boxes, we use a ⟨*dimension*⟩ to specify the width of the ruled elements. This dimension defaults to:

```
\testrulewidth=\boxrulewidth
```

Because we prefer whole numbers for specifying the dimensions, we often use even multiples of `\testrulewidth`.

\visiblestretch

A second variable is introduced because of the stretch components of ⟨*skips*⟩. At the cost of some accuracy we can make this stretch visible.

```
\visiblestretchtrue
```

33

```
\newdimen\testrulewidth  \testrulewidth=\boxrulewidth
\newif\ifvisiblestretch  \visiblestretchfalse
```

\ruledhss
\ruledhfil
\ruledhfilneg
\ruledhfill
\ruledhfillneg

We start with the easiest part, the fills. The scheme we follow is *visual filling – going back – normal filling*. Visualizing is implemented using `\cleaders`. Because the ⟨*box*⟩ that follows this command is constructed only once, the `\copy` is not really a prerequisite. We prefer using a `\normalhbox` here instead of a `\hbox`.

34

```
\def\setvisiblehfilbox#1\to#2#3#4%
  {\setbox#1=\normalhbox
     {\vrule
        \!!width#2\testrulewidth
        \!!height#3\testrulewidth
        \!!depth#4\testrulewidth}%
   \smashbox#1}
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

35
```
\def\doruledhfiller#1#2#3#4%
  {#1#2%
   \bgroup
     \dontinterfere
     \dontcomplain
     \setvisiblehfilbox0\to{4}{#3}{#4}%
     \setvisiblehfilbox2\to422%
     \copy0\copy2
     \bgroup
       \setvisiblehfilbox0\to422%
       \cleaders
         \normalhbox to 12\testrulewidth
           {\normalhss\copy0\normalhss}%
         #1%
     \egroup
     \setbox0=\normalhbox
       {\normalhskip-4\testrulewidth\copy0\copy2}%
     \smashbox0
     \box0
   \egroup}
```

The horizontal fillers differ in their boundary visualization. Watch the small dots. Fillers can be combined within reasonable margins.

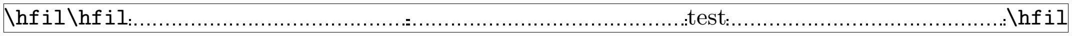
\hss...test

\hfil:...:test

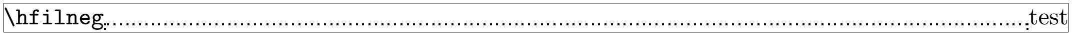
\hfill:...:test

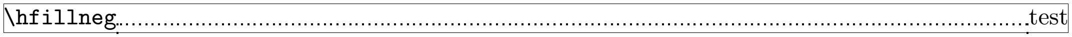
supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

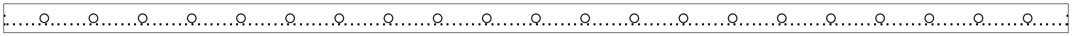


The negative counterparts are visualizes, but seldom become visible, apart from their boundaries.





Although leaders are used for visualizing, they are visualized themselves correctly as the next example shows.



All five substitutions use the same auxiliary macro. Watch the positive first – negative next approach.

```
\def\ruledhss%
  {\doruledhfiller\normalhss\normalhfilneg{0}{0}}

\def\ruledhfil%
  {\doruledhfiller\normalhfil\normalhfilneg{10}{-6}}

\def\ruledhfill%
  {\doruledhfiller\normalhfill\normalhfillneg{18}{-14}}

\def\ruledhfilneg%
  {\doruledhfiller\normalhfilneg\normalhfil{-6}{10}}

\def\ruledhfillneg%
  {\doruledhfiller\normalhfillneg\normalhfill{-14}{18}}
```

\ruledvss
\ruledvfil
\ruledvfilneg
\ruledvfill
\ruledvfillneg

The vertical mode commands adopt the same visualization scheme, but are implemented in a slightly different way.

41
```
\def\setvisiblevfilbox#1\to#2#3#4%
  {\setbox#1=\normalhbox
     {\vrule
        \!!width#2\testrulewidth
        \!!height#3\testrulewidth
        \!!depth#4\testrulewidth}%
   \smashbox#1}%
```

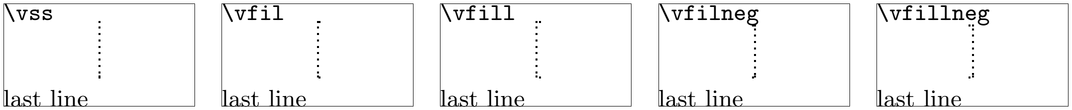
42
```
\def\doruledvfiller#1#2#3%
  {#1#2%
   \bgroup
     \dontinterfere
     \dontcomplain
     \offinterlineskip
     \setvisiblevfilbox0\to422%
     \setbox2=\normalvcue
       {\normalhskip -#3\testrulewidth\copy0}%
     \smashbox2
     \copy2
     \bgroup
       \setbox2=\normalvcue
         {\normalhskip -2\testrulewidth\copy0}%
       \smashbox2
       \copy2
       \cleaders
         \normalvbox to 12\testrulewidth
           {\normalvss\copy2\normalvss}%
```

supp-mis supp-ver supp-vis supp-lan supp-pdf supp-spe supp-mps supp-tpi supp-fil supp-ini supp-box supp-mrk supp-mul supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
        #1%
      \setbox2=\normalvbox
        {\normalvskip-2\testrulewidth\copy2}%
      \smashbox2
      \box2
    \egroup
    \box2
  \egroup}
```

Because they act the same as their horizontal counterparts we only show a few examples.



Keep in mind that **\vfillneg** is not part of PLAIN TEX, but are mimmicked by a macro.

43
```
\def\ruledvss%
  {\doruledvfiller\normalvss\normalvfilneg{2}}
```

44
```
\def\ruledvfil%
  {\doruledvfiller\normalvfil\normalvfilneg{-4}}
```

45
```
\def\ruledvfill%
  {\doruledvfiller\normalvfill\normalvfillneg{-12}}
```

46
```
\def\ruledvfilneg%
  {\doruledvfiller\normalvfilneg\normalvfil{8}}
```

47
```
\def\ruledvfillneg%
  {\doruledvfiller\normalvfillneg\normalvfill{16}}
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

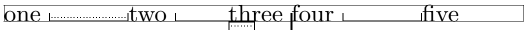
contents register context syst mult supp lang font colo spec core cont m s exit go back

\ruledhskip

Skips differ from kerns in two important aspects:

- line and pagebreaks are allowed at a skip
- skips can have a positive and/or negative stretchcomponent

Stated a bit different: kerns are fixed skips at which no line or pagebreak can occur. Because skips have a more open character, they are visualized in a open way.



```
one
\hskip +30pt plus 5pt
two
\hskip +30pt
\hskip -10pt plus 5pt
three
\hskip   0pt
four
\hskip +30pt
five
```

When skips have a stretch component, this is visualized by means of a dashed line. Positive skips are on top of the baseline, negative ones are below it. This way we can show the combined results. An alternative visualization of stretch could be drawing the mid line over a length of the stretch, in positive or negative direction.

48

```
\def\doruledhskip%
  {\relax
   \dontinterfere
   \dontcomplain
   \investigateskip\scratchskip
   \ifzero
```

```
      \setbox0=\normalhbox
        {\normalhskip-\testrulewidth
         \vrule
           \!!width4\testrulewidth
           \!!height16\testrulewidth
           \!!depth16\testrulewidth}%
    \else
      \setbox0=\normalhbox to \ifnegative-\fi\scratchskip
        {\vrule
           \!!width2\testrulewidth
           \ifnegative\!!depth\else\!!height\fi16\testrulewidth
         \cleaders
           \hrule
             \ifnegative
               \!!depth2\testrulewidth
               \!!height\!!zeropoint
             \else
               \!!height2\testrulewidth
               \!!depth\!!zeropoint
             \fi
           \normalhfill
         \ifflexible
           \normalhskip\ifnegative\else-\fi\scratchskip
           \normalhskip2\testrulewidth
           \cleaders
             \normalhbox
               {\normalhskip 2\testrulewidth
                \vrule
                  \!!width2\testrulewidth
```

```
              \!!height\ifnegative-7\else9\fi\testrulewidth
              \!!depth\ifnegative9\else-7\fi\testrulewidth
            \normalhskip 2\testrulewidth}%
         \normalhfill
     \fi
     \vrule
       \!!width2\testrulewidth
       \ifnegative\!!depth\else\!!height\fi16\testrulewidth}%
  \setbox0=\normalhbox
    {\ifnegative\else\normalhskip-\scratchskip\fi
     \box0}%
\fi
\smashbox0%
\ifvisiblestretch \else
  \flexiblefalse
\fi
\ifflexible
  % breaks ok but small displacements can occur
  \skip2=\scratchskip
  \advance\skip2 by -1\scratchskip
  \divide\skip2 by 2
  \advance\scratchskip by -\skip2
  \normalhskip\scratchskip
  \normalpenalty\!!tenthousand
  \box0
  \normalhskip\skip2
\else
  \normalhskip\scratchskip
  \box0
```

```
    \fi
    \egroup}
```

49
```
\def\ruledhskip%
  {\bgroup
    \afterassignment\doruledhskip
    \scratchskip=}
```

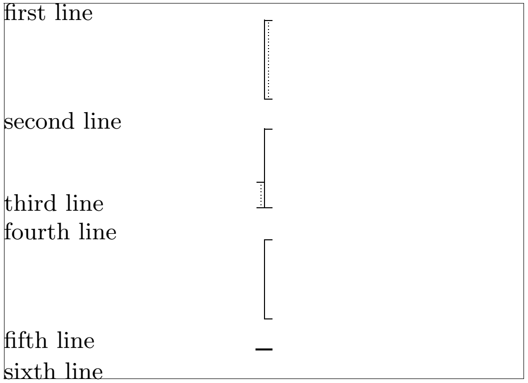
The visual skip is located at a feasible point. Normally this does not interfere with the normaltypesetting process. The next examples show (1) the default behavior, (2) the (not entirely correct) distributed stretch and (3) the way the text is typeset without cues.

test test testtest test testtest test testtest test testtest test testtest test
testtest test testtest test testtest test testtest test testtest test testtest
test testtest test testtest test testtest test test

test test testtest test testtest test testtest test testtest test testtest test
testtest test testtest test testtest test testtest test testtest test testtest
test testtest test testtest test testtest test test

test test testtest test testtest test testtest test testtest test testtest test
testtest test testtest test testtest test testtest test testtest test testtest
test testtest test testtest test testtest test test

`\ruledvskip` We are less fortunate when implementing the vertical skips. This is a direct result of interference between the boxes that visualize the skip and skip removal at a pagebreak. Normally skips disappear at the top of a page, but not of course when visualized in a `\vbox`. A quite perfect simulation could have been built if we would have had available two more primitives: `\hnop` and `\vnop`. These new primitives could stand for boxes that are visible but are not taken into account in any way. They are there for us, but not for TEX.



```
first line
\vskip +30pt plus 5pt
second line
\vskip +30pt
\vskip -10pt plus 5pt
third line
\par
fourth line
\vskip +30pt
fifth line
\vskip  0pt
sixth line
```

We have to postpone `\prevdepth`. Although this precaution probably is not completely waterproof, it works quite well.

50

```
\def\dodoruledvskip%
  {\nextdepth=\prevdepth
   \dontinterfere
   \dontcomplain
   \offinterlineskip
   \investigateskip\scratchskip
   \ifzero
     \setbox0=\normalvcue
       {\vrule
          \!!width32\testrulewidth
          \!!height2\testrulewidth
          \!!depth2\testrulewidth}%
   \else
```

```
\setbox0=\normalvbox to \ifnegative-\fi\scratchskip
  {\hrule
     \!!width16\testrulewidth
     \!!height2\testrulewidth
   \ifflexible
     \cleaders
       \normalhbox to 16\testrulewidth
         {\normalhss
          \normalvbox
            {\normalvskip 2\testrulewidth
             \hrule
             \!!width2\testrulewidth
             \!!height2\testrulewidth
             \normalvskip 2\testrulewidth}%
          \normalhss}%
       \normalvfill
   \else
     \normalvfill
   \fi
   \hrule
     \!!width16\testrulewidth
     \!!height2\testrulewidth}%
\setbox2=\normalvbox to \ht0
  {\hrule
     \!!width2\testrulewidth
     \!!height\ht0}%
\ifnegative
  \ht0=\!!zeropoint
  \setbox0=\normalhbox
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
          {\normalhskip2\testrulewidth % will be improved
           \normalhskip-\wd0\box0}%
     \fi
     \smashbox0%
     \smashbox2%
     \setbox0=\normalvcue
       {\box2\box0}%
     \setbox0=\normalvbox
       {\ifnegative\normalvskip\scratchskip\fi\box0}%
     \smashbox0%
   \fi
   \ifvisiblestretch
     \ifflexible
       \skip2=\scratchskip
       \advance\skip2 by -1\scratchskip
       \divide\skip2 by 2
       \advance\scratchskip by -\skip2
       \normalvskip\skip2
     \fi
   \fi
   \normalpenalty\!!tenthousand
   \box0
   \prevdepth=\nextdepth % not \dp0=\nextdepth
   \normalvskip\scratchskip}
```

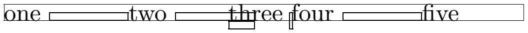
We try to avoid interfering at the top of a page. Of course we only do so when we are in the main vertical list.

51
```
\def\doruledvskip%
  {\endgraf % \par
```

```
    \ifdim\pagegoal=\maxdimen
      \ifinner
        \dodoruledvskip
      \fi
    \else
      \dodoruledvskip
    \fi
    \egroup}
```

52
```
\def\ruledvskip%
  {\bgroup
    \afterassignment\doruledvskip
    \scratchskip=}
```

\ruledkern The macros that implement the kerns are a bit more complicated than needed, because they also serve the visualization of glue, our PLAIN defined kerns with stretch or shrink. We've implemented both horizontal and vertical kerns as ruled boxes.



```
one
\kern +30pt
two
\kern +30pt
\kern -10pt
three
\kern   0pt
four
\kern +30pt
five
```

Positive and negative kerns are placed on top or below the baseline, so we are able to track their added result. We didn't mention spacings of 0 pt yet. Zero values are visualized a bit different, because we want to see them anyhow.

53
```
\def\doruledhkern%
  {\dontinterfere
   \dontcomplain
   \baselinerulefalse
   \investigateskip\scratchskip
   \boxrulewidth=2\testrulewidth
   \ifzero
     \setbox0=\ruledhbox to 8\testrulewidth
       {\vrule
          \!!width\!!zeropoint
          \!!height16\testrulewidth
          \!!depth16\testrulewidth}%
     \setbox0=\normalhbox
       {\normalhskip-4\testrulewidth\box0}%
   \else
     \setbox0=\ruledhbox to \ifnegative-\fi\scratchskip
       {\vrule
          \!!width\!!zeropoint
          \ifnegative\!!depth\else\!!height\fi16\testrulewidth
        \ifflexible
          \normalhskip2\testrulewidth
          \cleaders
            \normalhbox
              {\normalhskip 2\testrulewidth
               \vrule
                 \!!width2\testrulewidth
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

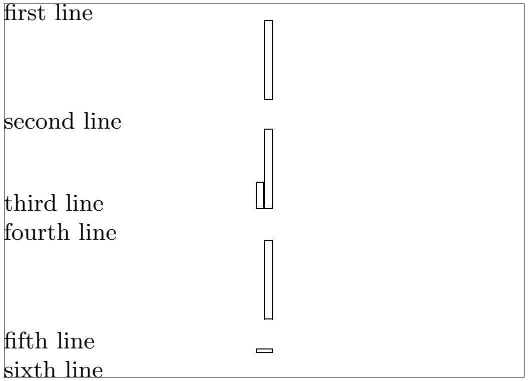
```
                  \!!height\ifnegative-7\else9\fi\testrulewidth
                  \!!depth\ifnegative9\else-7\fi\testrulewidth
                \normalhskip 2\testrulewidth}%
             \normalhfill
         \else
           \normalhfill
         \fi}%
      \testrulewidth=2\testrulewidth
      \setbox0=\ruledhbox{\box0}%  \make...
    \fi
    \smashbox0%
    \normalpenalty\!!tenthousand
    \normalhbox to \!!zeropoint
      {\ifnegative\normalhskip1\scratchskip\fi
       \box0}%
    \afterwards\scratchskip
    \egroup}
```

54
```
\def\ruledhkern#1%
  {\bgroup
   \let\afterwards=#1\relax
   \afterassignment\doruledhkern
   \scratchskip=}
```

After having seen the horizontal ones, the vertical kerns will not surprise us. In this example we use `\par` to switch to vertical mode.

contents register context syst mult supp lang font colo spec core cont m s exit go back

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun



```
first line
\par \kern +30pt
second line
\par \kern +30pt
\par \kern -10pt
third line
\par
fourth line
\par \kern +30pt
fifth line
\par \kern   0pt
sixth line
```

Like before, we have to postpone `\prevdepth`. If we leave out this trick, we got ourselves some wrong spacing.

```
\def\dodoruledvkern%
  {\nextdepth=\prevdepth
   \dontinterfere
   \dontcomplain
   \baselinerulefalse
   \offinterlineskip
   \investigateskip\scratchskip
   \boxrulewidth=2\testrulewidth
   \ifzero
     \setbox0=\ruledhbox to 32\testrulewidth
       {\vrule
          \!!width\!!zeropoint
          \!!height4\testrulewidth
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
       \!!depth4\testrulewidth}%
\else
  \setbox0=\ruledvbox to \ifnegative-\fi\scratchskip
    {\hsize16\testrulewidth
     \ifflexible
       \cleaders
         \normalhbox to 16\testrulewidth
           {\normalhss
            \normalvbox
              {\normalvskip 2\testrulewidth
               \hrule
                 \!!width2\testrulewidth
                 \!!height2\testrulewidth
               \normalvskip 2\testrulewidth}%
            \normalhss}%
         \normalvfill
     \else
       \vrule
         \!!width\!!zeropoint
         \!!height\ifnegative-\fi\scratchskip
       \normalhfill
     \fi}
\fi
\testrulewidth=2\testrulewidth
\setbox0=\ruledvbox{\box0}%  \make...
\smashbox0%
\setbox0=\normalvbox
  {\ifnegative\normalvskip\scratchskip\fi
   \normalvcue
```

```
      {\ifnegative\normalhskip-16\testrulewidth\fi\box0}}%
   \smashbox0%
   \normalpenalty\!!tenthousand
   \box0
   \prevdepth=\nextdepth} % not \dp0=\nextdepth
```

56
```
\def\doruledvkern%
  {\ifdim\pagegoal=\maxdimen
     \ifinner
       \dodoruledvkern
     \fi
   \else
     \dodoruledvkern
   \fi
   \afterwards\scratchskip
   \egroup}
```

57
```
\def\ruledvkern#1%
  {\bgroup
   \let\afterwards=#1\relax
   \afterassignment\doruledvkern
   \scratchskip=}
```

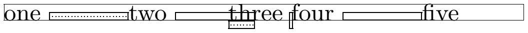
58
```
\def\ruledkern%
  {\ifvmode
     \let\next=\ruledvkern
   \else
     \let\next=\ruledhkern
   \fi
   \next\normalkern}
```

A a bit more TEXnic solution is:

```
\def\ruledkern%
  {\csname ruled\ifvmode v\else h\fi kern\endcsname\normalkern}
```

\ruledhglue
\ruledvglue

The non-primitive glue commands are treated as kerns with stretch. This stretch is presented as a dashed line. I have to admit that until now, I've never used these glue commands.

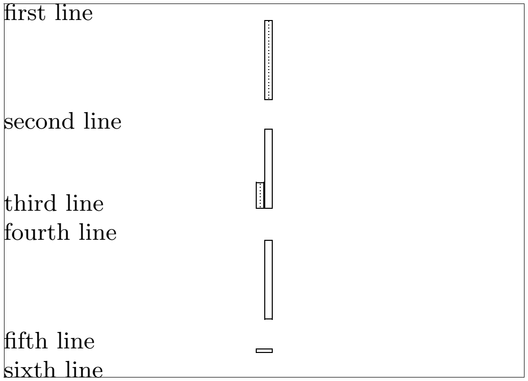


```
one
\hglue +30pt plus 5pt
two
\hglue +30pt
\hglue -10pt plus 5pt
three
\hglue   0pt
four
\hglue +30pt
five
```

59
```
\def\doruledhglue%
  {\leavevmode
   \scratchcounter=\spacefactor
   \vrule\!!width\!!zeropoint
   \normalpenalty\!!tenthousand
   \ruledhkern\normalhskip\scratchskip
   \spacefactor=\scratchcounter
   \egroup}
```

60
```
\def\ruledhglue%
  {\bgroup
```

```
    \afterassignment\doruledhglue\scratchskip=}
```



```
first line
\vglue +30pt plus 5pt
second line
\vglue +30pt
\vglue -10pt plus 5pt
third line
\par
fourth line
\vglue +30pt
fifth line
\vglue  0pt
sixth line
```

61
```
\def\doruledvglue%
  {\endgraf % \par
   \nextdepth=\prevdepth
   \hrule\!!height\!!zeropoint
   \normalpenalty\!!tenthousand
   \ruledvkern\normalvskip\scratchskip
   \prevdepth=\nextdepth
   \egroup}
```

62
```
\def\ruledvglue%
  {\bgroup
   \afterassignment\doruledvglue\scratchskip=}
```

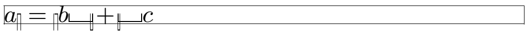
\ruledmkern
\ruledmskip

Mathematical kerns and skips are specified in mu. This font related unit is incompatible with those of ⟨*dimensions*⟩ and ⟨*skips*⟩. Because in math mode spacing is often a very subtle matter, we've used a very simple, not overloaded way to show them.

63

```
\def\dodoruledmkern#1%
  {\dontinterfere
   \dontcomplain
   \setbox0=\normalhbox
     {$\normalmkern\ifnegative-\fi\scratchmuskip$}%
   \setbox0=\normalhbox to \wd0
     {\vrule
        \!!height16\testrulewidth
        \!!depth16\testrulewidth
        \!!width\testrulewidth
      \leaders
        \hrule
          \!!height\ifpositive16\else-14\fi\testrulewidth
          \!!depth\ifpositive-14\else16\fi\testrulewidth
        \normalhfill
      \ifflexible
        \normalhskip-\wd0
        \leaders
          \hrule
            \!!height\testrulewidth
            \!!depth\testrulewidth
          \normalhfill
      \fi
      \vrule
        \!!height16\testrulewidth
        \!!depth16\testrulewidth
```

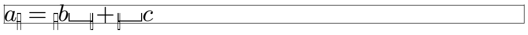
```
       \!!width\testrulewidth}%
  \smashbox0%
  \ifnegative
     #1\scratchmuskip
    \box0
  \else
    \box0
     #1\scratchmuskip
  \fi
  \egroup}
```



```
$a \mkern3mu = \mkern3mu
 b \quad
 \mkern-2mu + \mkern-2mu
 \quad c$
```

64
```
\def\doruledmkern%
  {\investigatemuskip\scratchmuskip
   \flexiblefalse
   \dodoruledmkern\normalmkern}
```

65
```
\def\ruledmkern%
  {\bgroup
   \afterassignment\doruledmkern\scratchmuskip=}
```

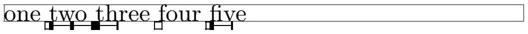


```
$a \mskip3mu = \mskip3mu
 b \quad
 \mskip-2mu + \mskip-2mu
 \quad c$
```

66
```
\def\doruledmskip%
  {\investigatemuskip\scratchmuskip
   \flexibletrue
   \dodoruledmkern\normalmskip}
```

67
```
\def\ruledmskip%
  {\bgroup
   \afterassignment\doruledmskip\scratchmuskip=}
```

`\penalty` After presenting fills, skip, kerns and glue we've come to see penalties. In the first implementation — most of the time needed to develop this set of macros went into testing different types of visualization — penalties were mere small blocks with one black half, depending on the sign. This most recent version also gives an indication of the amount of penalty. Penalties can go from less than −10000 to over +10000, and their behavior is somewhat non-lineair, with some values having special meanings. We therefore decided not to use its value for a lineair indicator.



```
one
\penalty +100
two
\penalty +100
\penalty -100
three
\penalty    0
four
\penalty +100
five
```

The small sticks at the side of the penalty indicate it size. The next example shows the positive and negative penalties of 0, 1, 10, 100, 1000 and 10000.

test test test test test test test

test test test test test test test

This way stacked penalties of different severance can be shown in combination.

test test test test

```
\def\setruledpenaltybox#1#2#3#4#5#6%
  {\setbox#1=\normalhbox
     {\ifnum#2=0 \else
        \ifnum#2>0
          \def\sign{+}%
        \else
          \def\sign{-}%
        \fi
        \dimen0=\ifnum\sign#2>9999
                  28\else
                \ifnum\sign#2>999
                  22\else
                \ifnum\sign#2>99
                  16\else
                \ifnum\sign#2>9
                  10\else
                  4
                \fi\fi\fi\fi \testrulewidth
        \ifnum#2<0
          \normalhskip-\dimen0
          \normalhskip-2\testrulewidth
          \vrule
```

```
            \!!width2\testrulewidth
            \!!height#3\testrulewidth
            \!!depth#4\testrulewidth
        \fi
        \vrule
          \!!width\dimen0
          \!!height#5\testrulewidth
          \!!depth#6\testrulewidth
        \ifnum#2>0
          \vrule
            \!!width2\testrulewidth
            \!!height#3\testrulewidth
            \!!depth#4\testrulewidth
         \fi
      \fi}%
    \smashbox#1}
```

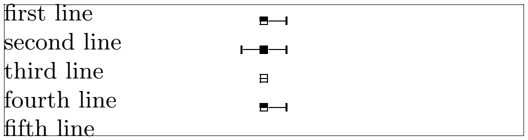
69
```
\def\doruledhpenalty%
  {\dontinterfere
   \dontcomplain
   \investigatecount\scratchcounter
   \testrulewidth=2\testrulewidth
   \boxrulewidth=\testrulewidth
   \setbox0=\ruledhbox to 8\testrulewidth
     {\ifnegative\else\normalhss\fi
      \vrule
        \!!depth8\testrulewidth
        \!!width\ifzero0\else4\fi\testrulewidth
      \ifpositive\else\normalhss\fi}%
   \setruledpenaltybox{2}{\scratchcounter}{0}{8}{-3.5}{4.5}%
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

```
  \normalpenalty\!!tenthousand
  \setbox0=\normalhbox
    {\normalhskip-4\testrulewidth
     \ifnegative
       \box2\box0
     \else
       \box0\box2
     \fi}%
  \smashbox0%
  \box0
  \normalpenalty\scratchcounter
  \egroup}
```

70
```
\def\ruledhpenalty%
  {\bgroup
   \afterassignment\doruledhpenalty
   \scratchcounter=}
```

The size of a vertical penalty is also shown on the horizontal axis. This way there is less interference with the often preceding or following skips and kerns.



```
first line
\par \penalty +100
second line
\par \penalty +100
\par \penalty -100
third line
\par \penalty    0
fourth line
\par \penalty +100
fifth line
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

71

```
\def\doruledvpenalty%
  {\ifdim\pagegoal=\maxdimen
   \else
     \nextdepth=\prevdepth
     \dontinterfere
     \dontcomplain
     \investigatecount\scratchcounter
     \testrulewidth=2\testrulewidth
     \boxrulewidth=\testrulewidth
     \setbox0=\ruledhbox
       {\vrule
          \!!height4\testrulewidth
          \!!depth4\testrulewidth
          \!!width\!!zeropoint
        \vrule
          \!!height\ifnegative.5\else4\fi\testrulewidth
          \!!depth\ifpositive.5\else4\fi\testrulewidth
          \!!width8\testrulewidth}%
     \setruledpenaltybox{2}{\scratchcounter}{4}{4}{.5}{.5}%
     \setbox0=\normalhbox
       {\normalhskip-4\testrulewidth
        \ifnegative
          \box2\box0
        \else
          \box0\box2
        \fi
        \normalhss}%
     \smashbox0%
     \normalpenalty\!!tenthousand
```

```
      \nointerlineskip
      \dp0=\nextdepth   % not \prevdepth=\nextdepth
      \normalvbox
        {\normalvcue{\box0}}%
    \fi
    \normalpenalty\scratchcounter
    \egroup}
```

72
```
\def\ruledvpenalty%
  {\bgroup
    \afterassignment\doruledvpenalty
    \scratchcounter=}
```

73
```
\def\ruledpenalty%
  {\ifvmode
      \let\next=\ruledvpenalty
    \else
      \let\next=\ruledhpenalty
    \fi
    \next}
```

At the cost of some more tokens, a bit more clever implementation would be:

```
    \def\ruledpenalty%
      {\csname ruled\ifvmode v\else h\fi penalty\endcsname}
```

\showfils
\dontshowfils
\showboxes
\dontshowboxes
\showskips
\dontshowskips
\showpenalties
\dontshowpenalties

For those who want to manipulate the visual cues in detail, we have grouped them.

74
```
\def\showfils%
  {\let\hss       = \ruledhss
   \let\hfil      = \ruledhfil
   \let\hfill     = \ruledhfill
   \let\hfilneg   = \ruledhfilneg
   \let\hfillneg  = \ruledhfillneg
   \let\vss       = \ruledvss
   \let\vfil      = \ruledvfil
   \let\vfill     = \ruledvfill
   \let\vfilneg   = \ruledvfilneg
   \let\vfillneg  = \ruledvfillneg}
```

75
```
\def\dontshowfils%
  {\let\hss       = \normalhss
   \let\hfil      = \normalhfil
   \let\hfill     = \normalhfill
   \let\hfilneg   = \normalhfilneg
   \let\hfillneg  = \normalhfillneg
   \let\vss       = \normalvss
   \let\vfil      = \normalvfil
   \let\vfill     = \normalvfill
   \let\vfilneg   = \normalvfilneg
   \let\vfillneg  = \normalvfillneg}
```

76
```
\def\showboxes%
  {\baselineruletrue
   \let\hbox      = \ruledhbox
   \let\vbox      = \ruledvbox
   \let\vtop      = \ruledvtop
   \let\vcenter   = \ruledvcenter}
```

77
```
\def\dontshowboxes%
  {\let\hbox      = \normalhbox
   \let\vbox      = \normalvbox
   \let\vtop      = \normalvtop
   \let\vcenter   = \normalvcenter}
```

78
```
\def\showskips%
  {\let\hskip     = \ruledhskip
   \let\vskip     = \ruledvskip
   \let\kern      = \ruledkern
   \let\mskip     = \ruledmskip
   \let\mkern     = \ruledmkern
   \let\hglue     = \ruledhglue
   \let\vglue     = \ruledvglue}
```

79
```
\def\dontshowskips%
  {\let\hskip     = \normalhskip
   \let\vskip     = \normalvskip
   \let\kern      = \normalkern
   \let\mskip     = \normalmskip
   \let\mkern     = \normalmkern
   \let\hglue     = \normalhglue
   \let\vglue     = \normalvglue}
```

80
```
\def\showpenalties%
  {\let\penalty   = \ruledpenalty}
```

81
```
\def\dontshowpenalties%
  {\let\penalty   = \normalpenalty}
```

\showingcomposition
\showcomposition
\dontshowcomposition
\

All these nice options come together in two macros. The first one turns the options on, the second turnes them off. Both macros only do their job when we are actually showing the composition.

```
\showingcompositiontrue
\showcomposition
```

Because the output routine can do tricky things, like multiple column typesetting and manipulation of the pagebody, shifting things around and so on, the macro `\dontshowcomposition` best can be called when we enter this routine. Too much visual cues just don't make sense. In CONTEXT this has been taken care of.

82
```
\newif\ifshowingcomposition
```

83
```
\def\showcomposition%
  {\ifshowingcomposition
     \showfils
     \showboxes
     \showskips
     \showpenalties
   \fi}
```

84
```
\def\dontshowcomposition%
  {\ifshowingcomposition
     \dontshowfils
     \dontshowboxes
     \dontshowskips
     \dontshowpenalties
   \fi}
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

\showmakeup
\defaulttestrulewidth

Just to make things even more easy, we have defined:

```
\showmakeup
```

For the sake of those who don't (yet) use CONTEXT we preset `\defaulttestrulewidth` to the already set value. Otherwise we default to a corps related value.

```
\def\defaulttestrulewidth{.2pt}
```

Beware, it's a macro not a ⟨*dimension*⟩.

85
```
\ifx\korpsgrootte\undefined
  \edef\defaulttestrulewidth{\the\testrulewidth}
\else
  \def\defaulttestrulewidth{.02\korpsgrootte}  % still dutch
\fi
```

86
```
\def\showmakeup%
  {\testrulewidth=\defaulttestrulewidth
   \showingcompositiontrue
   \showcomposition}
```

87
```
\protect
```

Lets end with some more advanced examples. Definitions and enumerations come in many flavors. The next one for instance is defined as:

```
\definedescription[test][place=left,hang=3,width=6em]
```

When applied to some text, this would look like:

visual debugger I would be very pleased if TEX had two more primitives: `\vnop` and `\hnop`. Both should act and show up as normal boxes, but stay invisible for TEX when it's doing calculations. The \vnop for instance should not interact with the internal mechanism responsible for the disappearing skips, kerns and penalties at a pagebreak. As long as we don't have these two boxtypes, visual debugging will never be perfect.

The index to this section looks like:

```
\normalvfilneg          \ruledvfill
\normalvglue            \ruledvfillneg
\normalvskip            \ruledvfilneg
\normalvss              \ruledvglue
\normalvtop             \ruledvskip
                        \ruledvss
\penalty                \ruledvtop

\ruledbox               \setruledbox
\ruledhbox              \showboxes
\ruledhfil              \showcomposition
\ruledhfill             \showfils
\ruledhfillneg          \showingcomposition
\ruledhfilneg           \showmakeup
\ruledhglue             \showpenalties
\ruledhskip             \showskips
\ruledhss
\ruledkern              \testrulewidth
\ruledmkern
\ruledmskip             \vfilneg
\ruledvbox              \visiblestretch
\ruledvcenter
\ruledvfil
```

Although not impressive examples or typesetting, both show us how and where things happen. When somehow the last lines in this two column index don't allign, then this is due to some still unknown interference.

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

4.4 Language Options

One of TEX's strong points in building paragraphs is the way hyphenations are handled. Although for real good hyphenation of non–english languages some extensions to the program are needed, fairly good results can be reached with the standard mechanisms and an additional macro, at least in Dutch.

1
```
\unprotect
```

CONTEXT originates in the wish to typeset educational materials, especially in a technical environment. In production oriented environments, a lot of compound words are used. Because the Dutch language poses no limits on combining words, we often favor putting dashes between those words, because it facilitates reading, at least for those who are not that accustomed to it.

In TEX compound words, separated by a hyphen, are not hyphenated at all. In spite of the multiple pass paragraph typesetting this can lead to parts of words sticking into the margin. The solution lays in saying `spoelwater||terugwinunit` instead of `spoelwater-terugwinunit`. By using a one character command like `|`, delimited by the same character `|`, we get ourselves both a decent visualization (in TEXEDIT and colored verbatim we color these commands yellow) and an efficient way of combining words.

The sequence `||` simply leads to two words connected by a hyphen. Because we want to distinguish such a hyphen from the one inserted when TEX hyphenates a word, we use a bit longer one.

`spoelwater||terugwinunit`	spoel-wa-ter–te-rug-win-unit	spoelwater–terugwinunit

As we already said, the `|` is a command. This commands accepts an optional argument before it's delimiter, which is also a `|`.

`polymeer|*|chemie`	po-ly-meer*che-mie	polymeer*chemie

Arguments like `*` are not interpreted and inserted directly, in contrary to arguments like:

`polymeer|~|chemie`	po-ly-meer-che-mie	polymeer chemie
`|(|polymeer|)|chemie`	(po-ly-meer-)che-mie	(polymeer)chemie
`polymeer|(|chemie|)|`	po-ly-meer(-che-mie)	polymeer(chemie)

Although such situations seldom occur —we typeset thousands of pages before we encountered one that forced us to enhance this mechanism— we also have to take care of comma's.

`op||, in|| en uitstellen`	op–, in– enuit-stel-len	op–, in– en uitstellen

The next special case (concerning quotes) was brought to my attention by Piet Tutelaers, one of the driving forces behind rebuilding hyphenation patterns for the dutch language.[3] We'll also take care of this case.

`AOW|'|er`	AOW-er	AOW'er
`cd|'|tje`	cd-tje	cd'tje
`ex|-|PTT|'|er`	ex-PTT-er	ex-PTT'er
`rock|-|'n|-|roller`	rock-'n-roller	rock-'n-roller

Tobias Burnus pointed out that I should also support something like

`well|_|known`	well-–known	wellknown

to strees the compoundness of hyphenated words.

Of course we also have to take care of the special case:

`text||color and ||font`	text–col-orand–font	text–color and –font

[3] In 1996 the spelling of the dutch language has been slightly reformed which made this topic actual again.

\installdiscretionaries

The mechanism described here is one of the older inner parts of CONTEXT. The most recent extensions concerns some special cases as well as the possibility to install other characters as delimiters. The prefered way of specifying compound words is using `||`, which is installed by:

```
\installdiscretionaries || -
```

Some alternative definitions are:

```
\installdiscretionaries ** -
\installdiscretionaries ++ -
\installdiscretionaries // -
\installdiscretionaries ~~ -
```

after which we can say:

`test**test**test`	test-test-test	test-test-test
`test++test++test`	test-test-test	test-test-test
`test//test//test`	test-test-test	test-test-test
`test~~test~~test`	test-test-test	test-test-test

\compoundhyphen
\beginofsubsentence
\endofsubsentence

Now let's go to the macros. First we define some variables. In the main CONTEXT modules these can be tuned by a setup command. Watch the (maybe) better looking compound hyphen.

2
```
\def\compoundhyphen     {{-}\kern-.25ex{-}}
\def\beginofsubsentence {---}
\def\endofsubsentence   {---}
```

The last two variables are needed for subsentences —like this one— which we did not yet mention.

We want to enable breaking but at the same time don't want compound characters like - or – to be separated from the words. TEX hackers will recognise the next two macro's:

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
3 \def\prewordbreak  {\penalty10000\hskip0pt\relax}
\def\postwordbreak {\penalty0\prewordbreak}
```

We first show the original implementation, which only supports `|` as command and delimiter. Before activating `|` we save it's value:

```
   \edef\domathmodediscretionary{\string|}
```

after which we're ready to define it's meaning to:

```
   \catcode`\|=\@@active

   \unexpanded\def|%
     {\ifmmode
        \expandafter\domathmodediscretionary
      \else
        \expandafter\dotextmodediscretionary
      \fi}
```

We need a two stage **`\futurelet`** because we want to look ahead for both the compound character definition and the (optional) comma that follows it, and because we want to prevent that TEX puts this comma on the next line. We use **`\next`** for easy and fast checking of the argument, we save this argument (which can consist of more tokens) and also save the character following the `|#1|` in **`\nextnext`**.

```
   \def\dotextmodediscretionary%
     {\bgroup
      \futurelet\next\dodotextmodediscretionary}

   \def\dodotextmodediscretionary#1|%
     {\def\betweendiscretionaries{#1}%
```

```
    \futurelet\nextnext\dododotextmodediscretionary}
```

The main macro consists of quite some `\ifx` tests while `\checkafterdiscretionary` handles the commas. We show the simplified version here:

```
\def\dododotextmodediscretionary%
  {\let\nextnextnext=\egroup
   \ifx     |\next
     \checkafterdiscretionary
     \prewordbreak\hbox{\compoundhyphen\nextnext}\postwordbreak
   \else\ifx=\next
     \prewordbreak\compoundhyphen
   \else\ifx~\next
     \discretionary{-}{}{\thinspace}\postwordbreak
   \else\ifx(\next
     \prewordbreak\discretionary{}{(-}{(}\prewordbreak
   \else\ifx)\next
     \prewordbreak\discretionary{-)}{}{)}\prewordbreak
   \else\ifx'\next
     \prewordbreak\discretionary{-}{}{'}\postwordbreak
   \else
     \checkafterdiscretionary
     \prewordbreak\hbox{\betweendiscretionaries\nextnext}\postwordbreak
   \fi\fi\fi\fi\fi\fi
   \nextnextnext}

\def\checkafterdiscretionary%
  {\ifx,\nextnext
     \def\nextnextnext{\afterassignment\egroup\let\next=}%
   \else
```

```
    \let\nextnext=\relax
  \fi}
```

Handling (and) is a a bit special, because TEX sees them as decent hyphenation points, according to their `\lccode` being non–zero. For the same reason, later on in this module we cannot manipulate the `\lccode` but take the `\uccode`.

The most recent implementation is more advanced. As demonstrated we can install delimiters, like:

```
\installdiscretionaries || \compoundhyphen
```

This time we have to use a bit more clever way of saving the math mode specification of the character we're going to make active. We also save the user supplied compound hyphen. We show the a bit more traditional implementation first.

```
\def\installdiscretionaries#1%
  {\catcode`#1\@@other
   \expandafter\doinstalldiscretionaries\string#1}

\def\doinstalldiscretionaries#1%
  {\setvalue{mathmodediscretionary#1}{#1}%
   \catcode`#1\@@active
   \dodoinstalldiscretionaries}

\def\dodoinstalldiscretionaries#1#2%
  {\setvalue{textmodediscretionary\string#1}{#2}%
   \unexpanded\def#1{\discretionarycommand#1}}
```

A bit more ⟨*catcode*⟩ and character trickery enables us to discard the two intermediate steps. This trick originates on page 394 of the TEXbook, in the appendix full of dirty tricks. The second argument

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

has now become redundant, but I decided to reserve it for future use. At least it remembers us of the symmetry.

4
```
\def\installdiscretionaries#1#2#3%
  {\setvalue{mathmodediscretionary\string#1}{\char`#1}%
   \setvalue{textmodediscretionary\string#1}{#3}%
   \catcode`#1=\@@active
   \scratchcounter=\the\uccode`~
   \uccode`~=`#1
   \uppercase{\unexpanded\def~{\discretionarycommand~}}%
   \uccode`~=\scratchcounter}
```

5
```
\def\dohandlemathmodebar#1%
  {\getvalue{mathmodediscretionary\string#1}}
```

6
```
\def\discretionarycommand%
  {\ifmmode
     \expandafter\dohandlemathmodebar
   \else
     \expandafter\dotextmodediscretionary
   \fi}
```

Although adapting character codes and making characters active can interfere with other features of macropackages, normally there should be no problems with things like:

```
   \installdiscretionary || +
   \installdiscretionary ++ =
```

The real work is done by the next set of macros. We have to use a double `\futurelet` because we have to take following characters into account.

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

7
```
\def\dotextmodediscretionary#1%
  {\bgroup
   \def\dodotextmodediscretionary##1#1%
     {\def\betweendiscretionary{##1}%
      \futurelet\nextnext\dododotextmodediscretionary}%
   \let\discretionarycommand=#1%
   \def\textmodediscretionary{\getvalue{textmodediscretionary\string#1}}%
   \futurelet\next\dodotextmodediscretionary}
```

8
```
\def\dododotextmodediscretionary%
  {\let\nextnextnext=\egroup
   \ifx\discretionarycommand\next
     \checkafterdiscretionary
     \bgroup
       \checkbeforediscretionary
       \prewordbreak\hbox{\textmodediscretionary\nextnext}\postwordbreak
     \egroup
   \else\ifx=\next
     \prewordbreak\textmodediscretionary
   \else\ifx~\next
     \prewordbreak\discretionary{-}{}{\thinspace}\postwordbreak
   \else\ifx_\next
     \prewordbreak\discretionary
       {\textmodediscretionary}{\textmodediscretionary}{}\prewordbreak
   \else\ifx(\next
     \ifdim\lastskip>\!!zeropoint\relax
       (\prewordbreak
     \else
       \prewordbreak\discretionary{}{(-}{(}\prewordbreak
     \fi
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
  \else\ifx)\next
    \ifx\nextnext\blankspace
      \prewordbreak)\relax
    \else
      \prewordbreak\discretionary{-)}{}{)}\prewordbreak
    \fi
  \else\ifx'\next
    \prewordbreak\discretionary{-}{}{'}\postwordbreak
  \else\ifx<\next
    \beginofsubsentence\prewordbreak\beginofsubsentencespacing
  \else\ifnum\uccode`>=\nextuccode
    \endofsubsentencespacing\prewordbreak\endofsubsentence
  \else
    \checkafterdiscretionary
    \bgroup
      \checkbeforediscretionary
      \prewordbreak\hbox{\betweendiscretionary\nextnext}\postwordbreak
    \egroup
  \fi\fi\fi\fi\fi\fi\fi\fi\fi
  \nextnextnext}
```

9
```
\def\checkbeforediscretionary%
  {\setbox0=\lastbox
   \ifdim\wd0=\!!zeropoint
     \let\postwordbreak=\prewordbreak
   \fi
   \box0\relax}
```

10
```
\def\checkafterdiscretionary%
  {\ifx,\nextnext
```

supp-mis supp-ver supp-vis supp-lan supp-pdf supp-spe supp-mps supp-tpi supp-fil supp-ini supp-box supp-mrk supp-mul supp-fun
contents register context syst mult supp lang font colo spec core cont m s exit go back

```
    \def\nextnextnext{\afterassignment\egroup\let\next=}%
  \else
    \let\nextnext=\relax
  \fi}
```

The macro `\checkbeforediscretionary` takes care of loners like `||word`, while it counterpart `\checkafterdiscretionary` is responsible for handling the comma.

`\beginofsubsentencespa..`
`\endofsubsentencespacing`

In the previous macros we provided two hooks which can be used to support nested sub–sentences. In CONTEXT these hooks are used to insert a small space when needed.

11
```
\let\beginofsubsentencespacing=\relax
\let\endofsubsentencespacing  =\relax
```

Before we show some more tricky alternative, we first install the mechanism:

12
```
\installdiscretionaries || \compoundhyphen
```

One of the drawbacks of this mechanism is that characters can be made active afterwards. The next alternative can be used in such situations. This time we don't compare the arguments directly but use the `\uccode`'s instead. TEX initializes these codes of the alphabetics glyphs to their uppercase counterparts. Normally the other characters remain zero. If so, we can use the `\uccode` as a signal.

`\enableactivediscretio..`

The more advanced mechanism is activated by calling:

```
  \enableactivediscretionaries
```

which is defined as:

13
```
\def\enableactivediscretionaries%
  {\uccode`'=`'\relax \uccode`~=`~\relax \uccode`_=`_\relax
   \uccode`(=`(\relax \uccode`)=`)\relax \uccode`==`=\relax
```

supp-mis supp-ver supp-vis supp-lan supp-pdf supp-spe supp-mps supp-tpi supp-fil supp-ini supp-box supp-mrk supp-mul supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
   \uccode`<=`<\relax \uccode`>=`>\relax
   \let\dotextmodediscretionary     = \activedotextmodediscretionary
   \let\dododotextmodediscretionary = \activedododotextmodediscretionary}
```

We only have to redefine two macros. While saving the `\uccode` in a macro we have to take care of empty arguments, like in `||`.

14
```
\def\activedotextmodediscretionary#1%
  {\bgroup
   \def\dodotextmodediscretionary##1#1%
     {\def\betweendiscretionary{##1}%
      \def\nextuccode####1####2\relax%
        {\ifcat\noexpand####1\noexpand\relax
           \edef\nextuccode{0}%
         \else
           \edef\nextuccode{\the\uccode`####1}%
         \fi}%
      \nextuccode##1@\relax
      \futurelet\nextnext\dododotextmodediscretionary}%
   \let\discretionarycommand=#1%
   \def\textmodediscretionary{\getvalue{textmodediscretionary\string#1}}%
   \futurelet\next\dodotextmodediscretionary}
```

This time we use `\ifnum`:

15
```
\def\activedododotextmodediscretionary%
  {\let\nextnextnext=\egroup
   \ifx\discretionarycommand\next
     \checkafterdiscretionary
     \bgroup
       \checkbeforediscretionary
```

```
      \prewordbreak\hbox{\textmodediscretionary\nextnext}\postwordbreak
    \egroup
  \else\ifnum\uccode`==\nextuccode
    \prewordbreak\textmodediscretionary
  \else\ifnum\uccode`~=\nextuccode
    \prewordbreak\discretionary{-}{}{\thinspace}\postwordbreak
  \else\ifnum\uccode`_=\nextuccode
    \prewordbreak\discretionary
      {\textmodediscretionary}{\textmodediscretionary}{}\prewordbreak
  \else\ifnum\uccode`(=\nextuccode
    \ifdim\lastskip>\!!zeropoint\relax
      (\prewordbreak
    \else
      \prewordbreak\discretionary{}{(-}{(}\prewordbreak
    \fi
  \else\ifnum\uccode`)=\nextuccode
    \ifx\nextnext\blankspace
      \prewordbreak)\relax
    \else
      \prewordbreak\discretionary{-)}{}{)}\prewordbreak
    \fi
  \else\ifnum\uccode`'=\nextuccode
    \prewordbreak\discretionary{-}{}{'}\postwordbreak
  \else\ifnum\uccode`<=\nextuccode
    \beginofsubsentence\prewordbreak\beginofsubsentencespacing
  \else\ifnum\uccode`>=\nextuccode
    \endofsubsentencespacing\prewordbreak\endofsubsentence
  \else
    \checkafterdiscretionary
```

```
      \bgroup
        \checkbeforediscretionary
        \prewordbreak\hbox{\betweendiscretionary\nextnext}\postwordbreak
      \egroup
    \fi\fi\fi\fi\fi\fi\fi\fi\fi
    \nextnextnext}
```

Now we can safely do things like:

```
    \catcode`<=\@@active  \def<{hello there}
    \catcode`>=\@@active  \def>{hello there}
    \catcode`(=\@@active  \def({hello there}
    \catcode`)=\@@active  \def){hello there}
```

In normal day–to–day production of texts this kind of activation is seldom used.[4] If so, we have to take care of the math mode explicitly, just like we did when making `|` active. It can be confusing too, especially when we load macropackages afterwards that make use of `<` in `\ifnum` or `\ifdim` statements.

\installcompoundcharac..

When Tobias Burnus started translating the dutch manual of PPCHTEX into german, he suggested to let CONTEXT support the `german.sty` method of handling compound characters, especially the umlaut. This package is meant for use with PLAIN TEX as well as LATEX.

I decided to implement compound character support as versatile as possible. As a result one can define his own compound character support, like:

```
    \installcompoundcharacter "a {\"a}
    \installcompoundcharacter "e {\"e}
    \installcompoundcharacter "i {\"i}
    \installcompoundcharacter "u {\"u}
```

[4] In the CONTEXT manual the < and > are made active and used for some cross–reference trickery.

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
   \installcompoundcharacter "o {\"o}
   \installcompoundcharacter "s {\SS}
```

or even

```
   \installcompoundcharacter "ck {\discretionary {k-}{k}{ck}}
   \installcompoundcharacter "ff {\discretionary{ff-}{f}{ff}}
```

The support is not limited to alphabetic characters, so the next definition is also valid.

```
   \installcompoundcharacter ". {.\doifnextcharelse{\spacetoken}{}{\kern.125em}}
```

The implementation looks familiar and uses the same tricks as mentioned earlier in this module. We take care of two arguments, which complicates things a bit.

16
```
\def\@nc@{@nc@} % normal character
\def\@cc@{@cc@} % compound character
\def\@cs@{@cs@} % compound characters
```

17
```
\def\installcompoundcharacter #1#2#3 #4%
  {\setvalue{\@nc@\string#1}{\char`#1}%
   \def\!!stringa{#3}%
   \ifx\!!stringa\empty
     \setvalue{\@cc@\string#1\string#2}{#4}%
   \else
     \setvalue{\@cs@\string#1\string#2\string#3}{#4}%
   \fi
   \catcode`#1=\@@active
   \scratchcounter=\the\uccode`~
   \uccode`~=`#1
   \uppercase{\unexpanded\def~{\handlecompoundcharacter~}}%
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

```
   \uccode`~=\scratchcounter}
```

In handling the compound characters we have to take care of `\bgroup` and `\egroup` tokens, so we end up with several interpretation macros.

18
```
\def\dohandlecompoundcharacter%
  {\ifx\next\bgroup
     \let\next=\relax
   \else\ifx\next\egroup
     \let\next=\relax
   \else
     \let\next=\dodohandlecompoundcharacter%
   \fi\fi
   \next}
```

After having taken care of the grouping tokens, we have to deal with three situations. First we look if the next character equals the first one, if so, then we just insert them both. Next we look is indeed a compound character is defined. We either execute the compound character or just insert the first. So we have

```
   <key><known>  <key><unknown>  <key><key>
```

We define these macros as `\long` because we can expect `\par` tokens. We need to look into the future with `\futurelet` to prevent spaces from disappearing.

19
```
\long\def\dododohandlecompoundcharacter#1#2#3%
  {\ifx#1#2%
     \def\next{\getvalue{\@nc@\string#1}\getvalue{\@nc@\string#1}}%
   \else
     \@EA\ifx\csname\@cs@\string#1\string#2\string#3\endcsname\relax
       \expandafter\ifx\csname\@cc@\string#1\string#2\endcsname\relax
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

```
        \def\next{\getvalue{\@nc@\string#1}#2#3}%
      \else
        \def\next{\getvalue{\@cc@\string#1\string#2}#3}%
      \fi
    \else
      \def\next{\getvalue{\@cs@\string#1\string#2\string#3}}%
    \fi
  \fi
  \next}
```

20

```
\long\def\dodohandlecompoundcharacter#1#2%
  {\ifx\next\blankspace
     \def\next{\dodododohandlecompoundcharacter#1#2\blankspace\ignorespaces}%
   \else
     \def\next{\dodododohandlecompoundcharacter#1#2}%
   \fi
   \next}
```

21

```
\long\def\handlecompoundcharacter#1#2%
  {\long\def\dohandlecompoundcharacter%
     {\dodohandlecompoundcharacter#1#2}%
   \futurelet\next\dohandlecompoundcharacter}
```

In later modules we will see how these commands are used.

22

```
\protect
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

\beginofsubsentence ●
\beginofsubsentencespacing ●

\compoundhyphen ●

\enableactivediscretionaries ●

\endofsubsentence ●
\endofsubsentencespacing ●

\installcompoundcharacter ●
\installdiscretionaries ●

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

4.5 METAPOST to PDF conversion

These macros are written as generic as possible. Some general support macro's are loaded from a small module especially made for non CONTEXT use. In this module I use a matrix transformation macro written by Tanmoy Bhattacharya. Thanks to extensive testing of Sebastian Ratz I was able to complete this module within reasonable time. First we take care of non–CONTEXT use:

```
\ifx \undefined \writestatus \input supp-mis.tex \relax \fi
```

This module handles some PDF conversion and insertions topics. The macros use the PDFTEX primitive \pdfliteral.

```
\writestatus{loading}{Context Support Macros / PDF}
```

```
\unprotect
```

```
\ifx\pdfliteral\undefined
  \def\pdfliteral#1{\message{[ignored pdfliteral: #1]}}
\fi
```

\convertPDFtoPDF

PDFTEX supports verbatim inclusion of PDF code. The following macro takes care of inserting externally defined illustrations in PDF format. According to a suggestion Tanmoy Bhattacharya posted to the PDFTEX mailing list, we first skip lines until **stream** is reached and then copy lines until **endstream** is encountered. This scheme only works with vectorized graphics in which no indirect references to objects are used. Bitmaps also don't work. Interpreting their specifications is beyond the current implementation.

```
\convertPDFtoPDF
  {filename}
  {x scale} {y scale}
  {x offset } {y offset}
```

```
    {width} {height}
```

When the scales are set to 1, the last last four values are the same as the bounding box, e.g.

```
    \convertPDFtoPDF{mp-pra-1.pdf} {1} {1}{-1bp}{-1bp}{398bp}{398bp}
    \convertPDFtoPDF{mp-pra-1.pdf}{.5}{.5} {0bp} {0bp}{199bp}{199bp}
```

Keep in mind, that this kind of copying only works for pure and valid pdf code (without fonts).

The scanning and copying is straightforward and quite fast. To speed up things we use two constants.

5
```
\def\@@PDFstream@@    {stream}
\def\@@PDFendstream@@ {endstream}
```

\PDFmediaboxprefered

If needed, the macros can scan for the mediabox that specifies the dimensions and offsets of the graphic. When we say:

```
    \PDFmediaboxpreferedtrue
```

the mediabox present in the file superseded the user specified, already scaled and calculated offset and dimensions. Beware: the user supplied values are not the bounding box ones!

6
```
\newif\ifPDFmediaboxprefered
```

7
```
\def\setPDFboundingbox#1#2#3#4#5#6%
  {\dimen0=#1\dimen0=#5\dimen0
   \ScaledPointsToBigPoints{\number\dimen0}\PDFxoffset
   \dimen0=#3\dimen0=#5\dimen0
   \xdef\PDFwidth{\the\dimen0}%
   \dimen0=#2\dimen0=#6\dimen0
   \ScaledPointsToBigPoints{\number\dimen0}\PDFyoffset
   \dimen0=#4\dimen0=#6\dimen0
```

```
    \xdef\PDFheight{\the\dimen0}%
    \global\let\PDFxoffset=\PDFxoffset
    \global\let\PDFyoffset=\PDFyoffset}
```

8
```
\def\setPDFmediabox#1[#2 #3 #4 #5]#6\done%
  {\dimen2=#2bp\dimen2=-\dimen2
   \dimen4=#3bp\dimen4=-\dimen4
   \dimen6=#4bp\advance\dimen6 by \dimen2
   \dimen8=#5bp\advance\dimen8 by \dimen4
   \setPDFboundingbox{\dimen2}{\dimen4}{\dimen6}{\dimen8}\PDFxscale\PDFyscale}
```

9
```
\def\checkPDFmediabox#1/MediaBox#2#3\done%
  {\ifx#2\relax \else
     \message{mediabox}%
     \setPDFmediabox#2#3\done
   \fi}
```

We use the general macro `\doprocessfile` and feed this with a line handling macro that changed it's behavior when the stream operators are encountered.

10
```
\def\handlePDFline%
  {\ifx\@@PDFstream@@\fileline
     \let\doprocessPDFline=\copyPDFobject
     \startPDFtoPDF
   \else\ifPDFmediaboxprefered
     \expandafter\checkPDFmediabox\fileline/MediaBox\relax\done
   \fi\fi}
```

11
```
\def\copyPDFobject%
  {\ifx\@@PDFendstream@@\fileline
     \ifPDFmediaboxprefered
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
      \let\doprocessPDFline=\findPDFmediabox
    \else
      \let\doprocessPDFline=\relax
    \fi
  \else
    \advance\scratchcounter by 1
    \pdfliteral{\fileline}%
  \fi}
```

12
```
\def\findPDFmediabox%
  {\expandafter\checkPDFmediabox\fileline/MediaBox\relax\done}
```

The main conversion macro wraps the PDF codes in a box that is output as an object. The graphics are embedded in `q` and `Q` and are scaled and positioned using one transform call (`cm`). This saves some additional scaling.

13
```
\def\startPDFtoPDF%
  {\setbox0=\vbox\bgroup
     \message{[PDF to PDF \PDFfilename}%
     \forgetall
     \scratchcounter=0
     \let\stopPDFtoPDF=\dostopPDftoPDF}
```

14
```
\def\dostopPDFtoPDF%
  {\ifnum\scratchcounter<0 \scratchcounter=1 \fi
   \message{(\the\scratchcounter\space lines)]}%
   \egroup
   \wd0=\PDFwidth
   \vbox to \PDFheight
     {\forgetall
      \vfill
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
      \pdfliteral{q}%
      \pdfliteral{1 0 0 1 \PDFxoffset\space \PDFyoffset\space cm}%
      \pdfliteral{\PDFxscale\space 0 0 \PDFyscale\space 0 0 cm}%
      \box0
      \pdfliteral{Q}}}
```

15
```
\def\stopPDFtoPDF%
   {\message{[PDF to PDF \PDFfilename\space not found]}}
```

16
```
\def\convertPDFtoPDF#1#2#3#4#5#6#7%
  {\bgroup
   \def\PDFfilename{#1}%
   \def\PDFxscale  {#2}%
   \def\PDFyscale  {#3}%
   \setPDFboundingbox{#4}{#5}{#6}{#7}{1}{1}%
   \uncatcodespecials
   \endlinechar=-1
   \let\doprocessPDFline=\handlePDFline
   \doprocessfile\scratchread\PDFfilename\doprocessPDFline
   \stopPDFtoPDF
   \egroup}
```

\convertMPtoPDF The next set of macros implements METAPOST to PDF conversion. Because we want to test as fast as possible, we first define the POSTSCRIPT operators that METAPOST uses. We don't define irrelevant ones, because these are skipped anyway.

17
```
\def \PScurveto          {curveto}
\def \PSlineto           {lineto}
\def \PSmoveto           {moveto}
\def \PSshowpage         {showpage}
\def \PSnewpath          {newpath}
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

```
\def \PSfshow              {fshow}
\def \PSclosepath          {closepath}
\def \PSfill               {fill}
\def \PSstroke             {stroke}
\def \PSclip               {clip}
\def \PSrlineto            {rlineto}
\def \PSsetlinejoin        {setlinejoin}
\def \PSsetlinecap         {setlinecap}
\def \PSsetmiterlimit      {setmiterlimit}
\def \PSsetgray            {setgray}
\def \PSsetrgbcolor        {setrgbcolor}
\def \PSsetdash            {setdash}
\def \PSgsave              {gsave}
\def \PSgrestore           {grestore}
\def \PStranslate          {translate}
\def \PSscale              {scale}
\def \PSconcat             {concat}
\def \PSdtransform          {dtransform}
```

18

```
\def \PSBoundingBox        {BoundingBox:}
\def \PSHiResBoundingBox {HiResBoundingBox:}
\def \PSExactBoundingBox {ExactBoundingBox:}
\def \PSPage               {Page:}
```

In POSTSCRIPT arguments precede the operators. Due to the fact that in some translations we need access to those arguments, as well as that sometimes we have to skip them, we stack them up. The stack is one–dimensional for non path operators and two–dimensional for operators inside a path. This is because we have to save the whole path for (optional) postprocessing. Values are pushed onto the stack by:

```
\setMPargument {value}
```

They can be retrieved by the short named macros:

```
\gMPa {number}
\sMPa {number}
```

When scanning a path specification, we also save the operator, using

```
\setMPkeyword {n}
```

The path drawing operators are coded for speed: `clip`, `stroke`, `fill` and `fillstroke` become 1, 2, 3 and 4.

When processing the path this code can be retrieved using

```
\getMPkeyword{n}
```

When setting an argument, the exact position on the stack depend on the current value of the ⟨*counters*⟩ `\nofMPsegments` and `\nofMParguments`.

19
```
\newcount\nofMPsegments
\newcount\nofMParguments
```

These variables hold the coordinates. The argument part of the stack is reset by:

```
\resetMPstack
```

We use the prefix `@@MP` to keep the stack from conflicting with existing macros. To speed up things bit more, we use the constant `\@@MP`.

20
```
\def\@@MP{@@MP}
```

21
```
\def\setMPargument#1%
  {\advance\nofMParguments by 1
   \expandafter\def
     \csname\@@MP\the\nofMPsegments\the\nofMParguments\endcsname%
     {\do#1}}
```

22
```
\def\gMPa#1%
  {\csname\@@MP0#1\endcsname}
```

23
```
\def\gMPs#1%
  {\csname\@@MP\the\nofMPsegments#1\endcsname}
```

24
```
\def\setMPkeyword#1
  {\expandafter\def\csname\@@MP\the\nofMPsegments0\endcsname{#1}%
   \advance\nofMPsegments by 1
   \nofMParguments=0\relax}
```

25
```
\def\getMPkeyword#1%
  {\csname\@@MP#10\endcsname}
```

When we reset the stack, we can assume that all further comment is to be ignored as well as handled in strings. By redefining the reset macro after the first call, we save some run time.

26
```
\def\resetMPstack%
  {\catcode`\%=\@@active
   \let\handleMPgraphic=\handleMPendgraphic
   \def\resetMPstack{\nofMParguments=0\relax}%
   \resetMPstack}
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

The arguments are saved with the preceding command `\do`. By default this command expands to nothing, but when we deal with strings it's used to strip off the `(` and `)`.

Strings are kind of tricky, because characters can be passed verbatim `(hello)`, by octal number `(\005)` or as command `(\()`. We therefore cannot simply ignore `(` and `)`, the way we do with `[` and `]`. Another complication is that strings may contain characters that normally have a special meaning in TEX, like `$` and `{}`.

A previous solution made `\` an active character and let it look ahead for a number or character. W ehad to abandon this scheme because of the need for verbatim support. The next solution involved some ⟨*catcode*⟩ trickery but works well.

27
```
\def\octalMPcharacter#1#2#3%
  {\char'#1#2#3\relax}
```

28
```
\bgroup
\catcode`\|=\@@comment
\catcode`\%=\@@active
\catcode`\[=\@@active
\catcode`\]=\@@active
\catcode`\{=\@@active
\catcode`\}=\@@active
\catcode`B=\@@begingroup
\catcode`E=\@@endgroup
\gdef\ignoreMPspecials|
  B\def%BE|
   \def[BE|
   \def]BE|
   \def{BE|
   \def}BEE
\gdef\obeyMPspecials|
  B\def%B\char 37\relax E|
   \def[B\char 91\relax E|
   \def]B\char 93\relax E|
   \def{B\char123\relax E|
   \def}B\char125\relax EE
\gdef\setMPspecials|
  B\catcode`\%=\@@active
   \catcode`\[=\@@active
   \catcode`\]=\@@active
   \catcode`\{=\@@active
   \catcode`\}=\@@active
   \catcode`\$=\@@letter
   \catcode`\_=\@@letter
   \catcode`\#=\@@letter
   \catcode`\^=\@@letter
   \catcode`\&=\@@letter
   \catcode`\|=\@@letter
   \catcode`\~=\@@letter
   \def\(B\char40\relax     E|
   \def\)B\char41\relax     E|
   \def\\B\char92\relax     E|
   \def\0B\octalMPcharacter0E|
   \def\1B\octalMPcharacter1E|
   \def\2B\octalMPcharacter2E|
   \def\3B\octalMPcharacter3E|
   \def\4B\octalMPcharacter4E|
   \def\5B\octalMPcharacter5E|
   \def\6B\octalMPcharacter6E|
   \def\7B\octalMPcharacter7E|
   \def\8B\octalMPcharacter8E|
   \def\9B\octalMPcharacter9EE
\egroup
```

We use the comment symbol as a sort of trigger:

29
```
\bgroup
\catcode`\%=\@@active
\gdef\startMPscanning{\let%=\startMPconversion}
\egroup
```

In earlier versions we used the sequence

```
   \expandafter\handleMPsequence\input filename\relax
```

Persistent problems in LaTeX however forced us to use a different scheme. Every POSTSCRIPT file starts with a `%`, so we temporary make this an active character that starts the scanning and redefines itself. (The problem originates in the redefinition by LaTeX of the `\input` primitive.)

30
```
\def\startMPconversion%
  {\catcode`\%=\@@ignore
   \ignoreMPspecials
   \handleMPsequence}
```

Here comes the main loop. Most arguments are numbers. This means that they can be recognized by their `\lccode`. This method saves a lot of processing time. We could speed up the conversion by handling the `path` seperately.

31
```
\def\dohandleMPsequence#1#2 %
  {\ifnum\lccode`#1=0
     \setMPargument{#1#2}%
   \else
```

```
\edef\somestring{#1#2}%
\ifx\somestring\PSmoveto
  \edef\lastMPmoveX{\gMPa1}%
  \edef\lastMPmoveY{\gMPa2}%
  \pdfliteral{\gMPa1 \gMPa2 m}%
  \resetMPstack
\else\ifx\somestring\PSnewpath
  \let\handleMPsequence=\handleMPpath
\else\ifx\somestring\PSgsave
  \pdfliteral{q}%
  \resetMPstack
\else\ifx\somestring\PSgrestore
  \pdfliteral{Q}%
  \resetMPstack
\else\ifx\somestring\PSdtransform  % == setlinewidth
  \let\handleMPsequence=\handleMPdtransform
\else\ifx\somestring\PSconcat
  \pdfliteral{\gMPa1 \gMPa2 \gMPa3 \gMPa4 \gMPa5 \gMPa6 cm}%
  \resetMPstack
\else\ifx\somestring\PSsetrgbcolor
  \pdfliteral{\gMPa1 \gMPa2 \gMPa3 rg \gMPa1 \gMPa2 \gMPa3 RG}%
  \resetMPstack
\else\ifx\somestring\PSsetgray
  \pdfliteral{\gMPa1 g \gMPa1 G}%
  \resetMPstack
\else\ifx\somestring\PStranslate
  \pdfliteral{1 0 0 1 \gMPa1 \gMPa2 cm}%
  \resetMPstack
\else\ifx\somestring\PSsetdash
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
      \handleMPsetdash
      \resetMPstack
    \else\ifx\somestring\PSsetlinejoin
      \pdfliteral{\gMPa1 j}%
      \resetMPstack
    \else\ifx\somestring\PSsetmiterlimit
      \pdfliteral{\gMPa1 M}%
      \resetMPstack
    \else\ifx\somestring\PSfshow
      \handleMPfshow
      \resetMPstack
    \else\ifx\somestring\PSsetlinecap
      \pdfliteral{\gMPa1 J}%
      \resetMPstack
    \else\ifx\somestring\PSrlineto
      \pdfliteral{\lastMPmoveX\space \lastMPmoveY\space l S}%
      \resetMPstack
    \else\ifx\somestring\PSscale
      \pdfliteral{\gMPa1 0 0 \gMPa2 0 0 cm}%
      \resetMPstack
    \else
      \handleMPgraphic{#1#2}%
    \fi\fi\fi\fi\fi\fi\fi\fi
    \fi\fi\fi\fi\fi\fi\fi\fi
  \fi
  \handleMPsequence}
```

Beginning and ending the graphics is taken care of by the macro `\handleMPgraphic`, which is redefined when the first graphics operator is met.

32
```
\def\handleMPendgraphic#1%
  {\ifx\somestring\PSshowpage
     \let\handleMPsequence=\finishMPgraphic
   \else
     \setMPargument{#1}%
   \fi}
```

33
```
\def\handleMPbegingraphic#1%
  {\ifx\somestring\PSBoundingBox
     \let\handleMPsequence=\handleMPboundingbox
   \else\ifx\somestring\PSHiResBoundingBox
     \let\handleMPsequence=\handleMPboundingbox
   \else\ifx\somestring\PSExactBoundingBox
     \let\handleMPsequence=\handleMPboundingbox
   \else\ifx\somestring\PSPage
     \let\handleMPsequence=\handleMPpage
   \else
     \setMPargument{#1}%
   \fi\fi\fi\fi}
```

34
```
\let\handleMPgraphic=\handleMPbegingraphic
```

We check for three kind of bounding boxes: the normal one and two high precission ones:

```
BoundingBox: llx lly ucx ucy
HiResBoundingBox: llx lly ucx ucy
ExactBoundingBox: llx lly ucx ucy
```

The dimensions are saved for later use.

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

35
```
\def\handleMPboundingbox #1 #2 #3 #4
  {\dimen0=#1pt\dimen0=-\MPxscale\dimen0
   \dimen2=#2pt\dimen2=-\MPyscale\dimen2
   \xdef\MPxoffset{\withoutpt{\the\dimen0}}%
   \xdef\MPyoffset{\withoutpt{\the\dimen2}}%
   \dimen0=#1bp\dimen0=-\dimen0
   \dimen2=#2bp\dimen2=-\dimen2
   \advance\dimen0 by #3bp
   \dimen0=\MPxscale\dimen0
   \xdef\MPwidth{\the\dimen0}%
   \advance\dimen2 by #4bp
   \dimen2=\MPyscale\dimen2
   \xdef\MPheight{\the\dimen2}%
   \nofMParguments=0
   \let\handleMPsequence=\dohandleMPsequence
   \handleMPsequence}
```

We use the `page` comment as a signal that stackbuilding can be started.

36
```
\def\handleMPpage #1 #2
  {\nofMParguments=0
   \let\handleMPsequence=\dohandleMPsequence
   \handleMPsequence}
```

METAPOST draws it dots by moving to a location and invoking `0 0 rlineto`. This operator is not available in PDF. Our solution is straightforward: we draw a line from $(current\_x, current\_y)$ to itself. This means that the arguments of the preceding `moveto` have to be saved.

37
```
\def\lastMPmoveX{0}
\def\lastMPmoveY{0}
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

These saved coordinates are also used when we handle the texts. Text handling proved to be a bit of a nuisance, but finaly I saw the light. It proved that we also had to take care of `(split arguments)`.

38

```
\def\handleMPfshow%
  {\setbox0=\hbox
     {\obeyMPspecials
      \edef\size{\gMPa{\the\nofMParguments} }%
      \advance\nofMParguments by -1
      \font\temp=\gMPa{\the\nofMParguments} at \size bp
      \advance\nofMParguments by -1
      \temp
      \ifnum\nofMParguments=1
        \def\do(##1){##1}%
        \gMPa1%
      \else
        \scratchcounter=1
        \def\do(##1{##1}%
        \gMPa{\the\scratchcounter}\space
        \def\do{}%
        \loop
          \advance\scratchcounter by 1
          \ifnum\scratchcounter<\nofMParguments
            \gMPa{\the\scratchcounter}\space
        \repeat
        \def\do##1){##1}%
        \gMPa{\the\scratchcounter}%
      \fi
      \unskip}%
   \dimen0=\lastMPmoveY bp
   \advance\dimen0 by \ht0
   \ScaledPointsToBigPoints{\number\dimen0}\lastMPmoveY
   \pdfliteral{n q 1 0 0 1 \lastMPmoveX\space\lastMPmoveY\space cm}%
   \dimen0=\ht0
   \advance\dimen0 by \dp0
   \box0
   \vskip-\dimen0
   \pdfliteral{Q}}
```

Most operators are just converted and keep their arguments. Dashes however need a bit different treatment, otherwise PDF viewers complain loudly. Another complication is that one argument comes after the]. When reading the data, we simple ignore the array boundary characters. We save ourselves some redundant newlines and at the same time keep the output readable by packing the literals.

39
```
\def\handleMPsetdash%
  {\bgroup
   \def\somestring{[}%
   \scratchcounter=1
   \loop
   \ifnum\scratchcounter<\nofMParguments
     \edef\somestring{\somestring\space\gMPa{\the\scratchcounter}}%
     \advance\scratchcounter by 1
   \repeat
   \edef\somestring{\somestring]\gMPa{\the\scratchcounter} d}%
   \pdfliteral{\somestring}%
   \egroup}
```

The `setlinewidth` commands look a bit complicated. There are two alternatives, that alsways look the same. As John Hobby says:

```
      x 0 dtransform exch truncate exch idtransform pop setlinewidth
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
0 y dtransform truncate idtransform setlinewidth pop
```

> These are just fancy versions of `x setlinewidth` and `y setlinewidth`. The `x 0 ...` form is used if the path is *primarily vertical*. It rounds the width so that vertical lines come out an integer number of pixels wide in device space. The `0 y ...` form does the same for paths that are *primarily horizontal*. The reason why I did this is Knuth insists on getting exactly the widths TEX intends for the horizontal and vertical rules in `btex...etex` output. (Note that PostScript scan conversion rules cause a horizontal or vertical line of integer width n in device space to come out $n+1$ pixels wide, regardless of the phase relative to the pixel grid.)

The common operator in these sequences is `dtransform`, so we can use this one to trigger setting the linewidth.

40
```
\def\handleMPdtransform%
  {\ifdim\gMPa1pt>\!!zeropoint
     \pdfliteral{\gMPa1 w}%
     \def\next##1 ##2 ##3 ##4 ##5 ##6 {\handleMPsequence}%
   \else
     \pdfliteral{\gMPa2 w}%
     \def\next##1 ##2 ##3 ##4 {\handleMPsequence}%
   \fi
   \let\handleMPsequence=\dohandleMPsequence
   \resetMPstack
   \next}
```

The most complicated command is `concat`. METAPOST applies this operator to `stoke`. At that moment the points set by `curveto` and `moveto`, are already fixed. In PDF however the `cm` operator affects the points as well as the pen (stroke). Like more PDF operators, `cm` is a defined in a bit ambiguous way. The only save route for non–circular penshapes, is saving teh path, recalculating the points and applying the transformation matrix in such a way that we can be sure that its behavior is well defined. This comes down to inverting the path and applying `cm` to that path as well as the pen. This all means that we have to save the path.

In METAPOST there are three ways to handle a path p:

```
draw p;  fill p;  filldraw p;
```

The last case outputs a `gsave fill grestore` before `stroke`. Handling the path outside the main loops saves about 40% run time.[5] Switching between the main loop and the path loop is done by means of the recursely called macro `\handleMPsequence`.

41
```
\def\handleMPpath%
  {\chardef\finiMPpath=0
   \let\closeMPpath=\relax
   \let\flushMPpath=\flushnormalMPpath
   \resetMPstack
   \nofMPsegments=1
   \let\handleMPsequence=\dohandleMPpath
   \dohandleMPpath}
```

Most paths are drawn with simple round pens. Therefore we've split up the routinein two.

42
```
\def\flushnormalMPpath%
  {\scratchcounter=\nofMPsegments
   \nofMPsegments=1
   \loop
     \expandafter\ifcase\getMPkeyword{\the\nofMPsegments}\relax
       \pdfliteral{\gMPs1 \gMPs2 l}%
     \or
       \pdfliteral{\gMPs1 \gMPs2 \gMPs3 \gMPs4 \gMPs5 \gMPs6 c}%
```

[5] We can save some more by following the METAPOST output routine, but for the moment we keep things simple.

```
      \or
        \pdfliteral{\lastMPmoveX\space \lastMPmoveY\space l S}%
      \or
        \edef\lastMPmoveX{\gMPs1}%
        \edef\lastMPmoveY{\gMPs2}%
        \pdfliteral{\lastMPmoveX\space \lastMPmoveY\space m}%
      \fi
      \advance\nofMPsegments by 1\relax
    \ifnum\nofMPsegments<\scratchcounter
    \repeat}
```

43
```
\def\flushconcatMPpath%
  {\scratchcounter=\nofMPsegments
    \nofMPsegments=1
    \loop
      \expandafter\ifcase\getMPkeyword{\the\nofMPsegments}\relax
        \doMPconcat{\gMPs1}\a{\gMPs2}\b
        \pdfliteral{\a\space \b\space l}%
      \or
        \doMPconcat{\gMPs1}\a{\gMPs2}\b
        \doMPconcat{\gMPs3}\c{\gMPs4}\d
        \doMPconcat{\gMPs5}\e{\gMPs6}\f
        \pdfliteral{\a\space \b\space \c\space \d\space \e\space \f\space c}%
      \or
        \bgroup
        \noMPtranslate
        \doMPconcat\lastMPmoveX\a\lastMPmoveY\b
        \pdfliteral{\a\space \b\space l S}%
        \egroup
      \or
      \edef\lastMPmoveX{\gMPs1}%
      \edef\lastMPmoveY{\gMPs2}%
      \doMPconcat\lastMPmoveX\a\lastMPmoveY\b
      \pdfliteral{\a\space \b\space m}%
    \fi
    \advance\nofMPsegments by 1\relax
  \ifnum\nofMPsegments<\scratchcounter
  \repeat}
```

The transformation of the coordinates is handled by one of the macros Tanmoy posted to the PDFTEX mailing list. I rewrote and optimized the original macro to suit the other macros in this module.

```
  \doMPconcat {x position} \xresult {y position} \yresult
```

By setting the auxiliary ⟨*dimensions*⟩ `\dimen0` upto `\dimen10` only once per path, we save over 20% run time. Some more speed was gained by removing some parameter passing. These macros can be optimized a bit more by using more constants. There is however not much need for further optimization because penshapes usually are round and therefore need no transformation. Nevertheless we move the factor to the outer level and use bit different `pt` removal macro. Although the values represent base points, we converted them to pure points, simply because those can be converted back.

44
```
\def\MPconcatfactor{256}
```

45
```
\def\doMPreducedimen#1
  {\count0=\MPconcatfactor
   \advance\dimen#1 \ifdim\dimen#1>\!!zeropoint .5\else -.5\fi\count0
   \divide\dimen#1 \count0\relax}
```

46
```
\def\doMPexpanddimen#1
  {\multiply\dimen#1 \MPconcatfactor\relax}
```

47
```
\def\presetMPconcat%
  {\dimen 0=\gMPs1 pt \doMPreducedimen 0    % r_x
   \dimen 2=\gMPs2 pt \doMPreducedimen 2    % s_x
   \dimen 4=\gMPs3 pt \doMPreducedimen 4    % s_y
   \dimen 6=\gMPs4 pt \doMPreducedimen 6    % r_y
   \dimen 8=\gMPs5 pt \doMPreducedimen 8    % t_x
   \dimen10=\gMPs6 pt \doMPreducedimen10 }  % t_y
```

48
```
\def\noMPtranslate% use this one grouped
  {\dimen 8=\!!zeropoint                    % t_x
   \dimen10=\!!zeropoint}                   % t_y
```

49
```
\def\doMPconcat#1#2#3#4%
  {\dimen12=#1 pt \doMPreducedimen12        % p_x
   \dimen14=#3 pt \doMPreducedimen14        % p_y
   %
   \dimen16  \dimen 0
   \multiply \dimen16  \dimen 6
   \dimen20  \dimen 2
   \multiply \dimen20  \dimen 4
   \advance  \dimen16 -\dimen20
   %
   \dimen18  \dimen12
   \multiply \dimen18  \dimen 6
   \dimen20  \dimen14
   \multiply \dimen20  \dimen 4
   \advance  \dimen18 -\dimen20
   \dimen20  \dimen 4
   \multiply \dimen20  \dimen10
   \advance  \dimen18  \dimen20
\dimen20  \dimen 6
\multiply \dimen20  \dimen 8
\advance  \dimen18 -\dimen20
%
\multiply \dimen12 -\dimen 2
\multiply \dimen14  \dimen 0
\advance  \dimen12  \dimen14
\dimen20  \dimen 2
\multiply \dimen20  \dimen 8
\advance  \dimen12  \dimen20
\dimen20  \dimen 0
\multiply \dimen20  \dimen10
\advance  \dimen12 -\dimen20
%
\doMPreducedimen16
\divide   \dimen18  \dimen16 \doMPexpanddimen18
\divide   \dimen12  \dimen16 \doMPexpanddimen12
%
\edef#2{\withoutpt{\the\dimen18}}%       % p_x^\prime
\edef#4{\withoutpt{\the\dimen12}}}       % p_y^\prime
```

The following explanation of the conversion process was posted to the PDFTEX mailing list by Tanmoy. The original macro was part of a set of macro's that included sinus and cosinus calculation as well as scaling and translating. The METAPOST to PDF conversion however only needs transformation.

Given a point (U_x, U_y) in user coordinates, the business of POSTSCRIPT is to convert it to device space. Let us say that the device space coordinates are (D_x, D_y). Then, in POSTSCRIPT (D_x, D_y) can be written in terms of (U_x, U_y) in matrix notation, either as

$$\begin{pmatrix} D_x & D_y & 1 \end{pmatrix} = \begin{pmatrix} U_x & U_y & 1 \end{pmatrix} \begin{pmatrix} s_x & r_x & 0 \\ r_y & s_y & 0 \\ t_x & t_y & 1 \end{pmatrix} \tag{4.1}$$

or

$$\begin{pmatrix} D_x \\ D_y \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & r_y & t_x \\ r_x & s_y & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} U_x \\ U_y \\ 1 \end{pmatrix} \tag{4.2}$$

both of which is a shorthand for the same set of equations:

$$D_x = s_x U_x + r_y U_y + t_x \tag{4.3}$$

$$D_y = r_x U_x + s_y U_y + t_y \tag{4.4}$$

which define what is called an 'affine transformation'.

POSTSCRIPT represents the 'transformation matrix' as a six element matrix instead of a 3×3 array because three of the elements are always 0, 0 and 1. Thus the above transformation is written in postscript as $[s_x\, r_x\, r_y\, s_y\, t_x\, t_y]$. However, when doing any calculations, it is useful to go back to the original matrix notation (whichever: I will use the second) and continue from there.

As an example, if the current transformation matrix is $[s_x\, r_x\, r_y\, s_y\, t_x\, t_y]$ and you say `[a b c d e f] concat`, this means:

> Take the user space coordinates and transform them to an intermediate set of coordinates using array $[a\, b\, c\, d\, e\, f]$ as the transformation matrix.
>
> Take the intermediate set of coordinates and change them to device coordinates using array $[s_x\, r_x\, r_y\, s_y\, t_x\, t_y]$ as the transformation matrix.

Well, what is the net effect? In matrix notation, it is

$$\begin{pmatrix} I_x \\ I_y \\ 1 \end{pmatrix} = \begin{pmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} U_x \\ U_y \\ 1 \end{pmatrix} \tag{4.5}$$

$$\begin{pmatrix} D_y \\ D_y \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & r_y & t_x \\ r_x & s_y & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} I_x \\ I_y \\ 1 \end{pmatrix} \tag{4.6}$$

where (I_x, I_y) is the intermediate coordinate.

Now, the beauty of the matrix notation is that when there is a chain of such matrix equations, one can always compose them into one matrix equation using the standard matrix composition law. The composite matrix from two matrices can be derived very easily: the element in the i^{th} horizontal row and j^{th} vertical column is calculated by'multiplying' the i^{th} row of the first matrix and the j^{th} column of the second matrix (and summing over the elements). Thus, in the above:

$$\begin{pmatrix} D_x \\ D_y \\ 1 \end{pmatrix} = \begin{pmatrix} s'_x & r'_y & t'_x \\ r'_x & s'_y & t'_y \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} U_x \\ U_y \\ 1 \end{pmatrix} \tag{4.7}$$

with

$$\begin{aligned} s'_x &= s_x a + r_y b \\ r'_x &= r_x a + s_y b \\ r'_y &= s_x c + r_y d \\ s'_y &= r_x c + s_y d \\ t'_x &= s_x e + r_y f + t_x \\ t'_y &= r_x e + s_y f + t_y \end{aligned} \tag{4.8}$$

In fact, the same rule is true not only when one is going from user coordinates to device coordinates, but whenever one is composing two 'transformations' together (transformations are 'associative'). Note that the formula is not symmetric: you have to keep track of which transformation existed before (i.e. the equivalent of $[s_x\, r_x\, r_y\, s_y\, t_x\, t_y]$) and which was specified later (i.e. the equivalent of $[a\, b\, c\, d\, e\, f]$). Note also that the language can be rather confusing: the one specified later 'acts earlier', converting the user space coordinates to intermediate coordinates, which are then acted upon by the pre–existing transformation. The important point is that order of transformation matrices cannot be flipped (transformations are not 'commutative').

Now what does it mean to move a transformation matrix before a drawing? What it means is that given a point (P_x, P_y) we need a different set of coordinates (P'_x, P'_y) such that if the transformation acts on (P'_x, P'_y), they produce (P_x, P_y). That is we need to solve the set of equations:

$$\begin{pmatrix} P_x \\ P_y \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & r_y & t_x \\ r_x & s_y & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P'_x \\ P'_y \\ 1 \end{pmatrix} \tag{4.9}$$

Again matrix notation comes in handy (i.e. someone has already solved the problem for us): we need the inverse transformation matrix. The inverse transformation matrix can be calculated very easily: it is

$$\begin{pmatrix} P'_x \\ P'_y \\ 1 \end{pmatrix} = \begin{pmatrix} s'_x & r'_y & t'_x \\ r'_x & s'_y & t'_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ 1 \end{pmatrix} \tag{4.10}$$

where, the inverse transformation matrix is given by

$$\begin{aligned} D &= s_x s_y - r_x r_y \\ s_x' &= s_y/D \\ s_y' &= s_x/D \\ r_x' &= -r_x/D \\ r_y' &= -r_y/D \\ t_x' &= (-s_y t_x + r_y t_y)/D \\ t_y' &= (r_x t_x - s_x t_y)/D \end{aligned} \tag{4.11}$$

And you can see that when expanded out, this does give the formulas:

$$P_x' = \frac{s_y(p_x - t_x) + r_y(t_y - p_y)}{s_x * s_y - r_x * r_y} \tag{4.12}$$

$$P_y' = \frac{s_x(p_y - t_y) + r_x(t_x - p_x)}{s_x * s_y - r_x * r_y} \tag{4.13}$$

The code works by representing a real number by converting it to a dimension to be put into a ⟨*dimension*⟩ register: 2.3 would be represented as 2.3pt for example. In this scheme, multiplying two numbers involves multiplying the ⟨*dimension*⟩ registers and dividing by 65536. Accuracy demands that the division be done as late as possible, but overflow considerations need early division.

Division involves dividing the two ⟨*dimension*⟩ registers and multiplying the result by 65536. Again, accuracy would demand that the numerator be multiplied (and/or the denominator divided) early: but that can lead to overflow which needs to be avoided.

If nothing is known about the numbers to start with (in concat), I have chosen to divide the 65536 as a 256 in each operand. However, in the series calculating the sine and cosine, I know that the terms are small (because I never have an angle greater than 45 degrees), so I chose to apportion the factor in a different way.

The path is output using the values saved on the stack. If needed, all coordinates are recalculated.

50
```
\def\processMPpath%
  {\flushMPpath
   \closeMPpath
   \pdfliteral{\ifcase\finiMPpath W n\or S\or f\or B\fi}%
   \let\handleMPsequence=\dohandleMPsequence
   \resetMPstack
   \nofMPsegments=0
   \handleMPsequence}
```

In PDF the cm operator must precede the path specification. We therefore can output the cm at the moment we encounter it.

51
```
\def\handleMPpathconcat%
  {\presetMPconcat
   \pdfliteral{\gMPs1 \gMPs2 \gMPs3 \gMPs4 \gMPs5 \gMPs6 cm}%
   \resetMPstack}
```

This macro interprets the path and saves it as compact as possible.

52
```
\def\dohandleMPpath#1#2 %
  {\ifnum\lccode`#1=0
     \setMPargument{#1#2}%
   \else
     \def\somestring{#1#2}%
     \ifx\somestring\PSlineto
       \setMPkeyword0
     \else\ifx\somestring\PScurveto
       \setMPkeyword1
     \else\ifx\somestring\PSrlineto
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
      \setMPkeyword2
    \else\ifx\somestring\PSmoveto
      \setMPkeyword3
    \else\ifx\somestring\PSclip
      \let\handleMPsequence=\processMPpath
    \else\ifx\somestring\PSgsave
      \chardef\finiMPpath=3
    \else\ifx\somestring\PSgrestore
    \else\ifx\somestring\PSfill
      \ifnum\finiMPpath=0
        \chardef\finiMPpath=2
        \let\handleMPsequence=\processMPpath
      \fi
    \else\ifx\somestring\PSstroke
      \ifnum\finiMPpath=0
        \chardef\finiMPpath=1
      \fi
      \let\handleMPsequence=\processMPpath
    \else\ifx\somestring\PSclosepath
      \def\closeMPpath{\pdfliteral{h}}%
    \else\ifx\somestring\PSconcat
      \let\flushMPpath=\flushconcatMPpath
      \handleMPpathconcat
    \fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
  \fi
  \handleMPsequence}
```

The main conversion command is

```
  \convertMPtoPDF {filename} {x scale} {y scale}
```

The dimensions are derived from the bounding box. So we only have to say:

```
   \convertMPtoPDF{mp-pra-1.eps}{1}{1}
   \convertMPtoPDF{mp-pra-1.eps}{.5}{.5}
```

53
```
\def\convertMPtoPDF#1#2#3%
  {\bgroup
   \message{[MP to PDF #1]}%
   \setMPspecials
   \startMPscanning
   \def\do{}%
   \edef\MPxscale{#2}%
   \edef\MPyscale{#3}%
   \setbox0=\vbox\bgroup
     \forgetall
     \offinterlineskip
     \pdfliteral{q}%
     \let\handleMPsequence=\dohandleMPsequence
     \input #1\relax}
```

54
```
\def\finishMPgraphic%
  {\pdfliteral{Q}%
   \egroup
   \wd0=\MPwidth
   \vbox to \MPheight
     {\forgetall
      \vfill
      \pdfliteral{q \MPxscale\space 0 0 \MPyscale\space
        \MPxoffset\space \MPyoffset\space cm}%
      \box0
      \pdfliteral{Q}}%
   \egroup}
```

This kind of conversion is possible because METAPOST does all the calculations. Converting other POSTSCRIPT files would drive both me and TEX crazy.

```
\protect \endinput
```

\convertMPtoPDF ●	\PDFmediaboxprefered ●
\convertPDFtoPDF ●	

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

4.6 Specials

1 `\unprotect`

This module implements some `\special` manipulation macros. I needed these when I implemented the code that handles the conversion of TPIC specials to PDF code.

2 `\writestatus{loading}{Context Support Macros / Specials}`

When interpreting specials we need to do some basic scanning. For the moment we distinguish between three cases. We need

```
\special{tag: arguments}
\special{tag arguments}
\special{tag}
```

We cannot be sure that the first case isn't

```
\special{tag:arguments}
```

So we have to take care of that one too.

\redefinespecial Specials that are to be interpreted are defined with commands like:

```
\redefinespecial a: \using#1\endspecial%
  {let's execute special 'a:' using '#1'}

\redefinespecial a  \using#1\endspecial%
  {let's execute special 'a' using '#1'}

\redefinespecial a  \using#1\endspecial%
  {let's execute special 'a' using nothing}
```

The first two always take an argument, the last one not. The definition of this redefinition macro is not that complex. The names are internally tagged with `\@rds@` which saves both time and space.

3
```
\def\@rds@{@rds@}
```

4
```
\def\redefinespecial #1 %
  {\setvalue{\@rds@#1}}
```

`\mimmickspecials` Mimmicking specials is activated by saying:

```
   \mimmickspecials
```

This commands redefines the PLAIN TEX primitive `\special`.

5
```
\def\mimmickspecials%
  {\let\special=\domimmickspecial}
```

The special mimmicking macro first looks if it can find an colon terminated tag, next it searches for a tag that end with a space. If both cannot find, the tag itself is treated without argument.

6
```
\def\domimmickspecial#1%
  {\domimmickcolonspecial#1:\relax/:\relax/\end}
```

7
```
\def\domimmickcolonspecial#1:#2#3:\relax/#4\end%
  {\ifx#2\relax
     \domimmickspacespecial#1 \relax/ \relax/\end
   \else
     \dodomimmickspecial#1:\using#2#3\endspecial
   \fi}
```

8
```
\def\domimmickspacespecial#1 #2#3 \relax/#4\end%
  {\ifx#2\relax
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

```
     \dodomimmickspecial#1\using\endspecial
   \else
     \dodomimmickspecial#1\using#2#3\endspecial
   \fi}
```

9
```
\def\dodomimmickspecial#1\using#2\endspecial%
  {\expandafter\ifx\csname\@rds@#1\endcsname\relax % \doifdefinedelse
     \defaultspecial{#2}%
   \else
     %\message{[mimmick special #1 with #2#3]}%
     \getvalue{\@rds@#1}\using#2\endspecial
   \fi}
```

Now let's show that things work the way we want, using the previous definitions of tag a.

```
   \mimmickspecials
   \special{a: 1 2 3 4 5}
   \special{a: 1 2 3 4 5}
   \special{a}
```

Which results in:

let's execute special 'a:' using '1 2 3 4 5'
let's execute special 'a:' using '1 2 3 4 5'
let's execute special 'a' using nothing

\mimmickspecial When needed, one can call a mimmicked special directly by saying for instance:

```
   \mimmickspecial a: \using...\endspecial
```

This can be handy when specials have much in common.

10
```
\def\mimmickspecial #1 %
  {\getvalue{\@rds@#1}}
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

\normalspecial
\defaultspecial

Unknown specials are passed to the default special handler. One can for instance ignore all further specials by saying `\normalspecial`:

```
   \def\defaultspecial#1{\normalspecial}
```

But here we default to idle.

11
```
\let\normalspecial =\special
\let\defaultspecial=\special
```

12
```
\protect \endinput
```

`\defaultspecial` ●	`\normalspecial` ●
`\mimmickspecial` ●	`\redefinespecial` ●
`\mimmickspecials` ●	

4.7 METAPOST Inclusion

METAPOST is John Hobbys alternative for METAFONT and produces superior POSTSCRIPT code. In this module we integrate METAPOST support int CONTEXT. We offer two tracks:

- generating METAPOST code, running this program from within TEX using `\write18`, and importing the result
- generating METAPOST code, processing the code afterward, and importing the result in a second pass

The first approach uses a non standard TEX feature, implemented in Web2c. I'm not going to discuss the pros and cons of running programs from within others, but all arguments against this can be overcome by implementing a TEX worthy primitive:

```
\excuteMetaPost filename
```

Ok then, let's start:

1
```
\writestatus{loading}{Context Support Macros / MetaPost Inclusion}
```

2
```
\unprotect
```

\startMPgraphic

From within TEX one can execute METAPOST code by putting it between the two commands

```
\startMPgraphic
\stopMPgraphic
```

This is implemented as:

3
```
\def\startMPgraphic#1\stopMPgraphic%
  {\startwritingMPgraphic
   \writeMPgraphic{#1}%
   \stopwritingMPgraphic}
```

\startwritingMPgraphic
\writeMPgraphic
\stopwritingMPgraphic

If the writing process is divided into more steps, one can use the components of this macro directly.

```
\startwritingMPgraphic
...
\writeMPgraphic{...}
...
\writeMPgraphic{...}
...
\stopwritingMPgraphic
```

\ifrunMPgraphics

These macros look a bit more complicated that one would expect at first sight. This is due to the two ways of processing these gaphics, mentioned in a previous paragraph. Which method is used, the direct or indirect one, depends on a boolean.

4
```
\newif\ifrunMPgraphics
```

If set to true, one can do with a single pass, else one must process the METAPOST file `mpgraph` between two succesive TEX runs.

5
```
\def\MPgraphicfile{mpgraph}
```

When we run METAPOST from within TEX, each graphic is processed at once, which means that we reuse this file many times. When however the execution is delayed, all graphics are saved in a separate figure. The current graphic is characterized bij a number:

6
```
\newcount\currentMPgraphic
```

The three macros responsible for writing the graphic implement both schemes.

7
```
\def\writeMPgraphic%
  {\immediate\write\scratchwrite}
```

8
```
\def\startwritingMPgraphic%
  {\ifrunMPgraphics
     \global\currentMPgraphic=1
     \immediate\openout\scratchwrite=\MPgraphicfile.mp
   \else
     \global\advance\currentMPgraphic by 1
     \ifnum\currentMPgraphic=1
       \immediate\openout\scratchwrite=\MPgraphicfile.mp
     \fi
   \fi
   \immediate\write\scratchwrite{beginfig(\the\currentMPgraphic)}%
   \global\let\flushMPgraphics\dodostopwritingMPgraphic
   \global\let\stopwritingMPgraphic=\dostopwritingMPgraphic}
```

9
```
\def\dostopwritingMPgraphic%
  {\immediate\write\scratchwrite{endfig;}%
   \ifrunMPgraphics
     \dodostopwritingMPgraphic
   \fi}
```

10
```
\def\dodostopwritingMPgraphic%
  {\ifnum\currentMPgraphic>0
     \immediate\write\scratchwrite{end.}%
     \immediate\closeout\scratchwrite
     \runMPgraphic{\MPgraphicfile}%
   \fi
   \global\let\flushMPgraphics=\relax}
```

11
```
\let\stopwritingMPgraphic=\relax
\let\flushMPgraphics     =\relax
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

\flushMPgraphics

When we use the indirect method, all graphics are saved in one file. This means that we cannot close this file after every `\stopMPgraphic`. Therefore we need to say:

```
\startMPgraphic
\stopMPgraphic
```

else the file is closed without writing the METAPOST end command. One will notice this fast enough when in indirect mode. When using the direct mode this command is not implicitly needed, but ommiting it makes files less portable.

\loadcurrentMPgraphic
\placeMPgraphic

Once defined, we can call for this graphic by saying:

```
\loadcurrentMPgraphic{setups}
\placeMPgraphic
```

This two stage insert permits some intermediate manipulations of the graphic, which temporary saved in:

12
```
\newbox\MPgraphic
```

13
```
\def\loadcurrentMPgraphic#1%
  {\loadMPgraphic{\MPgraphicfile.\the\currentMPgraphic}{#1}}
```

14
```
\def\loadMPgraphic#1#2%
  {\setbox\MPgraphic=\hbox{\insertMPfile{#1}{#2}}}
```

15
```
\def\placeMPgraphic%
  {\box\MPgraphic}
```

We didn't yet define the macro responsible for processing the graphic from within TEX.

16
```
\def\runMPgraphic#1%
  {\ifrunMPgraphics
```

```
      \executeMetaPost{#1}%
    \else
      \message{[flush and process \MPgraphicfile.mp afterwards]}%
    \fi}
```

\executeMetaPost

With `\executeMetaPost` being defined as:

17
```
\ifx\undefined\executeMetaPost
  \def\executeMetaPost#1{\immediate\write18{mpost #1}}
\fi
```

\insertMPfile

One can define this command in advance or redefine it after loading this module. The same goes for the forward reference to the figure loading macro:

18
```
\ifx\undefined\insertMPfile
```

19
```
  \def\insertMPfile#1#2%
      {\ifx\undefined\externalfigure
         \message{[insert file #1 here]}%
       \else
         \externalfigure[#1][\c!type=eps,\c!methode=mps,#2]%
       \fi}
```

20
```
 \fi
```

This macro takes *two* arguments, the second one can be used to pass info to the inclusion macro.

Some examples of the use of this module can be found in the modules `supp-tpi` and `prag-log`.

21
```
\protect \endinput
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

4.8 TPIC Conversion

This modules implements the conversion of graphic TPIC specials using METAPOST.

We reimplement the TPIC specials using the special mimmicking mechanism implemented in the support module **supp-spe** as well as the METAPOST run–time support implemented in **supp-mps**.

1
```
\ifx\undefined\writestatus     \input supp-mis \relax \fi
\ifx\undefined\mimmickspecials \input supp-spe \relax \fi
\ifx\undefined\MPgraphic       \input supp-mps \relax \fi
```

2
```
\writestatus{loading}{Context Support Macros / TPIC Conversion}
```

Beware: we haven't activated both mechanism yet. This is to be done in the calling module.

3
```
\unprotect
```

When we want to mimmick TPIC specials in PDFTEX, we need to map its graphic primitives into PDF ones. The main problem in doing so is that PDF does not support b-splines directly and also does not offer us something to draw arcs. Of course all this scan be implemented in TEX, and the first implementation of this module did so, but the results were not that satisfying. Not having used these specials before, I had for instance to find out that the TPIC specials were not that unambiguesly defined.

Then, while discussing something else, Sebastian Ratz told me that the Web2c implementation that PDFTEX is base upon, offers some rather discutable, but nevertheless handy feature:

```
  \write18{execute program with arguments}
```

Knowing this, I immediatelly decided to throw away the old conversion macros and use the marvelous METAPOST, TEX related, drawing program to do the conversion in as high a quality as possible.

implementation we're going to present here, not only uses for drawing purposes, but also uses the more efficient METAPOST features to store the path.
Table 4.1 lists the TPIC specials as mentioned in the LATEX Graphics Companion and the relevant part of the DVIPS source. This list shows us that we have to store the path before we can use it, simply because we don't know in advance what actions to apply on it.

tag	**arguments**	**meaning**
pn	w	set linewidth
pa	$x\ y$	add point to path
fp		draw/fill path
ip		fill path
da	l	draw dashed path
dt	l	draw doted path
sp	d	draw spline
ar	$x\ y\ r_x\ r_y\ b\ e$	draw (partial) arc
ia	$x\ y\ r_x\ r_y\ b\ e$	fill (partial) arc
sh	s	fill next path

Table 4.1 The TPIC special syntax.

The first problem we have to take care of is the fact that there is no decent begin or end of the drawing process defined. We can however be quite sure that writers of packages using these specials will put them into a box, simply because else this is the most common used way to treat something TEX as as a whole, like:

```
\hbox{\special{}\special{}...}
```

We just start a picture as soon as the first special is encountered, so this becomes:

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
\hbox{\openpicture\newspecial{}\newspecial{}...
```

The first step in opening the picture is to start a group. Now we can savely use the egroup that closes the box to also end the picture.

4
```
\def\startTPICspecials%
  {\bgroup
   \let\startTPICspecials=\relax
   \aftergroup\stopTPICspecials
   \startwritingMPgraphic
   \writeMPgraphic{pair p[];}}
```

As soon as we begin a picture, we inhibit nesting by relaxing the start macro. The first METAPOST action we take is declaring an array of pairs named *p*.

Ending the picture is invoked by closing the current group. Because the TPIC picture comes out mirrored, we have to reflect the current METAPOST picture, stored in the system variable *currentpicture*, around the x-axis.

5
```
\def\stopTPICspecials%
  {\writeMPgraphic
     {currentpicture:=currentpicture reflectedabout ((0,0),(4095,0));}%
   \stopwritingMPgraphic
   \flushMPgraphics
   \loadcurrentMPgraphic{}%
   \setbox\MPgraphic=\hbox to \!!zeropoint
     {\kern-\wd\MPgraphic
      \vbox to \!!zeropoint{\box\MPgraphic\vss}\hss}%
   \ht\MPgraphic=\!!zeropoint
   \wd\MPgraphic=\!!zeropoint
   \dp\MPgraphic=\!!zeropoint
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
   \box\MPgraphic
   \egroup}
```

Here the macro `\stopwritingMPgraphic` has to take care of executing and including the METAPOST code.

We need to keep track of the number of elements that form the path. This is needed because we don't know in advance how the points are to be connected.

6
```
\newcount\TPICcounter
```

When a path is draw, we can connect the points using a smooth curve of drawing straight lines. A closed path can be drawn or filled.

7
```
\newif\ifTPICdraw
\newif\ifTPICfill
\newif\ifTPICcurve
```

The TPIC specials permit specifying the line and fill color as well as the linetype, which can be solid, dashed or dotted. We'll save those specifications as a METAPOST string, using:

8
```
\let\TPIClinetype =\empty
\let\TPICgrayscale=\empty
```

The magic reduction factor .07227 is needed to map the TPIC 1/1000 of an inch to POSTSCRIPT points. We cannot delegate this task to METAPOST because this program does not accept values greater than 4095.

I won't discuss all the specifics used in implementing the specials. The METAPOST part is rather trivial. Many specials have much in common, so the amout of code is not that large.

9
```
\redefinespecial pa \using#1 #2\endspecial
  {\startTPICspecials
```

```
    \bgroup
    \global\advance\TPICcounter by 1
    \dimen0=#1pt \dimen0=.07227\dimen0
    \dimen2=#2pt \dimen2=.07227\dimen2
    \writeMPgraphic{p[\the\TPICcounter]:=(\the\dimen0,\the\dimen2);}%
    \egroup}
```

10
```
\redefinespecial pn \using#1\endspecial
  {\startTPICspecials
    \bgroup
    \dimen0=#1pt \dimen0=.07227\dimen0
    \writeMPgraphic{pickup pencircle scaled \the\dimen0;}%
    \egroup}
```

11
```
\redefinespecial sh \using#1\endspecial
  {\startTPICspecials
    \bgroup
    \edef\g{#1}%
    \edef\g{\ifx\g\empty.5\else#1\fi}%
    \xdef\TPICgrayscale{withcolor (\g,\g,\g)}%
    \egroup}
```

12
```
\redefinespecial wh \using#1\endspecial
  {\mimmickspecial sh \using0\endspecial}
```

13
```
\redefinespecial bk \using#1\endspecial
  {\mimmickspecial sh \using1\endspecial}
```

14
```
\redefinespecial da \using#1\endspecial
  {\startTPICspecials
    \bgroup
    \edef\l{#1}%
    \ifx\l\empty
      \gdef\TPIClinetype{dashed evenly}%
    \else
      \dimen0=#1in
      \ifdim\dimen0<\!!zeropoint \dimen0=-\dimen0\fi
      \edef\f{\the\dimen0 \space}%
      \dimen0=.5\dimen0
      \edef\h{\the\dimen0 \space}%
      \xdef\TPIClinetype{dashed dashpattern (on \h off \f on \h)}%
    \fi
    \egroup
    \TPICcurvefalse\TPICdrawtrue
    \drawTPICpath\using#1\endspecial}
```

15
```
\redefinespecial dt \using#1\endspecial
  {\startTPICspecials
    \bgroup
    \edef\l{#1}%
    \xdef\TPIClinetype{dashed withdots \ifx\l\empty\else scaled #1in\fi}%
    \egroup
    \TPICcurvefalse\TPICdrawtrue
    \drawTPICpath\using#1\endspecial}
```

16
```
\redefinespecial fp \using#1\endspecial
  {\startTPICspecials
    \TPICcurvefalse\TPICdrawtrue
    \ifdim0#1pt=\!!zeropoint
      \drawTPICpath\using#1\endspecial
    \else\ifdim0#1pt<\!!zeropoint
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

```
     \mimmickspecial dt\using#1\endspecial
   \else
     \mimmickspecial da\using#1\endspecial
   \fi\fi}
```

17
```
\redefinespecial sp
  {\startTPICspecials\TPICdrawtrue\TPICcurvetrue\drawTPICpath}
```

18
```
\redefinespecial ip
  {\startTPICspecials\TPICfilltrue\drawTPICpath}
```

19
```
\redefinespecial ar
  {\startTPICspecials\TPICdrawtrue\drawTPICarc}
```

20
```
\redefinespecial ia
  {\startTPICspecials\TPICfilltrue\drawTPICarc}
```

These substitutes use two auxiliary macros that take care of actually drawing the shape or arc. Here we use the stored linetype (solid, dashed, dotted) and color (grayscale).

21
```
\def\drawTPICpath\using#1\endspecial
  {\bgroup
   \ifTPICdraw
     \def\TPICgrayscale{}%
   \fi
   \writeMPgraphic
     {\ifTPICfill fill\fi\ifTPICdraw draw\fi\space
      for i:=1 upto \the\TPICcounter-1:
        p[i]\ifTPICcurve..\else--\fi
      endfor
      p[\the\TPICcounter]
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
      \ifTPICfill\ifTPICcurve..\else--\fi cycle \fi
      \TPIClinetype\space\TPICgrayscale;}%
   \resetTPICvariables
   \egroup}
```

I have to admit that at the moment I wrote this macro, I could not write this piece of METAPOST. Fortunately Thortsen Ohl promptly answered the question I posted to the METAFONT discussion list.

22
```
\def\drawTPICarc\using#1 #2 #3 #4 #5 #6\endspecial
  {\bgroup
   \ifTPICdraw
     \def\TPICgrayscale{}%
   \fi
   \dimen 0=#1pt\dimen 0=.07227\dimen 0
   \dimen 2=#2pt\dimen 2=.07227\dimen 2
   \dimen10=#3pt\dimen10=.14454\dimen10
   \dimen12=#4pt\dimen12=.14454\dimen12
   \dimen20=#5pt
   \dimen22=#6pt
   \writeMPgraphic
     {\ifTPICfill fill\fi\ifTPICdraw draw\fi \space
      \ifTPICfill\else subpath 4/3.14159*(\the\dimen20,\the\dimen22) of \fi
      fullcircle xscaled \the\dimen10 \space yscaled \the\dimen12 \space
      shifted (\the\dimen0,\the\dimen2)
      \TPIClinetype \space \TPICgrayscale;}%
   \resetTPICvariables
   \egroup}
```

Resetting the variables need to be done globally because we cannot be sure if any further grouping is used by the envelopping macros.

23
```
\def\resetTPICvariables%
  {\global\TPICcounter=0
   \global\TPICfillfalse
   \global\TPICdrawfalse
   \global\let\TPIClinetype=\empty
   \global\let\TPICgrayscale=\empty}
```

I have to admit that by using the METAPOST Bézier cubics routines these implementation does produce better curves then most DVI drivers do using the TPIC prescribed b-splines. Take for instance the sequence:

```
   \special{pa 2000 1000}
   \special{pa 1000 2000}
   \special{pa 0000 1000}
   \special{pa 1000 0000}
   \special{pa 2000 1000}
   \special{sp}
```

One would expect that this code produced a closed circle, but the curve that comes out using b-splines is far from round. We can however savely asume that the arc producing specials will be used for drawing circle fragments, while the path specials will be used for arbitraty curves. And for b-splines to produce nice curves, one will often use many points to get the desired results. Therefore, using the METAPOST Bézier curves will certainly produce similar and even better graphics, except in those rare cases where one uses delinberately the not that accurate features of b-splines. Hereby the user is warned.

24
```
\protect \endinput
```

4.9 Files

TEX operates on files, so one wouldn't wonder that there is a separate module for file support. In CONTEXT files are used for several purposes:

- general textual input
- logging status information
- saving registers, lists and references
- buffering defered textual input

When dealing with files we can load them as a whole, using the `\input` primitive or load them on a line–by–line basis, using `\read`. Writing is always done line by line, using `\write`.

1
```
\writestatus{loading}{Context Support Macros / Files}
```

2
```
\unprotect
```

\pushendofline
\popendofline

When we are loading files in the middle of the typesetting process, for instance when we load references, we have to be sure that the reading process does not generate so called 'spurious spaces'. This can be prevented by assigning the line ending character the ⟨*catcode*⟩ comment. This is accomplished by

```
\pushendofline
... reading ...
\popendofline
```

Just to be sure, we save the current meaning of `^^M` in `\poppedendofline`.

3
```
\def\pushendofline
  {\chardef\poppedendofline=\the\catcode`\^^M\relax
   \catcode`\^^M=\@@comment\relax}
```

4
```
\def\popendofline
  {\catcode`\^^M=\poppedendofline}
```

\scratchread
\scratchwrite

We define a scratch file for reading. Keep in mind that the number of files is limited to 16, so use this one when possible. We also define a scratch output file.

5
```
\newread  \scratchread
\newwrite \scratchwrite
```

\processfile
\fileline

The next macro offers a framework for processing files on a line by line basis.

```
   \processfile \identifier {name} \action
```

The first argument can for instance be `\scratchread`. The action must do something with `\fileline`, which holds the current line.

6
```
\def\doprocessfile#1#2#3%
  {\openin#1=#2\relax
   \gdef\doprocessline%
     {\ifeof#1%
        \gdef\doprocessline{\closein#1}%
      \else
        \global\read#1 to \fileline
        #3\relax
      \fi
      \doprocessline}%
   \doprocessline}
```

supp-mis supp-ver supp-vis supp-lan supp-pdf supp-spe supp-mps supp-tpi supp-fil supp-ini supp-box supp-mrk supp-mul supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

\readfile
\ReadFile
\maxreadlevel
\normalinput

One cannot be sure if a file exists. When no file can be found, the `\input` primitive gives an error message and switches to interactive mode. The macro `\readfile` takes care of non–existing files. This macro has two faces.

```
\ReadFile {filename}
\readfile {filename} {before loading} {not found}
```

Many TEX implementations have laid out some strategy for locating files. This can lead to unexpected results, especially when one loads files that are not found in the current directory. Let's give an example of this. In CONTEXT illustrations can be defined in an external file. The resizing macro first looks if an illustration is defined in the local definitions file. When no such file is found, it searches for a global file and when this file is not found either, the illustration itself is scanned for dimensions. One can imagine what happens if an adapted, localy stored illustration, is scaled according to dimensions stored somewhere else.

When some TEX implementation starts looking for a file, it normally first looks in the current directory. When no file is found, TEX starts searching on the path where format and/or style files are stored. Depending on the implementation this can considerably slow down processing speed.

In CONTEXT, we support a project–wise ordening of files. In such an approach it seems feasible to store common files in a lower directory. When for instance searching for a general layout file, we therefore have to backtrack.

These three considerations have lead to a more advanced approach for loading files.

We first present an earlier implementation of `\readfile`. This command backtracks parent directories, upto a predefined level. Users can change this level, but we default to 3.

```
\def\maxreadlevel {3}
```

This is a pseudo ⟨*counter*⟩.

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

We use `\normalinput` instead of `\input` because we want to be able to redefine the original `\input` when needed, for instance when loading third party libraries.

7
```
\let\normalinput=\input
```

8
```
\def\maxreadlevel {3}
```

9
```
\def\doreadfile#1#2#3%
  {\immediate\openin\scratchread=#1\relax
   \ifeof\scratchread
     \immediate\closein\scratchread
     \decrement\readlevel
     \ifnum\readlevel>0\relax
       \doreadfile{\f!parentpath/#1}{#2}{#3}%
     \else
       #3%
     \fi
   \else
     \immediate\closein\scratchread
     #2%
     \normalinput #1\relax
   \fi}
```

10
```
\def\readfile#1%
  {\let\readlevel=\maxreadlevel
   \doreadfile{#1}}
```

11
```
\def\ReadFile#1%
  {\readfile{#1}{}{}}
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

\readjobfile
\readlocfile
\readsysfile
\readfixfile

This implementation honnors the third situation, but we still can get unwanted files loaded and/or can get involved in extensive searching.

Due to different needs, we decided to offer four alternative loading commands. With `\readjobfile` we load a local file and do no backtracking, while `\readlocfile` backtracks 3 directories, including the current one.

12
```
\def\readjobfile#1%
  {\newcounter\readlevel
   \doreadfile{\f!currentpath/#1}}
```

13
```
\def\readlocfile#1%
  {\let\readlevel=\maxreadlevel
   \doreadfile{\f!currentpath/#1}}
```

System files can be anywhere and therefore `\readsysfile` is not bound to the current directory and obeys the TeX implementation.

14
```
\def\readsysfile#1%
  {\let\readlevel=\maxreadlevel
   \doreadfile{#1}}
```

The last one, `\readfixfile` searches on the directory specified and backtracks too.

15
```
\def\readfixfile#1#2%
  {\let\readlevel=\maxreadlevel
   \doreadfile{#1/#2}}
```

After having defined this commands, we reconsidered the previously defined `\readfile`. This time we more or less impose the search order.

supp-mis supp-ver supp-vis supp-lan supp-pdf supp-spe supp-mps supp-tpi supp-fil supp-ini supp-box supp-mrk supp-mul supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

16
```
\def\readfile#1#2#3%
  {\readlocfile{#1}{#2}
     {\readjobfile{#1}{#2}
        {\readsysfile{#1}{#2}{#3}}}}
```

So now we've got ourselves five file loading commands:

```
\readfile              {filename} {before loading} {not found}

\readjobfile           {filename} {before loading} {not found}
\readlocfile           {filename} {before loading} {not found}
\readfixfile           {filename} {before loading} {not found}
\readsysfile {directory} {filename} {before loading} {not found}
```

\readjobfile
\readlocfile
\readsysfile
\readfixfile

The next four alternatives can be used for opening files for reading on a line–by–line basis. These commands get an extra argument, the filetag. Explicit closing is done in the normal way by `\closein`.

17
```
\def\doopenin#1#2%
  {\increment\readlevel
   \immediate\openin#1=#2\relax
   \ifeof#1\relax
     \ifnum\readlevel>\maxreadlevel\relax
     \else
       \immediate\closein#1\relax
       \doopenin{#1}{\f!parentpath/#2}%
     \fi
   \fi}
```

18
```
\def\openjobin#1#2%
  {\newcounter\readlevel
   \doopenin{#1}{\f!currentpath/#2}}
```

19
```
\def\opensysin#1#2%
  {\let\readlevel=\maxreadlevel
   \doopenin{#1}{#2}}
```

20
```
\def\openlocin#1#2%
  {\let\readlevel=\maxreadlevel
   \doopenin{#1}{\f!currentpath/#2}}
```

21
```
\def\openfixin#1#2#3%
  {\let\readlevel=\maxreadlevel
   \doopenin{#1}{#2/#3}}
```

\doiffileelse
\doiflocfileelse

The next alternative only looks if a file is present. No loading is done. This one obeys the standard TEX implementation method.

```
   \doiffileelse {filename} {before loading} {not found}
```

We use `\next` here, because we want to close the file first. We also provide the local alternative:

```
   \doiflocfileelse {filename} {before loading} {not found}
```

22
```
\def\doiffileelse#1#2#3%
  {\immediate\openin\scratchread=#1\relax
   \ifeof\scratchread
     \def\next{#3}%
   \else
     \def\next{#2}%
   \fi
   \immediate\closein\scratchread
   \next}
```

23
```
\def\doiflocfileelse#1%
  {\doiffileelse{\f!currentpath/#1}}
```

\doinputonce

Especially macropackages need only be loaded once. Repetitive loading not only costs time, relocating registers often leads to abortion of the processing because TEX's capacity is limited. One can prevent multiple loading by using:

```
\doloadonce{filename}
```

This command obeys the standard method for locating files.

24
```
\def\doinputonce#1%
  {\doifundefined{@@@#1@@@}%
     {\setgvalue{@@@#1@@@}{}%
      \doiffileelse{#1}{\normalinput #1}{}}}
```

\doifparentfileelse

The test `\doifelse{\jobname}{filename}` does not give the desired result, simply because `\jobname` expands to characters with ⟨*catcode*⟩ 12, while the characters in `filename` have ⟨*catcode*⟩ 11. So we can better use:

```
\doifparentfileelse{filename}{yes}{no}
```

25
```
\def\doifparentfileelse#1#2#3%
  {\edef\!!stringa{#1}%
   \@EA\convertargument\!!stringa\to\!!stringa
   \@EA\def\@EA\!!stringb\@EA{\jobname}%
   \ifx\!!stringa\!!stringb#2\else#3\fi}
```

26
```
\protect
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

4.10 Initializations

```
\writestatus{loading}{Context Support Macros / Initializations}

\newif\ifeightbitcharacters  \eightbitcharactersfalse

\endinput
```

4.11 Boxes

This module implements some box manipulation macros. Some are quite simple, some are more advanced and when understood well, all can be of use.

```
\writestatus{loading}{Context Support Macros / Boxes}
```

```
\unprotect
```

\nextdepth

Let's start with a rather simple declaration. Sometimes we need to save the TeX ⟨*dimension*⟩ `\prevdepth` and append it later on. The name `\nextdepth` suits this purpose well.

```
\newdimen\nextdepth
```

\smashbox

Smashing is introduced in PLAIN TeX, and stands for reducing the dimensions of a box to zero. The most resolute one is presented first.

```
\def\smashbox#1%
  {\wd#1=\!!zeropoint
   \ht#1=\!!zeropoint
   \dp#1=\!!zeropoint}
```

\hsmashbox
\vsmashbox

Smashing can be used for overlaying boxes. Depending on the mode, horizontal or vertical, one can use:

```
\def\hsmashbox#1%
  {\wd#1=\!!zeropoint}
```

```
\def\vsmashbox#1%
  {\ht#1=\!!zeropoint
   \dp#1=\!!zeropoint}
```

\hsmash
\vsmash
\hsmashed
\vsmashed

While the previous macros expected a ⟨*box*⟩, the next act on a content. They are some subtle differences betreen the smash and smashed alternatives. The later ones reduce all dimensions to zero.

7
```
\def\hsmash#1%
  {\bgroup
   \setbox0=\normalhbox{#1}%
   \hsmashbox0%
   \box0
   \egroup}
```

8
```
\def\vsmash#1%
  {\bgroup
   \setbox0=\normalvbox{#1}%
   \vsmashbox0%
   \box0
   \egroup}
```

9
```
\def\hsmashed#1%
  {\bgroup
   \setbox0=\normalhbox{#1}%
   \smashbox0%
   \box0
   \egroup}
```

10
```
\def\vsmashed#1%
  {\bgroup
   \setbox0=\normalvbox{#1}%
   \smashbox0%
   \box0
   \egroup}
```

\getboxheight

Although often needed, TEX does not support arithmics like:

```
\dimen0 = \ht0 + \dp0
```

so we implemented:

```
\getboxheight ... \of \box...
```

For instance,

```
\getboxheight \dimen0 \of \box0
\getboxheight \someheight \of \box \tempbox
```

11
```
\def\getboxheight#1\of#2\box#3%
  {#1=\ht#3%
   \advance#1 by \dp#3\relax}
```

\dowithnextbox
\nextbox

Sometimes we want a macro to grab a box and do something on the content. One could pass an argument to a box, but this can violate the specific ⟨*catcodes*⟩ of its content and leads to unexpected results. The next macro treats the following braced text as the content of a box and manipulates it afterwards in a predefined way.

The first argument specifies what to do with the content. This content is available in `\nextbox`. The second argument is one of `\hbox`, `\vbox` or `\vtop`. The third argument must be grouped with `\bgroup` and `\egroup`, `{...}` or can be a `\box` specification.

In CONTEXT this macro is used for picking up a box and treating it according to earlier specifications. We use for instance something like:

```
\def\getfloat%
  {\def\handlefloat{...\box\nextbox...}
   \dowithnextbox\handlefloat\vbox}
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

in stead of:

```
\def\getfloat#1%
  {...#1...}
```

In this implementation the `\aftergroup` construction is needed because `\afterassignment` is executed inside the box.

12
```
\newbox\nextbox
```

13
```
\def\dowithnextbox#1%
  {\def\dodowithnextbox{#1}%
   \afterassignment\dododowithnextbox
   \setbox\nextbox}
```

14
```
\def\dododowithnextbox%
  {\aftergroup\dodowithnextbox}
```

So in fact we get:

```
\setbox\nextbox { \aftergroup\dodowithnextbox ... }
```

or

```
\setbox\nextbox { ... } \dodowithnextbox
```

\beginofshapebox
\reshapebox
\flushshapebox
\shapebox
\ifreshapingbox

The next utility macro originates from some linenumbering mechanism. Due to TEX's advanced way of typesetting paragraphs, it's not easy to do things on a line–by–line basis. This macro is able to reprocess a given box and can act upon its vertical boxed components, such as lines. The unwinding sequence in this macro is inspired by a NTG workshop of David Salomon in June 1992.

First we have to grab the piece of text we want to act upon. This is done by means of the duo macros:

```
\beginofshapebox
a piece of text
\endofshapebox
```

When all texts is collected, we can call `\reshapebox` and do something with it's vertical components. We can make as much passes as needed. When we're done, the box can be unloaded with `\flushshapebox`. The only condition in this scheme is that `\reshapebox` must somehow unload the ⟨*box*⟩ `\shapebox`.

An important aspect is that the content is unrolled bottom–up. The next example illustrates this maybe unexpected characteristic.

```
\beginofshapebox
\em \input tufte
\endofshapebox

\newcounter\LineNumber

\reshapebox
  {\doglobal\increment\LineNumber
   \hbox{\llap{\LineNumber\hskip2em}\box\shapebox}}

\flushshapebox
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

7 *We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit,*
6 *single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense,*
5 *reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate,*
4 *discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump,*
3 *skip, smooth, chunk, avarage, approximate, cluster, aggregate, outline, summarize, itemize, review,*
2 *dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize,*
1 *winnow the wheat from the chaff and seperate the sheep from the goats.*

As we can see, when some kind of numbering is done, we have to add a second pass.

```
\newcounter\LineNumber
\newcounter\NumberOfLines

\reshapebox
  {\doglobal\increment\NumberOfLines
   \box\shapebox}

\reshapebox
  {\doglobal\increment\LineNumber
   \hbox
     {\llap{\LineNumber\ (\NumberOfLines)\hskip2em}%
      \box\shapebox}%
   \doglobal\decrement\NumberOfLines}

\flushshapebox
```

7 (1) *We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit,*
6 (2) *single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense,*
5 (3) *reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate,*

4 (4) *discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump,*
3 (5) *skip, smooth, chunk, avarage, approximate, cluster, aggregate, outline, summarize, itemize, review,*
2 (6) *dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize,*
1 (7) *winnow the wheat from the chaff and seperate the sheep from the goats.*

This example shows that the content of the box is still available after flushing. Another feature is that only the last reshaping counts. Multiple reshaping can be done by:

```
\beginofshapebox
\flushshapebox
\endofshapebox

\reshapebox
  {\doglobal\increment\LineNumber
   \hbox{\llap{$\star$\hskip1em}\box\shapebox}%
   \doglobal\decrement\NumberOfLines}

\flushshapebox
```

7 (1) ⋆ *We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit,*
6 (2) ⋆ *single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense,*
5 (3) ⋆ *reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate,*
4 (4) ⋆ *discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump,*
3 (5) ⋆ *skip, smooth, chunk, avarage, approximate, cluster, aggregate, outline, summarize, itemize, review,*
2 (6) ⋆ *dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize,*
1 (7) ⋆ *winnow the wheat from the chaff and seperate the sheep from the goats.*

The macros are surprisingly easy to follow and in fact introduce no new concepts. Nearly all books on TEX show similar solutions for unwinding ⟨*boxes*⟩.

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

Some macros, like footnote ones, can be sensitive for reshaping, which can result in an endless loop. We therefore offer:

```
\ifreshapingbox
```

Some CONTEXT commands are protected this way. Anyhow, reshaping is aborted after 100 dead cycles.

By the way, changing the height and depth of ⟨*box*⟩ `\shapebox` results in bad spacing. This means that for instance linenumbers etc. should be given zero height and depth before being lapped into the margin. The previous examples ignore this side effect, but beware!

15
```
\newif\ifsomeshapeleft
\newif\ifreshapingbox
```

16
```
\def\shapesignal {.12345678pt}
```

17
```
\newbox\shapebox
\newbox\newshapebox
\newbox\oldshapebox
```

18
```
\newcount\shapecounter
```

19
```
\def\reshapebox#1%
  {\setbox\newshapebox=\normalvbox
     \bgroup
       \unvcopy\oldshapebox
       \setbox\newshapebox=\box\voidb@x
       \shapecounter=0
       \loop
         \someshapelefttrue
         \ifdim\lastskip=\!!zeropoint\relax
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
      \ifdim\lastkern=\!!zeropoint\relax
        \ifnum\lastpenalty=0
          \setbox\shapebox=\lastbox
          \ifvoid\shapebox
            \unskip\unpenalty\unkern
          \else
            \ifdim\wd\shapebox=\shapesignal\relax
              \someshapeleftfalse
            \else
              \shapecounter=0
              \setbox\newshapebox=
                \normalvbox{#1\unvbox\newshapebox}
            \fi
          \fi
        \else
          \scratchcounter=\lastpenalty
          \setbox\newshapebox=
            \normalvbox{\penalty\scratchcounter \unvbox\newshapebox}
          \unpenalty
        \fi
      \else
        \dimen0=\lastkern
        \setbox\newshapebox=
          \normalvbox{\kern\dimen0 \unvbox\newshapebox}
        \unkern
      \fi
    \else
      \skip0=\lastskip
      \setbox\newshapebox=
          \normalvbox{\vskip\skip0 \unvbox\newshapebox}
        \unskip
      \fi
    \ifnum\shapecounter>100
      \message{<<forced exit from shapebox>>}%
      \someshapeleftfalse
    \else
      \advance\shapecounter by 1
    \fi
    \ifsomeshapeleft \repeat
    \unvbox\newshapebox
  \egroup}
```

20
```
\def\beginofshapebox%
  {\setbox\oldshapebox=\normalvbox
     \bgroup
     \reshapingboxtrue
     \hbox to \shapesignal{\hss}}
```

21
```
\def\endofshapebox%
  {\endgraf
   \egroup}
```

22
```
\def\flushshapebox%
  {\ifdim\ht\newshapebox=\!!zeropoint\relax
   \else
     % make \prevdepth legal
     \par
     % and take a look
     \ifdim\prevdepth=\hideskip\relax
```

```
      \prevdepth=\!!zeropoint
    \fi
    \ifdim\prevdepth<\!!zeropoint
      % something like a line or a signal or ...
    \else
      \ifinner
        % not watertight and not ok
      \else\ifdim\pagegoal=\maxdimen\else
        % give the previous line a normal depth
        \vbox to \!!zeropoint{}
        % go back one line
        \vskip-\lineheight
      \fi\fi
    \fi
    \unvcopy\newshapebox\relax
    % \prevdepth=0pt and \dp\newshapebox depend on last line
    \kern-\dp\newshapebox\relax
    % now \prevdepth=0pt
  \fi}
```

\hyphenatedword
\dohyphenateword

The next one is a tricky one. PLAIN TEX provides `\showhyphens` for showing macros on the terminal. When preparing a long list of words we decided to show the hyphens, but had to find out that the PLAIN alternative can hardly be used and/or adapted to typesetting. The next two macros do the job and a little more.

The simple command `\hyphenatedword` accepts one argument and gives the hyphenated word. This macro calls for

```
  \dohyphenateword {n} {pre} {word}
```

The next examples tell more than lots of words:

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
\dohyphenateword{0} {}    {dohyphenatedword}
\dohyphenateword{1} {...} {dohyphenatedword}
\dohyphenateword{2} {...} {dohyphenatedword}
```

Here, `\hyphenatedword{dohyphenatedword}` is the shorter alternative for the first line.
do-hy-phen-at-ed-word
...hy-phen-at-ed-word
...phen-at-ed-word

These macros are slow but effective and not that hard to program at all.

23
```
\def\dohyphenateword#1#2#3%
  {\bgroup
   \setbox0=\hbox
     {\mindermeldingen
      \widowpenalty=0
      \clubpenalty=0
      \setbox0=\vbox
        {\hsize\!!zeropoint \ #3}%
      \ifnum#1>0
        \dorecurse{#1}
          {\setbox2=\hbox
             {\vsplit0 to \baselineskip}}%
        #2%
      \fi
      \loop
        \setbox2=\hbox
          {\vsplit0 to \baselineskip}%
        \hbox
          {\unhbox2
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
          \setbox2=\lastbox
          \vbox
            {\unvbox2
             \setbox2=\lastbox
             \hbox{\unhbox2}}}%
        \ifdim\ht0>\!!zeropoint
      \repeat}%
    \ht0=\ht\strutbox
    \dp0=\dp\strutbox
    \box0
    \egroup}
```

24

```
\def\hyphenatedword%
  {\dohyphenateword{0}{}}
```

\doboundtext

Sometimes there is not enough room to show the complete (line of) text. In such a situation we can strip of some characters by using `\doboundtext`. When the text is wider than the given width, it's split and the third argument is appended. When the text to be checked is packed in a command, we'll have to use `\expandafter`.

```
   \doboundtext{a very, probably to long, text}{3cm}{...}
```

When calculating the room needed, we take the width of the third argument into account, which leads to a bit more complex macro than needed at first sight.

25

```
\def\dodoboundtext#1%
  {\setbox0=\hbox{\unhcopy0 #1}%
   \ifdim\wd0>\dimen0
     \let\dodoboundtext=\gobbleoneargument
   \else
     #1\relax
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

```
   \fi}
```

26
```
\def\doboundtext#1#2#3%
  {\hbox
     {\setbox0=\hbox{#1}%
      \dimen0=#2\relax
      \ifdim\wd0>\dimen0
        \setbox2=\hbox{#3}%
        \advance\dimen0 by -\wd2
        \setbox0=\hbox{}%
        \processtokens
          {\dodoboundtext}
          {\dodoboundtext}
          {}
          {\space}
          {#1}%
        \box2
      \else
        \box0
      \fi}}
```

\limitatetext

A bit more beautiful alternative for the previous command is the next one. This command is more robust because we let TEX do most of the job. The previous command works better on text that cannot be hyphenated.

```
   \limitatetext {text} {width} {sentinel}
```

When no width is given, the whole text comes available. The sentinel is optional.

27
```
\def\limitatetext#1#2#3%
  {\doifelse{#2}{}
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
  {#1}
  {\bgroup
   \setbox0=\hbox{#1}%
   \dimen0=#2\relax
   \ifdim\wd0>\dimen0
     \setbox2=\hbox{\ #3}%
     \advance\dimen0 by -\wd2
     \setbox0=\vbox
       {\hsize=\dimen0\relax
        \hfuzz\maxdimen
        \raggedright
        \strut\unhbox0}%
        \vbox % if omitted: missing brace reported
          {\setbox0=\vsplit0 to \ht\strutbox
           \unvbox0
           \setbox0=\lastbox
           \unhbox0\kern0pt\box2}%
   \else
     \unhbox0
   \fi
   \egroup}}
```

\processisolatedwords

References are often made up of one word or a combination of tightly connected words. The typeset text **chapter 5** is for instance the results of the character sequence:

```
The typeset text \in{chapter}[texniques] is for instance
```

When such words are made active in interactive texts, the combination cannot longer be hyphenated. Normally this is no problem, because TEX tries to prevent hyphenation as best as can.

Sometimes however we need a few more words to make things clear, like when we want to refer to **TEX by Topic**. The macros that are responsible for typesetting hyperlinks, take care of such sub–sentences by breaking them up in words. Long ago we processed words using the space as a separator, but the more advanced our interactive text became, the more we needed a robust solution. Well, here it is and it called as:

```
\processisolatedwords{some words}\someaction
```

The second argument `someactions` handles the individual words, like in:

```
\processisolatedwords{some more words}           \ruledhbox \par
\processisolatedwords{and some $x + y = z$ math} \ruledhbox \par
\processisolatedwords{and a \hbox{$x + y = z$}}  \ruledhbox \par
```

which let the words turn up as:

some more words
and some $x + y = z$ math
and a $x + y = z$

The macro has been made a bit more clever than needed at first sight. This is due to the fact that we don't want to generate more overhead in terms of interactive commands than needed.

```
\processisolatedwords{see this \ruledhskip1em}     \ruledhbox
\processisolatedwords{and \ruledhskip1em this one} \ruledhbox
```

becomes:
see this and this one

Single word arguments are treated without further processing. This was needed because this command is used in the `\goto` command, to which we sometimes pass very strange and/or complicated arguments or simply boxes whose dimensions are to be left intact.

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

First we build a \hbox. This enables us to save the last skip. Next we fill a \vbox without hyphenating words. After we've tested if there is more than one word, we start processing the individual lines (words). We need some splitting, packing and unpacking to get the spacing and dimensions right.

28
```
\def\processisolatedwords#1#2%
  {\bgroup
   \mindermeldingen
   \forgetall
   \setbox0=\hbox
     {#1%
      \xdef\isolatedlastskip{\the\lastskip}}%
   \setbox2=\vbox
     {\hyphenpenalty10000
      \hsize\!!zeropoint
      \unhcopy0}% == #1
   \ifdim\ht0=\ht2
      #2{\unhcopy0}% == #2{#1}
   \else
     \setbox0=\hbox
       {\ignorespaces
        \loop
          \setbox4=\hbox
            {\vsplit2 to \baselineskip}%
          \hbox
            {\unhbox4
             \setbox4=\lastbox
             \vbox
               {\unvbox4
                \setbox4=\lastbox
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
                #2{\hbox{\unhbox4}}}}%
          \hskip\fontdimen2\font
            \!!plus \fontdimen3\font
            \!!minus \fontdimen4\font
        \ifdim\ht2>\!!zeropoint \repeat
        \unskip}%
      \unhbox0\unskip\hskip\isolatedlastskip
    \fi
    \egroup}
```

`\sbox` This is a rather strange command. It grabs some box content and and limits the size to the height and depth of a `\strut`. The resulting bottom–alligned box can be used aside other ones, without disturbing the normal baseline distance.

```
   \ruledhbox to .5\hsize{\sbox{eerste\par tweede \par derde}}
```

Shows up as:
eerste

tweede

derde

Before displaying the result we added some skip, otherwise the first two lines would have ended up in the text. This macro can be useful when building complicated menus, headers and footers and/or margin material.

This macro still needs some improvement.

29
```
\def\sbox%  in handleiding, voorbeeld \inlinker{xx} \extern..
  {\dowithnextbox
     {\setbox0=\hbox
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

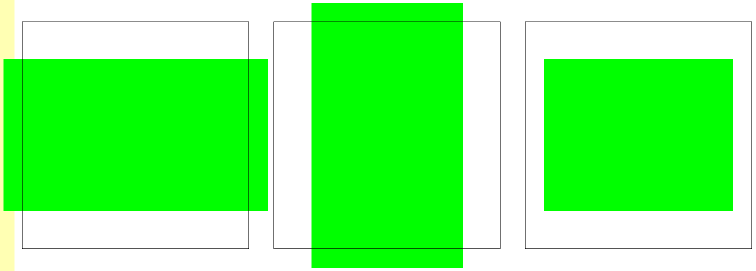
```
          {\strut
           \dp\nextbox=0pt
           \lower\strutdepth\box\nextbox}%
        \dp0=\strutdepth
        \ht0=\strutheight
        \box0}%
      \vbox}
```

\centeredbox

Here is another strange one. This one offers a sort of overlay with positive or negative offsets. This command can be used in well defined areas where no offset options are available. We first used it when building a button inside the margin footer, where the button should have a horizontal offset and should be centered with respect to the surrounding box. The last of the three examples we show below says:

```
   \vsize=3cm
   \hsize=3cm
   \ruledvbox to \vsize
     {\centeredbox height .5cm width -1cm
        {\vrule width \hsize height \vsize}}}
```

Here the `\ruledvbox` just shows the surrounding box and `\vrule` is used to show the centered box.



supp-box

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

This command takes two optional arguments: `width` and `height`. Observing readers can see that we use TEX's own scanner for grabbing these arguments: `#1#` reads everyting till the next brace and passes it to both rules. The setting of the box dimensions at the end is needed for special cases. The dimensions of the surounding box are kept intact. This commands handles positive and negative dimensions (which is why we need two boxes with rules).

```
\def\centeredbox#1#%   height +/-dimen width +/-dimen
  {\bgroup
   \setbox0=\vbox to \vsize
     \bgroup
       \mindermeldingen
       \forgetall
       \setbox0=\hbox{\vrule\!!width\!!zeropoint#1}%
       \setbox2=\vbox{\hrule\!!height\!!zeropoint#1}%
       \advance\vsize by \ht2
       \advance\hsize by \wd0
       \vbox to \vsize
         \bgroup
           \vskip-\ht2
           \vss
           \hbox to \hsize
             \bgroup
               \dowithnextbox
                 {\hskip-\wd0
                  \hss
                  \box\nextbox
                  \hss
             \egroup
           \vss
         \egroup
```

supp-mis supp-ver supp-vis supp-lan supp-pdf supp-spe supp-mps supp-tpi supp-fil supp-ini supp-box supp-mrk supp-mul supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
    \egroup
    \wd0=\hsize
    \ht0=\vsize
    \box0
    \egroup}
  \hbox}
```

\startruledboxcorrection
\ruledboxcorrection

Spacing around ruled boxes can get pretty messed up. The next macro tries as good as possible to fix this.

Rule Brittanica

We also take page breaks into account. One can assing additional spacing commands to the macro

```
  \ruledboxcorrection
```

31
```
\let\ruledboxcorrection=\relax
```

32
```
\def\startruledboxcorrection%
  {\bgroup
   \setbox0=\vbox\bgroup
   \ignorespaces}
```

33
```
\def\stopruledboxcorrection%
  {\egroup
   \endgraf
   \ifdim\pagegoal<\maxdimen
     \dimen0=2\dp\strutbox
     \ifdim\prevdepth>\!!zeropoint\relax
       \advance\dimen0 by -\prevdepth
     \fi
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
    \else
      \dimen0=\!!zeropoint % alternative: \dp\strutbox
    \fi
    \advance\dimen0 by \ht0
    \dimen2=\dp\strutbox
    \advance\dimen2 by \lineskip
    \ruledboxcorrection
    \noindent
    \vrule\!!height\dimen0\!!depth\dimen2\!!width\!!zeropoint
    \box0\relax
    \ruledboxcorrection
    \egroup}
```

34

```
\protect
```

4.12 Marks

There are 256 ⟨*counters*⟩, ⟨*dimensions*⟩, ⟨*skips*⟩, ⟨*muskips*⟩ and ⟨*boxes*⟩, 16 in- and output buffers, but there is only one ⟨*mark*⟩. In TugBoat 8 (1987, no 1) Jim Fox presents a set of macros that can be used to mimmick multiple marks. We gladly adopt them here.

```
\writestatus{loading}{Context Support Macros / Marks}
```

```
\unprotect
```

This implementation is more or less compatible with the other register macros in PLAIN TEX. A mark is defined by:

```
\newmark\name
```

and can be called upon with:

```
\topname
\botname
\firstname
```

The only drawback of his approach is that the marks must be preloaded in the output routine. This is accomplished by means of:

```
\getmarks\name
```

The macros presented here are in most aspects copies of those presented by Jim Fox. We've taken the freedom to change a few things for more or less obvious reasons:

- Because the original macros look quite complicated, which is mainly due to extensive use of `\expandafter`'s and `\csname`'s, we changed those in favor of `\getvalue`.
- To be more in line with the rest of CONTEXT, we've changed some of the names of macros.

supp-mis supp-ver supp-vis supp-lan supp-pdf supp-spe supp-mps supp-tpi supp-fil supp-ini supp-box supp-mrk supp-mul supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

- Because we are already short on ⟨*counters*⟩ we use macros when possible.
- We maintain a list of defined marks and use one call for getting them all at once.
- We have extended the mechanism to splitmarks (not perfected yet).
- We've introduced optional expansion of the contents of marks.

Whatever changes we've made, the credits still go to Jim, whatever goes wrong is due to me. The method is described in the TugBoat mentioned before, so we won't go into details. All marks belonging to a group are packed in a list. In this list they are preceded by a macro that can be defined at will and a number concerning the position at which it was defined.

```
\def\somelist{... \domark5{this} ... \domark31{that} ...}
```

The original `\mark` keeps track of the number and `\topmark` and `\botmark` are used to extract the actual marks from the list. The counting is done by

```
\currentmarker
```

In CONTEXT we use the mark mechanism to keep track of colors. In a complicated documents with many colors per page, `\currentmarker` can therefore get pretty high. (Well, this is not completely true, because we don't always have to use marks.)

3
```
\newcount\currentmarker
```

The original implementation used a few more ⟨*counters*⟩. Two have been substituted by macros, one has been replaced by our scratch counter.

```
% \newcount\topmarker
% \newcount\botmarker
% \newcount\foundmarker
```

We've also introduced some constants, one for the lists and three for composing the mark commands.

4
```
\def\@@marklist@@  {marklist}
\def\@@marktop@@   {top}
\def\@@markbot@@   {bot}
\def\@@markfirst@@ {first}
```

The next one is new too. All defined marks are packed in a comma seperated list. This could of course have been a token list but CONTEXT has some preference for comma lists.

5
```
\def\markers {}
```

\expandmarks

There are two booleans. The first one handles the first marks, the second concerns expansion. This second one is new.

6
```
\newif\ifnofirstmarker
\newif\ifexpandmarks     \expandmarkstrue
```

We use an indirect call to the mack mechanism.

7
```
\let\normalmark             = \mark
\let\normaltopmark          = \topmark
\let\normalbotmark          = \botmark
\let\normalfirstmark        = \firstmark
\let\normalsplitbotmark     = \splitbotmark
\let\normalsplitfirstmark = \splitfirstmark
```

The next macro replaces the multiple step expansion and command name constructors of Jim. This alternative leads to a more readable source (we hope).

8
```
\def\makemarknames#1%
  {\bgroup
```

```
    \escapechar=-1
    \xdef\markname{\string#1}%
    \xdef\marklist{\@@marklist@@\string#1}%
    \egroup}
```

\newmark A mark is defined by **\newmark**. At the same time, the name of the mark is added to a commalist. The three initializations were not in the original design, but make calls from outside the output routine a bit more robust.

9
```
\def\newmark#1%
  {\bgroup
    \makemarknames{#1}%
    \doglobal\addtocommalist{\markname}\markers%
    \long\setgvalue{\@@marktop@@\markname}{}%
    \long\setgvalue{\@@markfirst@@\markname}{}%
    \long\setgvalue{\@@markbot@@\markname}{}%
    \setgvalue{\marklist}{\domark0{}}%
    \long\gdef#1{\addmarker#1}%
    \egroup}
```

Setting a new mark and adding a mark to the designated list is done by **\addmarker**. This is an internal command, the user set a marks bij calling it's name:

```
    \mymark{some text}
```

Where **\mymark** is previously defined by **\newmark**.

10
```
\long\def\addmarker#1#2%
  {\bgroup
    \makemarknames{#1}%
    \global\advance\currentmarker by 1\relax
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

```
    \normalmark{\the\currentmarker}%
    \@EA\!!toksa\@EA=\@EA\@EA\@EA{\csname\marklist\endcsname}%
    \ifexpandmarks
      \setxvalue{\marklist}%
        {\the\!!toksa
         \noexpand\domark
         \the\currentmarker{#2}}%
    \else
      \!!toksb=\@EA{#2}%
      \setxvalue{\marklist}%
        {\the\!!toksa
         \noexpand\domark
         \the\currentmarker{\the\!!toksb}}%
    \fi
    \egroup}
```

\getmarks
\getallmarks
\getsplitmarks
\getallsplitmarks

In fact, marks make only sense in the output routine. Marks are derived from their list by means of **\getmarks**. Only one call per mark is permitted in the output routine. Therefore, it's far more easy to get them all at once, by means of **\getallmarks**, which is not part of the original design.

This grabbing is done by processing the list using the embedded **\domark** macros. When a relevant mark is found, this macro is reassigned and from then on serves in building the new list.

11

```
\def\getmarks#1%
  {\bgroup
    \makemarknames{#1}%
    \edef\topmarker{0\normaltopmark}%
    \edef\botmarker{0\normalbotmark}%
    \!!toksb={}%
    \nofirstmarkertrue
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
   \let\@fi=\fi     \let\fi=\relax
   \let\@or=\or     \let\or=\relax
   \let\@else=\else \let\else=\relax
   \let\domark=\doscanmarks
   \getvalue{\marklist}\lastmark
   %\message{markstatus : [\the\!!toksa\the\!!toksb\the\!!toksc]}%
   \long\setxvalue{\marklist}{\the\!!toksa\the\!!toksb\the\!!toksc}%
   \egroup}
```

12
```
\def\getallmarks%
  {\processcommacommand[\markers]\getmarks}
```

13
```
\def\getsplitmarks#1%
  {\bgroup
   \makemarknames{#1}%
   \@EA\let\@EA\savedmarklist\@EA=\csname\marklist\endcsname
   \edef\topmarker{0\normalsplitfirstmark}%
   \edef\botmarker{0\normalsplitbotmark}%
   \!!toksb={}%
   \nofirstmarkertrue
   \let\@fi=\fi     \let\fi=\relax
   \let\@or=\or     \let\or=\relax
   \let\@else=\else \let\else=\relax
   \let\domark=\doscanmarks
   \getvalue{\marklist}\lastmark
   \@EA\global\@EA\let\csname\marklist\endcsname=\savedmarklist
   \egroup}
```

14
```
\def\getallsplitmarks%
  {\processcommacommand[\markers]\getsplitmarks}
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

15
```
\long\def\dodoscanmarks#1%
  {\ifnum\scratchcounter>\topmarker\relax
   \@else
     \long\setgvalue{\@@marktop@@\markname}{#1}%
   \@fi
   \ifnum\scratchcounter>\botmarker\relax
     \let\domark=\dorecovermarks
     \!!toksb=\@EA{\@EA\domark\the\scratchcounter{#1}}%
   \@else
     \ifnofirstmarker
       \long\setgvalue{\@@markfirst@@\markname}{#1}%
       \ifnum\scratchcounter>\topmarker\relax
         \nofirstmarkerfalse
       \@fi
     \@fi
     \long\setgvalue{\@@markbot@@\markname}{#1}%
     \!!toksa=\@EA{\@EA\domark\the\scratchcounter{#1}}%
   \@fi}
```

16
```
\def\doscanmarks%
  {\afterassignment\dodoscanmarks\scratchcounter=}
```

17
```
\long\def\dorecovermarks#1\lastmark%
  {\!!toksc={\domark#1}}
```

18
```
\def\lastmark%
  {\!!toksc={}}
```

No watch what happens next. Because we used an indirect call to the mark mechanism we can redefine the original `\mark` command.

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

19
```
\newmark\mark
```

One final advice. Use marks with care. When used in globally assigned boxes, the list can grow quite big, and processing can slow down considerably.

20
```
\protect
```

4.13 Multi Column Output

1
```
\writestatus{loading}{Context Support Macros / Multi Column Output}
```

2
```
\unprotect
```

Multi–column output: the main routines

The following macro's implement a multi–column output routine. The original implementation was based on Donald Knuth's implementation, which was adapted by Craig Platt to support balancing of the last page. I gradually adapted Platt's version to our needs but under certain circumstances things still went wrong. I considered all calls to Platt's `\balancingerror` as undesirable.

3
```
\startmessages  dutch  library: columns
  title: kolommen
      1: maximaal -- kolommen
      2: gebruik eventueel \string\filbreak
      3: probleempje, probeer [balanceren=nee]
      4: plaatsblok boven nog niet mogelijk
      5: plaatsblok onder nog niet mogelijk
      6: -- plaatsblok(en) opgeschort
      7: balanceren afgebroken na 100 stappen
      8: gebalanceerd in -- stap(pen)
      9: uitlijnen controleren!
     10: (minder dan) 1 regel over
     11: plaatsblok te breed voor kolom
     12: plaatsblok verplaatst naar volgende kolom
     13: breed figuur geplaatst boven kolommen
\stopmessages
```

4
```
\startmessages  english  library: columns
  title: columns
      1: only -- columns possible
      2: use \string\filbreak\space as alternative
      3: problems, disable balancing
      4: top float not yet supported
      5: bottom float not yet supported
      6: -- float(s) postponed
      7: balancing aborted after 100 steps
      8: balanced in -- step(s)
      9: check raggedness
     10: (less than) 1 line left
     11: float to wide for column
     12: float moved to next column
     13: wide float moved to top of columns
\stopmessages
```

5
```
\startmessages  german  library: columns
  title: Spalten
      1: nur -- Spalten moeglich
      2: benutzte \string\filbreak\space als Alternative
      3: Problem, verwende [ausgleich=nein]
      4: Gleitobjekt oben ncoh nicht unterstuetzt
      5: Gleitobjekt unten ncoh nicht unterstuetzt
      6: -- Gleitobjekt(e) verschoben
      7: ausgleich nach 100 Schritten abgebrocheb
      8: ausgeglichen nach  -- Schritt(en)
      9: Ausrichtung ueberpruefen
     10: (weniger als) 1 Zeile uebrig
     11: Gleitobjekt zu breit fuer Spalte
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
     12: Gleitobjekt in naechste Zeile verschoben
     13: breites Gleitobjekt an den Anfang der Spalten verschoben
\stopmessages
```

This completely new implementation can handle enough situations for everyday documents, but is still far from perfect. While at the moment the routine doesn't support all kind of floats, it does support:

- an unlimitted number of columns
- ragged or not ragged bottoms
- optional balancing without `\balancingerrors`
- different `\baselineskips`, `\spacing`, `\topskip` and `\maxdepth`
- left- and right indentation, e.g. within lists
- moving columns floats to the next column or page
- handling of floats that are to wide for a columns

One could wonder why single and multi–columns modes are still separated. One reason for this is that TEX is not suited well for handling multi–columns. As a result, the single columns routines are more robust. Handling one column as a special case of multi–columns is posible but at the cost of worse float handling, worse page breaking, worse etc. Complicated multi–column page handling should be done in DTP–systems anyway.

There are three commands provided for entering and leaving multi–column mode and for going to the next column:

```
\beginmulticolumns
```

`\endmulticolumns`

`\ejectcolumn`

This routines are sort of stand–alone. They communicate with the rest of CONTEXT by means of some interface macro's, which we only mention.

`\nofcolumns`	the number of columns
`\betweencolumns`	the stuff between columns
`\finaloutput{material}`	some kind of `\pagebody` and `\shipout`
`\ifbalancecolumns`	balancing the colums or not
`\ifstretchcolumns`	ragging the bottom or not
`\ifheightencolumns`	fix the heigh tor not
`\fixedcolumnheight`	the optional fixed height
`\ifinheritcolumns`	handle ragging or not
`\ifr@ggedbottom`	use ragged bottoms
`\ifb@selinebottom`	put the bottom line on the
`\ifnormalbottom`	put the bottom line at the
`\usercolumnwidth`	the calculated width of a column
`\columntextwidth`	the maximum width of a column
`\columntextheight`	the minimum width of a column

`\spacingfactor`	the spacing factor
`\openlineheight`	the lineheight (including `\spacing`)
`\openstrutheight`	the height of a `\strut` (including `\spacing`)
`\openstrutdepth`	the depth of a `\strut` (including `\spacing`)
`\EveryCorps`	communication channel to font switching routines
`\global\settopskip`	set `\topskip`
`\setcolumnwarnings`	set `\badness` and `\fuzz`
`\setcolumninserts`	set `\insert`'s
`\setvsize`	set `\vsize` and `\pagegoal`
`\sethsize`	set `\hsize`
`\flushcolumnfloats`	push saved column floats (next page)
`\flushcolumnfloat`	push saved column floats (next column)
`\setcolumnfloats`	initialize column floats
`\finishcolumnbox`	do something special (a hook)

These interface macro's are called upon or initialized by the multi–column macro's.

```
\def\columntextwidth          {\zetbreedte}
\def\columntextheight         {\teksthoogte}
\def\usercolumnwidth          {\tekstbreedte}
```

7
```
\def\fixedcolumnheight      {\teksthoogte}
\def\betweencolumns         {\hskip1em}
```

8
```
\def\setcolumnwarnings      {\dontcomplaincolumnboxes}
\def\setcolumninserts       {\dontpermitcolumninserts}
```

9
```
\def\setcolumnfloats        {} % in CONTEXT used for floats
\def\flushcolumnfloats      {} % in CONTEXT used for floats
\def\flushcolumnfloat       {} % in CONTEXT used for floats
```

10
```
\def\finishcolumnbox        {} % in CONTEXT used for backgrounds
```

In fact, the column height and width are set by mens of two macro's. One can change their meaning if needed:

11
```
\def\setcolumntextheight%
   {\def\columntextheight{\teksthoogte}}
```

12
```
\def\setcolumntextwidth%
   {\def\columntextwidth{\tekstbreedte}}
```

Both macros are redefined in CONTEXT when backgrounds are applied to columns.

13
```
\newcount\nofcolumns        \nofcolumns=2
```

14
```
\def\maxnofcolumns          {16}
\def\allocatednofcolumns    {0}
```

15
```
\newif\ifbalancecolumns    \balancecolumnsfalse
\newif\ifstretchcolumns    \stretchcolumnsfalse
\newif\ifinheritcolumns    \inheritcolumnsfalse
\newif\ifheightencolumns   \heightencolumnsfalse
```

16
```
\newbox\partialpage
\newskip\partialpageskip
```

17
```
\newbox\restofpage
\newbox\savedfloatlist
```

18
```
\newdimen\intercolumnwidth
\newdimen\localcolumnwidth
\newdimen\partialpageheight
```

19
```
\newtoks\singlecolumnout
```

During initialization the temporary boxes are allocated. This enables us to use as much columns as we want, without exhausting the pool of boxes too fast. We could have packed them in one box, but we've got enough boxes.

Two sets of boxes are declared, the txtboxes are used for the text, the topboxes are for moved column floats.

20
```
\def\@@txtcol{@@txtcol}
\def\@@topcol{@@topcol}
```

21
```
\def\initializemulticolumns#1%
  {\ifnum#1>\maxnofcolumns\relax
     \showmessage{\m!columns}{1}{\maxnofcolumns}%
     \nofcolumns=\maxnofcolumns
   \else
     \nofcolumns=#1\relax
   \fi
   \ifnum\nofcolumns>\allocatednofcolumns\relax
     \dorecurse
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

```
        {#1}
        {\ifnum\recurselevel>\allocatednofcolumns\relax
           \newbox\next
           \global\letvalue{\@@txtcol\recurselevel}=\next
           \newbox\next
           \global\letvalue{\@@topcol\recurselevel}=\next
         \fi}%
      \xdef\allocatednofcolumns{\the\nofcolumns}%
    \fi
    \edef\firstcolumn{\getvalue{\@@txtcol1}}%
    \edef\firsttopcolumn{\getvalue{\@@topcol1}}%
    \edef\lastcolumn{\getvalue{\@@txtcol\the\nofcolumns}}%
    \edef\lasttopcolumn{\getvalue{\@@topcol\the\nofcolumns}}}
```

Without going in details we present two macro's which handle the columns. The action which is transfered by the the first and only parameter can do something with `\currentcolumn`. In case of the mid columns, `\firstcolumn` and `\lastcolumn` are handled outside these macro's.

22
```
\def\dohandlemidcolumns#1%
  {\dorecurse
     {\nofcolumns}
     {\ifnum\recurselevel>1
        \ifnum\recurselevel<\nofcolumns\relax
           \edef\currentcolumn{\getvalue{\@@txtcol\recurselevel}}%
           \edef\currenttopcolumn{\getvalue{\@@topcol\recurselevel}}%
           #1\relax
        \fi
      \fi}}
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

23
```
\def\dohandleallcolumns#1%
  {\dorecurse
     {\nofcolumns}
     {\edef\currentcolumn{\getvalue{\@@txtcol\recurselevel}}%
      \edef\currenttopcolumn{\getvalue{\@@topcol\recurselevel}}%
      #1\relax}}
```

Going to a new columns is done by means of a `\ejectcolumn`. The following definition does not always work.

24
```
\def\ejectcolumn%
  {\goodbreak
   \showmessage{\m!columns}{2}{}}
```

The next macro should never be called so let's deal with it. There were several solutions to these kind of errors. First we check for a good breakpoint before firing up the multi–column routine (`\break` or `\allowbreak`). We do the same at the end of the routine (`\allowbreak`). These allowances are definitely needed!

Some on first sight redundant calls to for instance `\setvsize` in the flushing, splitting and balancing macro's can definitely not be omitted! Some are just there to handle situations that only few times arise. One of those can be that the output routine is invoked before everything is taken care of. This happens when we flush (part of) the current page with an `\unvbox` with a `\pagetotal` ≈ `\pagegoal`. One simply cannot balance columns that are just balanced.

I hope one never sees the following message. Because it took me a lot of time to develop the multi–columns routines, every (although seldom) warning gives me the creeps!

25
```
\def\balancingerror%
  {\showmessage{\m!columns}{3}{}%
   \finaloutput{\unvbox255}}
```

Here we present the two `\dont...` macro's, which are of course CONTEXT–specific ones.

26
```
\def\dontcomplaincolumnboxes%
  {\mindermeldingen}
```

27
```
\def\dontpermitcolumninserts%
  {\def\dotopfloat%
     {\showmessage{\m!columns}{4}{}%
      \doexecfloat}%
   \def\dobotfloat%
     {\showmessage{\m!columns}{5}{}%
      \doexecfloat}}
```

28
```
\def\getinsertionheights\to#1\\%
  {#1=\!!zeropoint
   \def\doaddinsertionheight##1%
     {\ifvoid##1\else
        \advance#1 by \skip##1
        \advance#1 by \ht##1
      \fi}%
   \doaddinsertionheight\topins
   \doaddinsertionheight\botins
   \doaddinsertionheight\footins}
```

The local column width is available in the dimension register `\localcolumnwidth`, which is calculated as:

29
```
\def\setcolumnhsize%
  {\setbox0=\hbox{\parindent\!!zeropoint\betweencolumns}%
   \intercolumnwidth=\wd0
   \localcolumnwidth=\columntextwidth
```

supp-mis supp-ver supp-vis supp-lan supp-pdf supp-spe supp-mps supp-tpi supp-fil supp-ini supp-box supp-mrk supp-mul supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

```
   \advance\localcolumnwidth by -\leftskip
   \advance\localcolumnwidth by -\rightskip
   \advance\localcolumnwidth by -\nofcolumns\intercolumnwidth\relax
   \advance\localcolumnwidth by \intercolumnwidth\relax
   \divide\localcolumnwidth  by \nofcolumns
   \usercolumnwidth=\localcolumnwidth
   \hsize=\localcolumnwidth} % we don't do it \global
```

30
```
\def\setcolumnvsize%
  {\global\vsize=\columntextheight
   \ifdim\partialpageheight>\!!zeropoint
     \global\advance\vsize by -\partialpageheight  % \ht\partialpage
     %\global\advance\vsize by -\openstrutdepth
   \fi
   \getinsertionheights\to\dimen0\\
   \global\advance\vsize by -\dimen0
   \global\vsize=\nofcolumns\vsize
   \global\pagegoal=\vsize} % let's do it only here
```

It really starts here. After some checks and initializations we change the output routine to continous multi–column mode. This mode handles columns that fill the current and next full pages. The method used is (more or less) multiplying `\vsize` and dividing `\hsize` by `\nofcolumns`. More on this can be found in the TEXbook. We save the top of the current page in box `\partialpage`.

We manipulate `\topskip` a bit, just to be shure that is has no flexibility. This has te be done every time a font switch takles place, because `\topskip` can depend on this.

Watch the trick with the `\vbox`. This way we get the right interlining and white space.

31
```
\def\beginmulticolumns%
  {\par
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
\begingroup
\dontshowcomposition
\setcolumntextwidth\relax
\setcolumntextheight\relax
\ifsomefloatwaiting
  \showmessage{\m!columns}{6}{\the\savednoffloats}%
  \global\setbox\savedfloatlist=\box\floatlist
  \edef\restoresavedfloats%
    {\global\savednoffloats=\the\savednoffloats
     \global\setbox\floatlist=\box\savedfloatlist
     \global\noexpand\somefloatwaitingtrue}%
  \global\savednoffloats=0
  \global\somefloatwaitingfalse
\else
  \let\restoresavedfloats=\relax
\fi
%\global\partialpageskip=\lastskip          % vervallen
\dimen0=\pagetotal
\advance\dimen0 by \parskip
\advance\dimen0 by \openlineheight
\ifdim\dimen0<\pagegoal
  \allowbreak
\else
  \break
\fi
\EveryCorps{\topskip=1\topskip}%            % nog nodig ?
\the\everycorps                             % nog nodig ?
\initializemulticolumns\nofcolumns
\setcolumninserts
```

```
\hangafter=0\relax
\hangindent=\!!zeropoint\relax
\everypar{}% \everypar={\flushcolumnfloat}%
\ifdim\pagetotal=\!!zeropoint\relax         % later toegevoegd
\else                                       % later toegevoegd
   \vbox{\strut}%                           % toegevoegd
   \vskip-\lineskip                         % toegevoegd
   \vskip-\openlineheight                   % toegevoegd
\fi                                         % later toegevoegd
%\global\partialpageheight=\pagetotal       % vervangen door \ht\partialpage
\global\singlecolumnout=\output
\global\output={\global\setbox\partialpage=\vbox{\unvbox255}}%
\eject
\global\partialpageheight=\ht\partialpage
\global\output={\continuousmulticolumnsout}%
\setcolumnfloats
\dohandleallcolumns
  {\global\setbox\currenttopcolumn=\box\voidb@x}%
\let\sethsize=\setcolumnhsize
\let\setvsize=\setcolumnvsize
\sethsize
\setvsize
\showcomposition}
```

When we leave the multi–column mode, we have to process the not yet shipped out part of the columns. When we don't balance, we simply force a continuous output, but a balanced output is more tricky.

First we try to fill up the page and when all or something is left we try to balance things. This is another useful adaption of the ancesters of these macro's. It takes some reasoning to find out what happens and maybe I'm making some mistake, but it works.

Unvoiding box `\partialpage` is sometimes necessary, e.g. when there is no text given between `\begin..` and `\end..`. The `\par` is needed!

32
```
\def\endmulticolumns%
  {\dontshowcomposition
   \doflushcolumnfloats % added recently
   \par
   \ifbalancecolumns
     \global\output={\continuousmulticolumnsout}%
     \goodbreak
     \global\output={\balancedmulticolumnsout}%
   \else
     \goodbreak
   \fi
   \eject                 % the prevdepth is important, try e.g. toclist in
   \prevdepth\!!zeropoint % columns before some noncolumned text text
   \global\output=\singlecolumnout
   \ifvoid\partialpage\else
     \unvbox\partialpage
   \fi
   \global\partialpageheight=\!!zeropoint
   \nofcolumns=1
   \setvsize
   \dosomebreak\allowbreak
   \restoresavedfloats
   \endgroup}
```

Because some initializations happen three times, we defined a macro for them. The `\everypar{}` is needed because we don't want anything to interfere.

33
```
\def\setmulticolumnsout%
  {\everypar{}%
   \setcolumnwarnings
   \settopskip
   \setmaxdepth
   \topskip=1\topskip
   \splittopskip=\topskip
   \splitmaxdepth=\maxdepth
   \boxmaxdepth=\maxdepth}
```

Flushing the page comes to pasting the columns together and appending the result to box `\partialpage`, if not void. I've seen a lot of implementations in which some skip was put between normal text and multi–column text. When we don't want this, the baselines can be messed up. I hope the seemingly complicated calculation of a correction `\kern` is adequate to overcome this. Although not watertight, spacing is taken into account and even multiple mode changes on one page go well. But cross your fingers and don't blame me.

One of the complications of flushing out the boxes is that `\partialpage` needs to be `\unvbox`'ed, otherwise there is too less flexibility in the page when using `\r@ggedbottom`. It took a lot of time before these kind of problems were overcome. Using `\unvbox` at the wrong moment can generate `\balancingerror`'s.

34
```
\def\flushcolumnedpage%
  {\bgroup
   \setmulticolumnsout
   \showcomposition
   %
   %\dohandleallcolumns{\wd\currentcolumn=\localcolumnwidth}%
%\ifheightencolumns
%  \dohandleallcolumns{\ht\currentcolumn=\fixedcolumnheight}%
%\fi
%
\dohandleallcolumns % \hbox i.v.m. \showcomposition
  {\global\setbox\currentcolumn=\hbox to \localcolumnwidth
     {\box\currentcolumn
      \global\wd\currentcolumn=\localcolumnwidth
      \ifheightencolumns
        \global\ht\currentcolumn=\fixedcolumnheight
      \fi}}%
\setbox0=\vbox
  {\hbox to \columntextwidth
     {\finishcolumnbox{\box\firstcolumn}\betweencolumns\hfil
      \dohandlemidcolumns
        {\finishcolumnbox{\box\currentcolumn}\betweencolumns\hfil}%
      \finishcolumnbox{\box\lastcolumn}}}%
\dohandleallcolumns
  {\global\setbox\currenttopcolumn=\box\voidb@x}%
\ifvoid\partialpage
\else
  \unvbox\partialpage
\fi
\global\partialpageheight=\!!zeropoint
\setvsize
\dosomebreak\nobreak
\dp0=\!!zeropoint
\box0
\egroup}
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

In case one didn't notice, finaly `\finishcolumnbox` is applied to all boxes. One can use this hook for special purposes.

Here comes the simple splitting routine. It's a bit longer than expected because of ragging bottoms or not. This part can be a bit shorter but I suppose that I will forget what happens. The splitting takes some already present material (think of floats) into account!

First we present some auxiliary routines. Any material, like for instance floats, that is already present in the boxes is preserved.

35
```
\def\splitcolumn#1from \box#2to \dimen#3 top \box#4%
  {\bgroup
   \ifdim\ht#4>\!!zeropoint
     \dimen0=\dimen#3\relax
     \dimen2=\dimen#3\relax
     \advance\dimen0 by -\ht#4
     \setbox0=\vsplit#2 to \dimen0
     \global\setbox#1=\vbox to \dimen2{\unvcopy#4\unvbox0}%
   \else
     \global\setbox#1=\vsplit#2 to \dimen#3
   \fi
   \egroup}
```

36
```
\def\splitcurrentcolumn from \box#1to \dimen#2%
  {\splitcolumn\currentcolumn from \box#1 to \dimen#2 top \box\currenttopcolumn}
```

37
```
\def\splitfirstcolumn from \box#1to \dimen#2%
  {\splitcolumn\firstcolumn from \box#1 to \dimen#2 top \box\firsttopcolumn}
```

38
```
\def\splitlastcolumn from \box#1to \dimen#2%
  {\global\setbox\lastcolumn=\vbox
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
     {\unvcopy\lasttopcolumn
      \unvbox#1}}
```

Here comes the routine that splits the long box in columns. The macro `\flushcolumnfloats` can be used to flush either floats that were present before the multi–column mode was entered, or floats that migrate to next columns. Flushing floats is a delicate process.

39
```
\def\continuousmulticolumnsout%
  {\bgroup
   \setmulticolumnsout
   \dontshowcomposition
   \dimen0=\columntextheight
   %\advance\dimen0 by -\maxdepth % wel of niet (niet dus)
   \advance\dimen0 by -\partialpageheight
   \getinsertionheights\to\dimen2\\% toegevoegd ivm voetnoten
   \advance\dimen0 by -\dimen2     % idem
   \dohandleallcolumns
     {\splitcurrentcolumn from \box255 to \dimen0}%
   \setbox\restofpage=\vbox{\unvbox255}%
   \ifinheritcolumns
     \ifr@ggedbottom
       \dohandleallcolumns
         {\global\setbox\currentcolumn=\vbox to \dimen0
            {\unvbox\currentcolumn
             \vfill}}%
     \fi
     \ifn@rmalbottom
       \advance\dimen0 by \maxdepth
       \dohandleallcolumns
         {\global\setbox\currentcolumn=\vbox to \dimen0
```

```
           {\unvbox\currentcolumn}}%
    \fi
    \ifb@selinebottom
      % the columns are on top of the baseline
    \fi
  \else
    \dohandleallcolumns
      {\global\setbox\currentcolumn=\vbox to \dimen0
         {\ifstretchcolumns
            \unvbox\currentcolumn
          \else
            \unvbox\currentcolumn % wel of niet \unvbox ?
            \vfill
          \fi}}%
    \dohandleallcolumns
      {\global\ht\currentcolumn=\dimen0}%
  \fi
  \finaloutput{\flushcolumnedpage}%
  \sethsize
  \setvsize
  \flushcolumnfloats
  \unvbox\restofpage
  % \penalty\outputpenalty % gaat gruwelijk mis in opsommingen
  \egroup}
```

And this is the balancing stuff. Again, part of the routine is dedicated to handling ragged bottoms, but here we also see some handling concerning the stretching of columns. We set `\widowpenalty` at 0, which enables us to balance columns with few lines. The use of `\box2` and `\box4` garantees a more robust check when skips are used.

40

```
\def\balancedmulticolumnsout%
  {\bgroup
   \setmulticolumnsout
   \dontshowcomposition
   \widowpenalty=0
   \setbox0=\vbox{\unvbox255}%
   \ifdim\ht0>\openlineheight
     \dimen0=\ht0
     \advance\dimen0 by \topskip
     \advance\dimen0 by -\baselineskip
     \divide\dimen0 by \nofcolumns
     \vbadness=\!!tenthousand\relax
     \count255=0
     \bgroup
     \dimen2=\!!onepoint
     \dimen2=\spacingfactor\dimen2
     \loop
       \advance\count255 by 1
       \global\setbox\restofpage=\copy0\relax
       \splitfirstcolumn from \box\restofpage to \dimen0
       \dohandlemidcolumns
         {\splitcurrentcolumn from \box\restofpage to \dimen0}%
       \splitlastcolumn from \box\restofpage to \dimen0
       \setbox2=\vbox{\unvcopy\firstcolumn}%
       \dimen4=\!!zeropoint
       \dohandleallcolumns
         {\setbox4=\vbox{\unvcopy\currentcolumn}%
          \dimen6=\ht4
          \ifdim\dimen6>\dimen4 \dimen4=\dimen6\fi}%
```

```
    \donefalse
    \ifnum\count255>100\relax
      \donefalse
    \fi
    \ifdim\dimen4>\ht2
      \donetrue
    \fi
    \ifdone
      \advance\dimen0 by \dimen2\relax
  \repeat
  \dohandleallcolumns
     {\global\setbox\currentcolumn=\vbox{\unvcopy\currentcolumn}}% NIEUW
  \ifnum\count255>100\relax
    \showmessage{\m!columns}{7}{}%
  \else
    \showmessage{\m!columns}{8}{\the\count255\space}%
  \fi
  \egroup
  \ifinheritcolumns
    \dimen0=\ht\firstcolumn
    \dimen2=\ht\firstcolumn
    \advance\dimen2 by -\openlineheight
    \dohandleallcolumns
      {\dimen4=\ht\currentcolumn
       \dimen6=10\openlineheight
       \global\setbox\currentcolumn=\vbox to \dimen0
         {\unvbox\currentcolumn
          \ifdim\dimen4>\dimen6
            \ifdim\dimen4<\dimen0
```

```
                  \ifdim\dimen4>\dimen2
                    \vskip\!!zeropoint  % !!
                  \else
                    \vskip\openlineheight
                    \vfill
                  \fi
                \else
                  \vskip\!!zeropoint
                \fi
              \else
                \vskip\openlineheight
                \vfill
              \fi}}%
      \else
        \bgroup
        \ifstretchcolumns
          \dimen0=\ht\firstcolumn
          \dimen2=\bottomtolerance\ht\firstcolumn
          \setbox0=\vbox{\unvcopy\lastcolumn}%
          \advance\dimen0 by -\ht0\relax
          \advance\dimen0 by -\dp0\relax
          \ifdim\dimen0>\openlineheight\relax
            \ifdim\dimen0>\dimen2\relax
              % \stretchcolumnsfalse % beter goed slecht dan slecht goed
              \showmessage{\m!columns}{9}{}%
            \fi
          \fi
        \fi
        \dohandleallcolumns
```

```
          {\global\setbox\currentcolumn=\vbox to \ht\firstcolumn
             {\ifstretchcolumns
                \unvbox\currentcolumn
              \else
                \box\currentcolumn
                \vfill
              \fi}}%
        \egroup
      \fi
    \else
      \showmessage{\m!columns}{10}{}%
      \global\setbox\firstcolumn=\vbox{\unvbox0}%
    \fi
    \global\output={\balancingerror}%
    \b@selinebottomtrue % forces depth in separation rule
    \flushcolumnedpage
    \egroup}
```

The multicolumn mechanism is incorporated in a CONTEXT interface, which acts like:

```
    \startcolumns[n=4,balance=no,stretch=no,line=on]
      some text
    \stopcolumns
```

The setup is optional. The default behaviour of columns can be set up with:

```
    \setupcolumns
      [n=2,
       balance=yes,
       stretch=text,
       line=off]
```

In this case, stretching is according to the way it's done outside columns (`\inheritcolumnstrue`). Also we can setup the `tolerance` within a column, the `distance` between columns and the fixed `height` of a column.

Multi–column output: the float routines

Here come the routines that handle the placement of column floats. Floats that are to big migrate to the next column. Floats that are too wide, migrate to the top of the next page, where they span as much columns as needed. Floats that are left over from outside the multi–column mode are flushed first. In macro `\finaloutput` the topfloats that are left from previous text should be set.

When there are some floats in the queue, we inhibit the flushing of floats on top of columns. The number of waiting floats is preswent in `\savednoftopfloats` and is saved. As long as there are floats waiting, the topfloats are places as if we are outside multi–column mode. This is neccessary for e.g. multicolumn lists.

When all those floats are flushed, we switch to the local flushing routine.

41
```
\def\setcolumnfloats%
  {\xdef\globalsavednoffloats{\the\savednoffloats}%
   \ifnum\globalsavednoffloats>0
     \setglobalcolumnfloats
   \else
     \setlocalcolumnfloats
   \fi}
```

42
```
\def\setglobalcolumnfloats%
  {\everypar={}%
   \let\flushcolumnfloat=\relax
   \let\doroomfloat=\relax
   \let\flushcolumnfloats=\noflushcolumnfloats}
```

43
```
\def\setlocalcolumnfloats%
  {\everypar={\flushcolumnfloat\checkindentation}% nog documenteren
   \let\flushcolumnfloat=\doflushcolumnfloat
   \let\doroomfloat=\docolumnroomfloat
   \let\flushcolumnfloats=\doflushcolumnfloats
   \let\dosetbothinserts=\relax
   \let\dotopinsertions=\relax}
```

44
```
\def\noflushcolumnfloats%
  {\bgroup
   \xdef\localsavednoffloats{\the\savednoffloats}%
   \global\savednoffloats=\globalsavednoffloats
   \dotopinsertions
   \xdef\globalsavenoffloats{\the\savednoffloats}%
   \ifnum\globalsavednoffloats=0
     \setlocalcolumnfloats
   \fi
   \global\savednoffloats=\localsavednoffloats
   \egroup}
```

We need to calculate the amount of free space in a columns. When there is not enough room, we migrate the float to the next column. These macro's are alternatives (and look–alikes) of `\doroomfloat`. When a float is to wide, for one column, it is moved to the top of the next page. Of course such moved floats have to be taken into account when we calculate the available space. It's a pitty that such things are no integral part of TEX.

45
```
\def\getcolumnstatus\column#1\total#2\goal#3\\%
  {\ifdim\pagegoal<\maxdimen
     \dimen0=\pagegoal
     \divide\dimen0 by \nofcolumns
```

```
      \dimen2=\!!zeropoint
      \count255=0\relax
      \dimen8=\columntextheight
      \advance\dimen8 by -\partialpageheight
      %\advance\dimen8 by -\maxdepth % recently deleted
      \def\dogetcolumnstatus
        {\advance\count255 by 1\relax
         \advance\dimen2 by \ht\currenttopcolumn
         \advance\dimen2 by \dp\currenttopcolumn
         \dimen4=\dimen2\relax
         \advance\dimen4 by \pagetotal
         \dimen6=\count255\dimen8
         \ifdim\dimen4>\dimen6
         \else
           \let\dogetcolumnstatus=\relax
         \fi}%
      \dohandleallcolumns{\dogetcolumnstatus}%
      #1=\count255
      #2=\dimen4
      #3=\dimen6
    \else
      #1=0
      #2=\pagetotal
      #3=\pagegoal
    \fi}
```

46
```
\def\docolumnroomfloat%
  {\ifnofloatpermitted
     \global\roomforfloatfalse
   \else
```

```
    \getcolumnstatus\column\count255\total\dimen0\goal\dimen2\\%
    \advance\dimen0 by \ht\floatbox
    \advance\dimen0 by \dp\floatbox
    \advance\dimen0 by \floattopskip
    % \advance\dimen0 by -\pageshrink nog eens testen
    \ifdim\dimen0>\dimen2
      \global\roomforfloatfalse
    \else
      \global\roomforfloattrue
    \fi
    \ifdim\wd\floatbox>\hsize
      \showmessage{\m!columns}{11}{}%
      \global\roomforfloatfalse
    \fi
  \fi}
```

Flushing one float is done as soon as possible, i.e. `\everypar`. This means that (at the moment) sidefloats are not supported (overulled)!

47
```
\def\doflushcolumnfloat%
  {\bgroup
   \ifsomefloatwaiting
    \let\doflushcolumnfloat=\relax
    \getcolumnstatus\column\count255\total\dimen0\goal\dimen2\\%
    \ifdim\dimen0>\!!zeropoint
      \dogetfloat
      \ifdim\wd\floatbox>\hsize
        \doresavefloat
      \else
        \setbox2=\vbox
```

```
          {\blanko[\@@bkvoorwit]
           \copy\floatbox
           \blanko[\@@bknawit]}%
        \advance\dimen0 by \ht2
        \advance\dimen0 by 2\openlineheight % still neccessary ?
        \ifdim\dimen0>\dimen2
          \showmessage{\m!columns}{12}{}%
          \doresavefloat
        \else
          \ifhmode{\setbox0=\lastbox}\fi% waar is die er in geslopen
          \par
          \ifdim\prevdepth<\!!zeropoint\relax % anders bovenaan kolom witruimte
          \else
            \blanko[\@@bkvoorwit]
          \fi
          \copy\floatbox
          \blanko[\@@bknawit]
        \fi
      \fi
    \fi
  \fi
  \egroup}
```

This one looks complicated. Upto `\nofcolumns` floats are placed, taking the width of a float into account. This routine can be improved on different ways:

- taking into account some imaginary baseline, just to get the captions in line
- multipass flushing until as many floats are displaced as possible

When handling lots of (small) floats spacing can get worse because of lining out the columns.

48
```
\def\doflushcolumnfloats%
  {\bgroup
   \ifnum\savednoffloats>1\relax % no \ifsomefloatwaiting
     \dimen8=\!!zeropoint
     \dimen4=\!!zeropoint
     \count0=0             % count0 can be used local
     \count2=\nofcolumns   % count2 can be used local
     \dohandleallcolumns
       {\ifnum\count0>0\relax % the wide one's reserved space
          \global\setbox\currenttopcolumn=
            \vbox{\vphantom{\copy\floatbox}\witruimte\blanko[\@@bknawit]}%
        \else
          \dogetfloat
          \ifdim\wd\floatbox>\hsize
            \dimen0=\wd\floatbox
            \advance\dimen0 by \intercolumnwidth
            \dimen2=\hsize
            \advance\dimen2 by \intercolumnwidth
            \divide\dimen0 by \dimen2
            \count0=\dimen0
            \advance\count0 by 1
            \ifnum\count0>\count2
              \doresavefloat
            \else
              \dimen0=\count0\hsize
              \advance\dimen0 by \count0\intercolumnwidth
              \advance\dimen0 by -\intercolumnwidth
              \wd\floatbox=.5\wd\floatbox
              \setbox\floatbox=\hbox to \dimen0{\hss\box\floatbox\hss}%
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
          \fi
          \showmessage{\m!columns}{13}{}%
        \else
          \showmessage{\m!columns}{13}{}%
        \fi
        \ifdim\ht\floatbox>\!!zeropoint\relax
          \global\setbox\currenttopcolumn=
            \vbox
              {\copy\floatbox
               \witruimte % nodig ?
               \blanko[\@@bknawit]}%
        \fi
        \dimen6=\ht\currenttopcolumn
        \advance\dimen6 by \dp\currenttopcolumn
      \fi
      \ifdim\dimen4<\ht\currenttopcolumn
        \dimen4=\ht\currenttopcolumn
      \fi
      \advance\dimen8 by \dimen6
      \advance\count2 by -1
      \advance\count0 by -1\relax}%
   \setvsize
   \global\advance\vsize by -\dimen8
   \global\pagegoal=\vsize
 \else
   \doflushfloats
 \fi
 \egroup}
```

This were the multi–column routines. They can and need to be improved but at the moment their behaviour is acceptable.

One inprovement can be to normalize the height of floats to $n \times \backslash lineheight$ with a macro like:

```
\normalizevbox{...}
```

49 `\protect`

4.14 Fun Stuff

1
```
\unprotect
```

2
```
\def\horizontalpositionbar#1\pos#2\min#3\max#4\token#5\\%
  {\hbox to \hsize
     {\hskip\!!zeropoint\!!plus #1\!!fill
      \hskip\!!zeropoint\!!plus-#2\!!fill
      #4\relax
      \hskip\!!zeropoint\!!plus #3\!!fill
      \hskip\!!zeropoint\!!plus-#1\!!fill}}
```

3
```
\def\verticalpositionbar\pos#1\min#2\max#3\token#4\\%
  {\vbox to \vsize
     {\vskip\!!zeropoint\!!plus #1\!!fill
      \vskip\!!zeropoint\!!plus-#2\!!fill
      \hbox{#4}\relax
      \vskip\!!zeropoint\!!plus #3\!!fill
      \vskip\!!zeropoint\!!plus-#1\!!fill}}
```

4
```
\def\horizontalgrowingbar\pos#1\min#2\max#3\height#4\depth#5\\%
  {\hbox to \hsize
     {\scratchcounter=#1\relax
      \advance\scratchcounter by -#2\relax
      \advance\scratchcounter by 1\relax
      \leaders\vrule\hskip\!!zeropoint\!!plus \scratchcounter\!!fill
      \vrule\!!width\!!zeropoint\!!height#4\!!depth#5\relax
      \hskip\!!zeropoint\!!plus #3\!!fill
      \hskip\!!zeropoint\!!plus-#1\!!fill}}
```

supp-mis
supp-ver
supp-vis
supp-lan
supp-pdf
supp-spe
supp-mps
supp-tpi
supp-fil
supp-ini
supp-box
supp-mrk
supp-mul
supp-fun

contents register context syst mult supp lang font colo spec core cont m s exit go back

5
```
\def\verticalgrowingbar\pos#1\min#2\max#3\width#4\\%
  {\vbox to \vsize
     {\scratchcounter=#1\relax
      \advance\scratchcounter by -#2\relax
      \advance\scratchcounter by 1\relax
      \leaders\hrule\vskip\!!zeropoint\!!plus\scratchcounter\!!fill
      \hrule\!!width#4\!!height\!!zeropoint\!!depth\!!zeropoint
      \vskip\!!zeropoint\!!plus #3\!!fill
      \vskip\!!zeropoint\!!plus-#1\!!fill}}
```

6
```
\protect
```

5 Language Support

5.1 Initialization

lang-ini

This module implements the (for the moment still simple) multi–language support of CONTEXT, which should not be confused with the multi–lingual interface. This support will be extended when needed.

```
\writestatus{loading}{Context Language Macros / Initialization}
```

```
\unprotect
```

```
\startmessages  dutch  library: linguals
  title: taal
      1: afbreekpatronen voor -- geladen
      2: geen afbreekpatronen voor --
      3: afbreekdefinities voor -- geladen
      4: geen afbreekdefinities voor --
      5: afbreekpatronen voor -- niet geladen
      6: taal -- is niet gedefinieerd
      7: taal specifieke opties [--] introduceren een skip van --
      8: taal specifieke opties [--] naadloos toegevoegd
\stopmessages
```

```
\startmessages  english  library: linguals
  title: language
      1: patterns for -- loaded
      2: no patterns for --
      3: hyphenations for -- loaded
      4: no hyphenations for --
      5: patterns for -- not loaded
      6: language -- is undefined
      7: language specific options [--] introduce a -- skip
      8: language specific options [--] seamless appended
\stopmessages
```

5
```
\startmessages  german  library: linguals
  title: Sprache
      1: Trennmuster fuer -- geladen
      2: Keine Trennmuster fuer --
      3: Trenndefinitionen fuer -- geladen
      4: Keine Trenndefinitionen fuer --
      5: Trennmuster fuer -- nicht geladen
      6: Sprache -- ist undefiniert
      7: Sprachenspezifische Option [--] fuegt eine Luecke von -- ein
      8: Sprachenspezifische Option [--] nahtlos hinzugefuegt
\stopmessages
```

When loading hyphenation patterns, TEX assign a number to each loaded table, starting with 0. Switching to a specific table is done by assigning the relevant number to the predefined ⟨*counter*⟩ `\language`. Unfortunately the name of this command suits very well the name of the language switching command we are to define, so let's save this primitive under another name:

6
```
\let\normallanguage = \language
```

We keep track of the last loaded patterns by means of a pseudo ⟨*counter*⟩. This just one of those situations in which we don't want to spent a real one.

7
```
\newcounter\loadedlanguage
```

We prefer a bit more tolerant hyphenating than PLAIN TEX does, which is definitely due to the dutch origin of CONTEXT.

8
```
\lefthyphenmin  = 2
\righthyphenmin = 2
```

\currentlanguage

Instead of numbers,we are going to use symbolic names for the languages. The current langage is saved in the macro `\currentlanguage`.

lang-ini

```
9 \let\currentlanguage = \empty
```

\installlanguage

Hyphenation patterns can only be loaded when the format file is prepared. The next macro takes care of this loading. A language is specified with

```
\installeertaal[...][..,..=..,..]
```

`...`	*naam*
`spatiering`	opelkaar ruim
`status`	start stop
`linkerzin`	*commando*
`rechterzin`	*commando*
`linkersubzin`	*commando*
`rechtersubzin`	*commando*
`linkerciteer`	*commando*
`rechterciteer`	*commando*
`linkercitaat`	*commando*
`rechtercitaat`	*commando*
`default`	*naam*

When `\c!status` equals `\v!start`, both patterns and additional hyphenation specifications are loaded. These files are seached for on the system path and are to be named:

```
\f!languageprefix-identifier.\f!patternsextension
\f!languageprefix-identifier.\f!hyhensextension
```

The `\c!spatiering` specifies how the spaces after punctuation has to be handled. English is by tradition more tolerant to inter–sentence spacing than other languages.

This macro also defines `\identifier` as a shortcut switch to the language. Furthermore the command defined as being language specific, are executed. With `\c!default` we can default to another language (patterns) at format generation time. This default language is overruled when the appropriate patterns are loaded (some implementations support run time addition of patterns to a preloaded format).

The values `\c!leftsentence` and `\c!rightsentence` can be (and are) used to implement automatic subsentence boundary glyphs, like in «french guillemots» or – german guillemots – or —dutch dashes— like situations. Furthermore `\c!leftquotation` and `\c!leftquote` come into view “when we quote” or ‘quote’ something.

```
\def\doinstalllanguage[#1][#2]%
  {\doifdefinedelse{\??la#1\c!nummer}%
     {\getparameters[\??la#1][#2]}
     {\setevalue{\??la#1\c!nummer}{\loadedlanguage}%
      \increment\loadedlanguage
      \setvalue{#1}{\language[#1]}%
      \getparameters
        [\??la#1]
        [\c!spatiering=\v!opelkaar,
         \c!leftsentence=---,
         \c!rightsentence=---,
         \c!leftsubsentence=---,
         \c!rightsubsentence=---,
         \c!leftquote={`},
         \c!rightquote={'},
         \c!leftquotation={``},
         \c!rightquotation={''},
         \c!datum={\v!dag,\ ,\v!maand,\ ,\v!jaar},
         \c!status=\v!stop,
         \s!done=\v!nee,
         \c!default=#1,
         #2]}%
   \language=\getvalue{\??la#1\c!nummer}\relax
   \doifelsevalue{\??la#1\c!status}{\v!start}
     {\doifelsevalue{\??la#1\s!done}{\v!nee}
        {\readsysfile{\f!languageprefix#1.\f!patternsextension}
           {\getparameters[\??la#1][\s!done=\v!ja,\c!default=#1]%
            \showmessage{\m!linguals}{1}{#1}}
           {\showmessage{\m!linguals}{2}{#1}}%
         \readsysfile{\f!languageprefix#1.\f!hyphensextension}
           {\showmessage{\m!linguals}{3}{#1}}
           {\showmessage{\m!linguals}{4}{#1}}}
        {\showmessage{\m!linguals}{1}{#1}%
         \showmessage{\m!linguals}{3}{#1}}}
     {\showmessage{\m!linguals}{5}{#1}}%
   \language[#1]}
```

11
```
\def\installlanguage%
  {\dodoubleargument\doinstalllanguage}
```

`\language` `\mainlanguage`

Switching to another language (actually another hyphenation pattern) is done with:

```
    \language[identifier]
```

or with `\identifier`. Just to be compatible with PLAIN TEX, we still support the original meaning, so

```
    \language=1
```

is a valid operation.

```
\taal[...]

...       nl fa en du sp
```

We can use `\mainlanguage[identifier]` for setting the (indeed) main language. This is the language used for translating labels like *figure* and *table*. The main language defaults to the current language.

12
```
\def\complexlanguage[#1]%
  {\doifdefinedelse{\??la#1\c!nummer}
     {\processaction
        [\getvalue{\??la#1\c!default}]
        [        #1=>\normallanguage=\getvalue{\??la#1\c!nummer},
         \s!default=>\normallanguage=\getvalue{\??la#1\c!nummer},
         \s!unknown=>\expanded{\language[\getvalue{\??la#1\c!default}]}]%
      \edef\currentlanguage{#1}%
      \enablelanguagespecifics[#1]% obsolete: \getvalue{\??la#1\c!commando}%
      \processaction
        [\getvalue{\??la#1\c!spatiering}]
        [\v!opelkaar=>\frenchspacing,
             \v!ruim=>\nonfrenchspacing,
          \s!unknown=>\frenchspacing]}
     {\showmessage{\m!linguals}{6}{#1}}}
```

13
```
\def\simplelanguage%
  {\normallanguage}
```

14
```
\definecomplexorsimple\language
```

15
```
\def\mainlanguage[#1]%
  {\def\currentmainlanguage{#1}}
```

\translate

Sometimes macros contain language specific words that are to be typeset. Such macros can be made (more) language independant by using:

```
\vertaal[..,..=..,..]

naam     tekst
```

like for instance:

```
   \translate[en=something,nl=iets]
```

which expands to *something* or *iets*, depending on de current language.

16
```
\def\dotranslate[#1]%
  {\getparameters[\??lg][#1]%
   \getvalue{\??lg\currentlanguage}}
```

17
```
\unexpanded\def\translate%
  {\dosingleempty\dotranslate}
```

When used without argument, the last defined values are used. This enables repetitive use like

```
   \en \translate\ means \nl \translate
```

\assigntranslation

This macro is a system macro, and can be used to assign a translation to a macro. Its form is:

```
\assigntranslation[en=something,nl=iets]\to\command
```

18

```
\def\assigntranslation[#1]\to#2%
  {\getparameters[\??lg][#1]%
   \edef#2{\getvalue{\??lg\currentlanguage}}}
```

\startlanguagespecifics
\enablelanguagespecifics

Each language has its own typographic pecularities. Some of those can be influenced by parameters, others are handled by the interface, but as soon as specific commands come into view we need another mechanism. In the macro that activates a language, we call `\enablelanguagespecifics`. This macro in return calls for the setup of language specific macros. Such specifics are defined as:

```
\startlanguagespecifics[du]
  \installcompoundcharacter "a {\"a}
  \installcompoundcharacter "e {\"e}
  \installcompoundcharacter "s {\SS}
\stoplanguagespecifics
```

Instead of `[du]` we can pass a comma separated list, like `[du,nl]`. Next calls to this macro add the specifics to the current list.

Before we actually read the specifics, we first take some precautions that will prevent spurious spaces to creep into the list.

19

```
\def\startlanguagespecifics%
  {\bgroup
   \catcode`\^^I=\@@ignore
   \catcode`\^^M=\@@ignore
   \catcode`\^^L=\@@ignore
   \dostartlanguagespecifics}
```

The main macro looks quite complicated but actually does nothing special. By embedding `\do` we can easily append to the lists and also execute them at will. Just to be sure, we check on spurious spaces.

20
```
\long\def\dostartlanguagespecifics[#1]#2\stoplanguagespecifics%
  {\egroup
   \long\def\docommando##1%
     {\doifdefinedelse{\??la##1\??la}
        {\long\def\do####1####2####3%
           {\setvalue{\??la####1\??la}{\do{####1}{####2####3}}}%
         \getvalue{\??la##1\??la}{#2}}
        {\setvalue{\??la##1\??la}{\do{##1}{#2}}}%
      \bgroup
      \setbox0=\hbox{\enablelanguagespecifics[##1]}%
      \ifdim\wd0>\!!zeropoint
        \showmessage{\m!linguals}{7}{##1,\the\wd0\space}\wait
      \else
        \showmessage{\m!linguals}{8}{##1}%
      \fi
      \egroup}%
   \processcommalist[#1]\docommando}
```

Enabling them is rather straightforward. We only have to define `\do` in such a way that `{ }` is removed and the language key is gobbled.

21
```
\def\enablelanguagespecifics[#1]%
  {\long\def\do##1##2{##2}%
   \getvalue{\??la#1\??la}}
```

\leftguillemot
\rightguillemot
\leftsubguillemot
\rightsubguillemot
\...single...quote
\...double...quote

We assign logical names to all kind of quote and sentence boundary characters.

22
```
\def\lowerleftsingleninequote  {\char44 }
\def\lowerleftdoubleninequote  {\char44\kern-.1em\char44 }
\def\upperleftsingleninequote  {\char39 }
\def\upperleftdoubleninequote  {\char34\kern-.1em}
\def\upperleftsinglesixquote   {\char96 }
\def\upperleftdoublesixquote   {\char96\kern-.1em\char96 }
```

23
```
\def\upperrightsingleninequote {\char39 }
\def\upperrightdoubleninequote {\char34 }
\def\upperrightsinglesixquote  {\char96 }
\def\upperrightdoublesixquote  {\kern-.125em\char92 }
```

24
```
\unexpanded\def\leftguillemot%
  {\dontleavehmode\hbox{\raise.25ex\hbox{$\scriptscriptstyle\ll$}}}
```

25
```
\unexpanded\def\rightguillemot%
  {\hbox{\raise.25ex\hbox{$\scriptscriptstyle\gg$}}}
```

26
```
\unexpanded\def\leftsubguillemot%
  {\dontleavehmode\hbox{\raise.25ex\hbox{$\scriptscriptstyle<$}}}
```

27
```
\unexpanded\def\rightsubguillemot%
  {\hbox{\raise.25ex\hbox{$\scriptscriptstyle>$}}}
```

What quotes will be used, depends on the language in use:

„nederlandsse zuinigheid” ‚dutch’
“engelse humor” ‘english’
„duits degelijkheid“ ‚german‘
≪franse slag≫ <french>

“spaans benauwd” ‘spanish’

macros smashaccent

When we let TEX put an accent on top of a character, such composed characters can get more height that height of a standard `\strut`. The next macro takes care of such unwanted compositions.

We need to reach over the number that specifies the accent, and in doing so we use `\hyphenchar` as a placeholder because it accepts 8 bit numbers in octal, decimal or hexadecimal format. Next we set the height of the accented character to the natural height of the character.

28
```
\let\normalaccent=\accent
```

29
```
\def\dodosmashaccent#1%
  {\setbox0=\hbox{#1}%
   \setbox2=\hbox{\normalaccent\the\hyphenchar\nullfont#1}%
   \ht2=\ht0\box2
   \egroup
   \nobreak}
```

30
```
\def\dosmashaccent%
  {\afterassignment\dodosmashaccent\hyphenchar\nullfont=}
```

31
```
\def\smashaccent%
  {\ifvmode\leavevmode\fi
   \bgroup
   \let\accent=\dosmashaccent}
```

For instance we can say:

```
   \smashaccent\accent""7F Uberhaupt
```

But normally one will use it as a prefix in definitions.

By default we load the most common European languages, including of course dutch, our native language. Watch the loading/postponing of patterns and the defaults!

32
```
\installlanguage
  [\c!nl]
  [\c!spatiering=\v!opelkaar,
   \c!leftsentence=---,
   \c!rightsentence=---,
   \c!leftsubsentence=---,
   \c!rightsubsentence=---,
   \c!leftquote=\lowerleftsingleninequote,
   \c!rightquote=\upperrightsingleninequote,
   \c!leftquotation=\lowerleftdoubleninequote,
   \c!rightquotation=\upperrightdoubleninequote,
   \c!datum={\v!dag,\ ,\v!maand,\ ,\v!jaar},
   \c!status=\v!start]
```

33
```
\installlanguage
  [\c!en]
  [\c!spatiering=\v!ruim,
   \c!leftsentence=---,
   \c!rightsentence=---,
   \c!leftsubsentence=---,
   \c!rightsubsentence=---,
   \c!leftquote=\upperleftsinglesixquote,
   \c!rightquote=\upperrightsingleninequote,
   \c!leftquotation=\upperleftdoublesixquote,
   \c!rightquotation=\upperrightdoubleninequote,
   \c!datum={\v!jaar,\ ,\v!maand,\ ,\v!dag},
   \c!status=\v!start]
```

34
```
\installlanguage
  [\c!du]
  [\c!spatiering=\v!opelkaar,
   \c!leftsentence={\hbox{---~}},
   \c!rightsentence={\hbox{~---}},
   \c!leftsubsentence={---},
   \c!rightsubsentence={---},
   \c!leftquote=\lowerleftsingleninequote,
   \c!rightquote=\upperrightsinglesixquote,
   \c!leftquotation=\lowerleftdoubleninequote,
   \c!rightquotation=\upperrightdoublesixquote,
   \c!datum={\v!dag,{.},\ ,\v!maand,\ ,\v!jaar},
   \c!status=\v!start]
```

35
```
\installlanguage
  [\c!fa]
  [\c!spatiering=\v!opelkaar,
   \c!leftsentence=\leftguillemot,
   \c!rightsentence=\rightguillemot,
   \c!leftsubsentence=\leftsubguillemot,
   \c!rightsubsentence=\rightsubguillemot,
   \c!leftquote=\leftsubguillemot,
   \c!rightquote=\rightsubguillemot,
   \c!leftquotation=\leftguillemot,
   \c!rightquotation=\rightguillemot,
   \c!datum={\v!dag,\ ,\v!maand,\ ,\v!jaar},
   \c!status=\v!start]
```

36
```
\installlanguage
  [\c!sp]
  [\c!spatiering=\v!opelkaar,
   \c!leftsentence=---,
   \c!rightsentence=---,
   \c!leftsubsentence=---,
   \c!rightsubsentence=---,
   \c!leftquote=\upperleftsinglesixquote,
   \c!rightquote=\upperrightsingleninequote,
   \c!leftquotation=\upperleftdoublesixquote,
   \c!rightquotation=\upperrightdoubleninequote,
   \c!datum={\v!dag,\ ,\v!maand,\ ,\v!jaar},
   \c!default=\c!en,
   \c!status=\v!start]
```

Hey look, some experiment:

37
```
\installlanguage
  [nlx]
  [\c!spatiering=\v!opelkaar,
   \c!default=\c!nl,
   \c!status=\v!start]
```

We default to the language belonging to the interface. This is one of the few places outside the interface modules where `\startinterface` is used.

38
```
\mainlanguage
  [\currentlanguage]
```

39
```
\startinterface dutch   \language[\c!nl] \stopinterface
\startinterface english \language[\c!en] \stopinterface
```

```
\startinterface french  \language[\c!fr] \stopinterface
\startinterface german  \language[\c!du] \stopinterface
\startinterface spanish \language[\c!sp] \stopinterface
```

We put these here temporary. Soon there will be a module `lang-ext` to handle this specifics.

```
40  \let\normaldoublequote="

41  \startlanguagespecifics[du]

42    \installcompoundcharacter "a {\"a}
      \installcompoundcharacter "e {\"e}
      \installcompoundcharacter "i {\"\i}
      \installcompoundcharacter "o {\"o}
      \installcompoundcharacter "u {\"u}
      \installcompoundcharacter "s {\SS}
      \installcompoundcharacter "z {\SS}

43    \installcompoundcharacter "A {\smashaccent\"A}
      \installcompoundcharacter "E {\smashaccent\"E}
      \installcompoundcharacter "I {\smashaccent\"I}
      \installcompoundcharacter "O {\smashaccent\"O}
      \installcompoundcharacter "U {\smashaccent\"U}
      \installcompoundcharacter "Z {SZ}
      \installcompoundcharacter "S {SS}

44  \stoplanguagespecifics

45  \startlanguagespecifics[du]

46    \installcompoundcharacter "ck {\discretionary {k-}{k}{ck}}
      \installcompoundcharacter "ff {\discretionary{ff-}{f}{ff}}
```

```
  \installcompoundcharacter "ll {\discretionary{ll-}{l}{ll}}
  \installcompoundcharacter "mm {\discretionary{mm-}{m}{mm}}
  \installcompoundcharacter "nn {\discretionary{nn-}{n}{nn}}
  \installcompoundcharacter "pp {\discretionary{pp-}{p}{pp}}
  \installcompoundcharacter "rr {\discretionary{rr-}{r}{rr}}
  \installcompoundcharacter "tt {\discretionary{tt-}{t}{tt}}

  \installcompoundcharacter "CK {\discretionary {K-}{K}{CK}}
  \installcompoundcharacter "FF {\discretionary{FF-}{F}{FF}}
  \installcompoundcharacter "LL {\discretionary{LL-}{L}{LL}}
  \installcompoundcharacter "MM {\discretionary{MM-}{M}{MM}}
  \installcompoundcharacter "NN {\discretionary{NN-}{N}{NN}}
  \installcompoundcharacter "PP {\discretionary{PP-}{P}{PP}}
  \installcompoundcharacter "RR {\discretionary{RR-}{R}{RR}}
  \installcompoundcharacter "TT {\discretionary{TT-}{T}{TT}}

\stoplanguagespecifics

\startlanguagespecifics[du]

  \installcompoundcharacter "` {\handlequotation\leftquotation}
  \installcompoundcharacter "' {\handlequotation\rightquotation}
  \installcompoundcharacter "< {|<|}
  \installcompoundcharacter "> {|>|}
  \installcompoundcharacter ". {\kern.1em\ignorespaces}

\stoplanguagespecifics

\protect
```

6 Font Support

font-ini
font-ans
font-ibm
font-cmr
font-con
font-eul
font-ams
font-lbr
font-pos
font-ptm
font-pcr
font-phv

contents register context syst mult supp lang font colo spec core cont m s exit go back

6.1 Initialization

1
```
\writestatus{loading}{Context Font Macros (ini)}
```

2
```
\unprotect
```

3
```
\startmessages  dutch  library: fonts
  title: korps
      1: codering --, groepeer zonodig
      2: variant -- wordt geladen
      3: onbekende variant --
      4: korps -- is niet gedefinieerd
      5: stijl -- is niet gedefinieerd
      6: -- wordt geladen
      7: onbekend formaat --
      8: stijl -- gedefinieerd
\stopmessages
```

4
```
\startmessages  english  library: fonts
  title: corps
      1: coding --, one could use grouping
      2: variant -- is loaded
      3: unknown variant --
      4: corps -- is not defined
      5: style -- is not defined
      6: -- is loaded
      7: unknown format --
      8: style -- defined
\stopmessages
```

font-ini
font-ans
font-ibm
font-cmr
font-con
font-eul
font-ams
font-lbr
font-pos
font-ptm
font-pcr
font-phv

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
5 \startmessages  german  library: fonts
    title: Fliesstext
        1: Kodierung --, Gruppierung moeglich
        2: Variante -- ist geladen
        3: Unbekannte Variante --
        4: Fliesstext -- ist nicht definiert
        5: Stil -- ist nicht definiert
        6: -- ist geladen
        7: unbekanntes Format --
        8: Stil -- definiert
  \stopmessages
```

This module is one of the oldest modules of CONTEXT. The macros below evolved out of the PLAIN TEX macros and therefore use a similar naming scheme (`\rm`, `\bf`, etc). This module grew out of our needs. We started with the PLAIN TEX definitions, generalized the underlaying macros, and extended those to a level at which probably no one will ever recognize them.

One important characteristic of the font mechanism presented here is the postponing of font loading. This makes it possible to distribute `fmt` files without bothering about the specific breed of `tfm` files.

Another feature implemented here is the massive switching from roman to sans serif, `teletype` or else. This means one doesn't have to take care of all kind of relations between fonts.

font-ini
font-ans
font-ibm
font-cmr
font-con
font-eul
font-ams
font-lbr
font-pos
font-ptm
font-pcr
font-phv

contents register context syst mult supp lang font colo spec core cont m s exit go back

\rm
\ss
\tt
\hw
\cg

Fonts are defined in separate files. When we define a font, we distinguish between several styles. In most cases we will use:

roman	`\rm`
sansserif	`\ss`
type	`\tt`

The number of styles is not limited to these three. When using Lucida Bright we can for instance also define:

handwritten	`\hw`
calligraphy	`\cg`

Anyone who feels the need, can define additional ones, like

faxfont	`\ff`
blackboard	`\bb`

Or even

hebrew	`\hb`

Styles are grouped in font sets. At the moment there are three main sets defined:

Computer Modern Roman	Knuth	`cmr`
Lucida Bright	Bigelow & Holmes	`lbr`
Standard Postscript Fonts	Adobe	`pos`

There are also some Computer Modern Roman alternatives:

Computer Modern Roman	Knuth & Sauter	`sau`
Euler fonts	Zapf	`eul`
Computer Modern Concrete	Knuth & Zapf	`con`

All these definitions are ordered in files with names like `font-cmr` and `font-pos`, where the last three characters specify the name as known to CONTEXT.

Within such a font set (`cmr`) and style (`\rm`) we can define a number of text font alternatives:

typeface	`\tf`
boldface	`\bf`
slanted	`\sl`
italic	`\it`
boldslanted	`\bs`
bolditalic	`\bi`
smallcaps	`\sc`

The more primitive is (still) predefined:

oldstyle	`\os`

The availability of these alternatives depends on the completeness of a font family and of course the definitions in the font files.

But let's not forget math. In addition to the previous TEX families (the mysterious `\fam`'s) we've got some more:

Math Roman	`\mr`
Math Italic	`\mi`
Math Symbol	`\sy`
Math Extra	`\ex`
Math A	`\ma`
Math B	`\mb`
Math C	`\mc`

Users can call for specific fonts in many ways. Switches to other typefaces, like the switch from normal to bold, are as intuitive as possible, which means that all dependant fonts also switch. One can imagine that this takes quite some processing time.

Internally fonts are stored as combination of size, style and alternative, e.g. `12pt+\ss+\bf`. Users are not confronted with sizes, but use the style or style+alternative to activate them.

During the definition of a corps one can also declare the available larger alternatives:

```
\tf \tfa \tfb \tfc ...
\bf \bfa \bfb \bfc ...
\sl \sla \slb \slc ...
```

The smaller ones are automatically supplied and derived from the the corps environment.

```
\tfx \tfxx
\bfx \bfxx
\slx \slxx
```

There are only two smaller alternatives per style. The larger alternatives on the other hand have no limitations.

These larger alternatives are mostly used in chapter and section titles or on title pages. When one switches to a larger alternative, the bold an other ones automatically adapt themselves:

```
\tfd Hi \bf there\sl, here \tfb I \bf am
```

therefore becomes:

Hi **there**, *here* I **am**

Maybe this mechanism isn't always as logic, but as said before, we tried to make it as intuitive as possible.

So a specific kind of glyph can be characterized by:

family (cmr) + corps (12pt) + style (rm) + alternative (bf) + size (a)

The last component (the size) is optional.

We introduced `\tf` as command to call for the current normally sized typeface. This commands results in roman, sans serif, teletype or whatever style is in charge. Such rather massive switches of style sometimes take more processing time than comfortable. Of course there is a workaround for this: we can call fonts directly by means of commands like:

```
\rmtf \sssl \tttf \rmbsa
```

font-ini
font-ans
font-ibm
font-cmr
font-con
font-eul
font-ams
font-lbr
font-pos
font-ptm
font-pcr
font-phv

contents register context syst mult supp lang font colo spec core cont m s exit go back

One should realize that this fast calls have limitations, they lack for instance automatic super- and subscript support.

This leaves us two more commands: `\tx` and `\txx`. These activate a smaller and even more smaller font than the current one and adapt themselves to the current alternative, so when `\bf` is active, `\tx` gives a smaller boldface, which in turn can be called directly by `\bfx`.

These two smaller alternatives are specified by the corps environment and therefore not necessarily have similar sizes as `\scriptsize` and `\scriptscriptsize`. The main reason for this incompatibility (which can easily be undone) lays in the fact that we often want a bit bigger characters than in math mode. In CONTEXT for instance the `\tx` and `\txx` commands are used for surrogate SMALLCAPS which support both nesting and alternatives, like in **A SMALL WORLD**, which was typeset by

```
\bf\kap{a \kap{small} world}
```

And compare THIS with the slightly larger THIS: scriptstyleTHIS or THIS X STYLE makes a big difference.

\mf Math fonts are a species in their own. They are tightly hooked into smaller and even smaller ones of similar breed to form a tight family. Let's first see how these are related:

```
$\tf x^2+\bf x^2+\sl x^2+\it x^2+\bs x^2+ \bi x^2 =\rm 6x^2$
$\tf x^2+\bf x^2+\sl x^2+\it x^2+\bs x^2+{\bi x^2}=\rm 6x^2$
$\tf x^2+\bf x^2+\sl x^2+\it x^2+\bs x^2+ \bi x^2 =\tf 6x^2$
$\tf x^2+\bf x^2+\sl x^2+\it x^2+\bs x^2+{\bi x^2}=\tf 6x^2$
$\tf x^2+\bf x^2+\sl x^2+\it x^2+\bs x^2+ \bi x^2 =\bf 6x^2$
$\tf x^2+\bf x^2+\sl x^2+\it x^2+\bs x^2+{\bi x^2}=\bf 6x^2$
$\tf x^2+\bf x^2+\sl x^2+\it x^2+\bs x^2+ \bi x^2 =\sl 6x^2$
$\tf x^2+\bf x^2+\sl x^2+\it x^2+\bs x^2+{\bi x^2}=\sl 6x^2$
```

Gives both an expected and unexpected result:

$$\mathrm{x}^2 + \mathbf{x}^2 + \mathrm{x}^2 + x^2 + \mathbf{x}^2 + \boldsymbol{x}^2 = 6\mathrm{x}^2$$
$$\mathrm{x}^2 + \mathbf{x}^2 + \mathrm{x}^2 + x^2 + \mathbf{x}^2 + \boldsymbol{x}^2 = 6\mathrm{x}^2$$
$$\mathrm{x}^2 + \mathbf{x}^2 + \mathrm{x}^2 + x^2 + \mathbf{x}^2 + \boldsymbol{x}^2 = 6\mathrm{x}^2$$
$$\mathrm{x}^2 + \mathbf{x}^2 + \mathrm{x}^2 + x^2 + \mathbf{x}^2 + \boldsymbol{x}^2 = 6\mathrm{x}^2$$
$$\mathrm{x}^2 + \mathbf{x}^2 + \mathrm{x}^2 + x^2 + \mathbf{x}^2 + \boldsymbol{x}^2 = \mathbf{6x}^{\mathbf{2}}$$
$$\mathrm{x}^2 + \mathbf{x}^2 + \mathrm{x}^2 + x^2 + \mathbf{x}^2 + \boldsymbol{x}^2 = \mathbf{6x}^{\mathbf{2}}$$
$$\mathrm{x}^2 + \mathbf{x}^2 + \mathrm{x}^2 + x^2 + \mathbf{x}^2 + \boldsymbol{x}^2 = \mathit{6x}^{\mathit{2}}$$
$$\mathrm{x}^2 + \mathbf{x}^2 + \mathrm{x}^2 + x^2 + \mathbf{x}^2 + \boldsymbol{x}^2 = \mathit{6x}^{\mathit{2}}$$

We see here that the character shapes change accordingly to the current family, but that the symbols are always typeset in the font assigned to `\fam0`.

$$\mathrm{x}^2 + \mathrm{x}^2 + \mathrm{x}^2 + \mathrm{x}^2 + \mathrm{x}^2 + \mathrm{x}^2 = 6\mathrm{x}^2$$
$$\mathbf{x^2 + x^2 + x^2 + x^2 + x^2 + x^2 = 6x^2}$$
$$\mathit{x}^2 + \mathit{x}^2 + \mathit{x}^2 + \mathit{x}^2 + \mathit{x}^2 + \mathit{x}^2 = \mathit{6x}^2$$
$$\boldsymbol{\mathit{x}^2 + \mathit{x}^2 + \mathit{x}^2 + \mathit{x}^2 + \mathit{x}^2 + \mathit{x}^2 = \mathit{6x}^2}$$
$$x^2 + x^2 + x^2 + x^2 + x^2 + x^2 = 6x^2$$
$$\boldsymbol{x^2 + x^2 + x^2 + x^2 + x^2 + x^2 = 6x^2}$$

In this example we see a new command `\mf` surface which means as much as *math font*. This commands reactivates the last font alternative and therefore equals `\bf`, `\sl` etc. but by default it equals `\tf`:

```
\def\mf{\tf}
```

The previous example was typeset saying:

```
$\tf\mf x^2 + x^2 + x^2 + x^2 + x^2 + x^2 = 6x^2$
$\bf\mf x^2 + x^2 + x^2 + x^2 + x^2 + x^2 = 6x^2$
$\sl\mf x^2 + x^2 + x^2 + x^2 + x^2 + x^2 = 6x^2$
$\bs\mf x^2 + x^2 + x^2 + x^2 + x^2 + x^2 = 6x^2$
```

font-ini
font-ans
font-ibm
font-cmr
font-con
font-eul
font-ams
font-lbr
font-pos
font-ptm
font-pcr
font-phv

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
$\it\mf x^2 + x^2 + x^2 + x^2 + x^2 + x^2 = 6x^2$
$\bi\mf x^2 + x^2 + x^2 + x^2 + x^2 + x^2 = 6x^2$
```

Beware: the exact location of `\mf` is not that important, we could as well has said

This is due to the way TEX handles fonts in math mode.

Of course we'll have to redefine `\mf` every time we change the current `\fam`.

`\mbox` `\enablembox` `\mathop`

Now how can we put this to use? Will the next sequence give the desired result?

```
$\bf x^2 + \hbox{\mf whatever} + \sin(2x)$
```

It won't!

$\mathbf{x^2} + \textbf{whatever} + \sin(\mathbf{2x})$

The reason for this is that `\sin` is defined as:

```
\def\sin{\mathop{\rm sin}\nolimits}
```

We can fix this by defining

7
```
\let\normalmathop=\mathop
```

8
```
\def\mathop%
  {\normalmathop
   \bgroup
   \let\rm=\mf
   \let\next=}
```

Of course this can be fixed, if not by a very dirty trick: redefining the TEX primitive `\hbox`:

```
\let\normalhbox=\hbox

\def\hbox%
  {\ifmmode\mbox\else\normalhbox\fi}
```

With

```
\def\mbox#1#%
  {\normalhbox#1\bgroup\mf\let\next=}
```

or more robust, that is, also accepting `\hbox\bgroup`:

```
\def\mbox%
  {\normalhbox\bgroup\mf
   \dowithnextbox{\box\nextbox\egroup}%
   \normalhbox}
```

And now:

```
$\bf x^2 + \hbox{whatever} + \sin(2x)$
```

Indeed gives:

$$\mathbf{x^2 + \text{whatever} + \sin(2x)}$$

But, do we want this kind of trickery to be activated? No, simply because we cannot be sure of incompatibilties, although for instance unboxing goes ok. Therefore we introduce:

9
```
\def\normalmbox%
  {\normalhbox\bgroup\mf
   \dowithnextbox{\box\nextbox\egroup}\normalhbox}
```

10
```
\def\mbox%
  {\ifmmode\normalmbox\else\normalhbox\fi}
```

11
```
\def\enablembox%
  {\appendtoks
     \let\normalhbox=\hbox
     \let\hbox=\mbox
   \to\everymath}
```

So in fact one can enable feature if needed. I would say: go along, but use grouping if needed!

\mrfam \mifam \syfam \exfam \bsfam \bifam \scfam \tffam \mafam \mbfam \msfam

After this short mathematical excursion, we enter the world of fonts and fontswitching. We start with something very TEX: **\fam** specified font families. TEX uses families for managing fonts in math mode. Such a family has three members: text, script and scriptscript: x^{y^z}. In CONTEXT we take a bit different approach than PLAIN TEX does. PLAIN TEX needs at least four families for typesetting math. We use those but give them symbolic names.

```
\def\mrfam{0}  %  0  (Plain TeX)  Math Roman
\def\mifam{1}  %  1  (Plain TeX)  Math Italic
\def\syfam{2}  %  2  (Plain TeX)  Math Symbol
\def\exfam{3}  %  3  (Plain TeX)  Math Extra
```

PLAIN TEX also defines families for *italic*, *slanted* and **bold** typefaces, so we don't have to define them here.

```
   \itfam  %  4  (Plain TeX)  Italic
   \slfam  %  5  (Plain TeX)  Slanted
   \bffam  %  6  (Plain TeX)  Boldface
```

Family 7 in PLAIN TEX is not used in CONTEXT, because we do massive switches from roman to sans serif, teletype or other faces.

```
   \ttfam   %  7  (Plain TeX)  can be reused!
```

We define ourselves some more families for ***bold slanted***, ***bold italic*** and Small Caps, so we can use them in math mode too. Instead of separate families for sans serif and `teletype` we use the more general `\tffam`, which stands for typeface.

13
```
\newfam\bsfam  %  8  (ConTeXt)  BoldSlanted
\newfam\bifam  %  9  (ConTeXt)  BoldItalic
\newfam\scfam  %  A  (ConTeXt)  SmallCaps
\newfam\tffam  %  B  (ConTeXt)  TypeFace
```

Normally `\mrfam` equals `\tffam`, but a more distinctive alternatives are possible, for instance the Euler and Concrete Typefaces.

After having defined all those in nature non–mathematical families, we define ourselves some real math ones. These are needed for the AMS Symbol Fonts and Extended Lucida Bright.

14
```
\newfam\mafam  %  C  (ConTeXt)  Math A Fam (AmsTeX A)
\newfam\mbfam  %  D  (ConTeXt)  Math B Fam (AmsTeX B)
\newfam\mcfam  %  E  (ConTeXt)  Math C Fam
```

Because there are 16 families and because `\ttfam` isn't used, at the moment we have two families left: 7 and F.

To ease the support of font packages, we als define shortcuts to these familynames. This is necessary because the family names are in fact `\chardef`'s, which means that we're dealing with numbers (one can check this by applying `\showthe` and `\show`). In the specification of math symbols however we need hexadecimal numbers, so we have to convert the `\fam`'s value.

15
```
\def\hexnumber#1%
  {\ifcase#1
     0\or1\or2\or3\or4\or5\or6\or7\or8\or9\or A\or B\or C\or D\or E\or F%
```

```
   \fi}
```

16
```
\edef\hexmrfam {\hexnumber\mrfam}  \edef\hexbsfam {\hexnumber\bsfam}
\edef\hexmifam {\hexnumber\mifam}  \edef\hexbifam {\hexnumber\bifam}
\edef\hexsyfam {\hexnumber\syfam}  \edef\hexscfam {\hexnumber\scfam}
\edef\hexexfam {\hexnumber\exfam}  \edef\hextffam {\hexnumber\tffam}
\edef\hexitfam {\hexnumber\itfam}  \edef\hexmafam {\hexnumber\mafam}
\edef\hexslfam {\hexnumber\slfam}  \edef\hexmbfam {\hexnumber\mbfam}
\edef\hexbffam {\hexnumber\bffam}  \edef\hexmcfam {\hexnumber\mcfam}
```

We define some (very private) constants to improve speed, memory usage and consistency.

17
```
\def\@plain@      {@f@pl@} % plain TeX encoding vector
\def\@size@       {@f@si@} % corps size prefix (12pt etc)
\def\@style@      {@f@st@} % full style prefix (roman etc)
\def\@shortstyle@ {@f@sh@} % short style prefix (rm etc)
\def\@letter@     {@f@le@} % first alternative typeface
\def\@noletter@   {@f@no@} % second alternative typeface
```

We also define a list of all text (i.e. non math symbol) families. I considered using something like:

```
   \def\familylist%
     {\do\c!tf\do\c!sl\do\c!it\do\c!bf\do\c!bs\do\c!bi\do\c!sc}
```

By assigning `\do` some suitable meaning one can process such lists quite fast. The current implementation uses the commalist processing macros and is not that slow either.

18
```
\def\familylist%
  {\c!tf,\c!sl,\c!it,\c!bf,\c!bs,\c!bi,\c!sc}
```

All used styles, like rm, ss and tt, are saved in a comma separated list. Appart from practical limitations one can define as many styles as needed, but first we

19
```
\let\stylelist=\empty
```

Further on we have to take some precautions when dealing with special characters like ~, _ and ^, so let us define ourselve some handy macros first.

20
```
\def\protectfontcharacters%
  {\catcode`\~=\@@letter
   \catcode`\_=\@@letter
   \catcode`\^=\@@letter\relax}
```

21
```
\def\unprotectfontcharacters%
  {\catcode`\~=\@@active
   \catcode`\_=\@@subscript
   \catcode`\^=\@@superscript\relax}
```

The completeness of the Computer Modern Roman typefaces makes clear how incomplete other faces are. To honour 7 bit ASCII, these fonts were designed using only the first 127 values of the 256 ones that can be presented by one byte. Nowadays 8 bit character codings are more common, mainly because they permit us to predefine some composed characters, which are needed in most european languages.

Supporting more than the standard TEX encoding vector —which in itself is far from standard and differs per font— puts a burden on the fonts mechanism. The CONTEXT mechanism is far from complete, but can handle several schemes at once. The main problem lays in the accented characters and ligatures like ff, although handling ligatures is not the responsibility of this module.

By default, we use PLAIN TEX's approach of placing accents. All other schemes sooner or later give problems when we distribute DVI–files are distributed across machines and platforms. Nevertheless, we have to take care of different encoding vectors, which tell us where to find the characters we need. This means that all kind of character placement macro's like `\"` and `\ae` have to be implemented and adapted in a way that suits these vectors.

The main difference between different vector is the way accents are ordered and/or the availability of prebuilt accented characters. Accented characters can for instance be called for by sequences like `\"e`. Here the `\"` is defined as:

```
\def\"#1{{\accent"7F #1}}
```

This macro places the accent ¨ on top of an e gives ë. Some fonts however can have prebuild accents and use a more direct approach like

```
\def\"#1{\if#1e\char 235\else ... \fi}
```

The latter approach is not used in CONTEXT, because we store relevant combinations of accents and characters in individual macros.

We define character substitutes and commands with definition commands like:

```
\startcoding[texnansi]

\defineaccent " a           228
\defineaccent ^ e           234
\defineaccent ' {\dotlessi} 237

\definecharacter ae 230
\definecharacter oe 156

\definecommand b \newansib
\definecommand c \newansic

\stopcoding
```

The last argument of `\defineaccent` and `\definecharacter` tells TEX the position of the accented character in the encoding vector. In order to complish this, we tag each implementation with the character coding identifier. We therefore need two auxiliary variables `\charactercoding` and `\nocharactercoding`. These contain the current and default encoding vectors and both default to the PLAIN one.

22
```
\let\charactercoding   = \@plain@
\let\nocharactercoding = \@plain@
```

\startcoding

Before we can redefine accents and special characters, we have to tell CONTEXT what encoding is in force. The next command is responsible for doing this and also takes care of the definition of the recoding commands.

23
```
\def\startcoding[#1]%
  {\protectfontcharacters
   \showmessage{\m!fonts}{1}{#1}%
   \def\charactercoding{@#1@}}
```

24
```
\def\stopcoding%
  {\let\charactercoding=\@plain@
   \unprotectfontcharacters}
```

\defineaccent
\definecharacter
\definecommand

The actual definition of accents, special characters and commands is done with the next three commands.

25
```
\def\defineaccent#1 #2 #3 %
  {\setvalue{\charactercoding#1\string#2}{\char#3}}%
```

26
```
\def\definecommand#1 #2 %
  {\setvalue{\charactercoding\string#1}{#2}}
```

27
```
\def\definecharacter#1 #2 %
  {\setvalue{\charactercoding\string#1}{\char#2}}
```

Here we see that redefining accents is characters is more or less the same as redefining commands. We also could have said:

```
\def\defineaccent#1 #2 {\definecommand#1\string#2 \char}
\def\definecharacter#1 {\definecommand#1 \char}
```

\redefineaccent

Telling CONTEXT how to treat accents and special characters is a two stage process. First we signal the system which commands are to be adapted, after which we can redefine their behavior when needed. We showed this in the previous paragraphs. These redefinitions are grouped at the end of this file, but we show some examples here.

Accents or accent generating commands are redefined by:

```
\redefineaccent  '  % grave
\redefineaccent  "  % dieresis
\redefineaccent  ^  % circumflex
\redefineaccent  v  % caron
```

The original PLAIN TEX meaning of each accent generating command is saved first. Next these commands are redefined to do an indirect call to a macro that acts according to the encoding vector in use.

28
```
\def\redefineaccent%
  {\protectfontcharacters
   \doredefineaccent}
```

29
```
\def\doredefineaccent#1 %
  {\def\!!stringa{\nocharactercoding\string#1}%
   \@EA\letvalue\@EA\!!stringa\@EA=\csname\string#1\endcsname
   \setvalue{\string#1}{\dohandleaccent#1}%
   \unprotectfontcharacters}
```

font-ini
font-ans
font-ibm
font-cmr
font-con
font-eul
font-ams
font-lbr
font-pos
font-ptm
font-pcr
font-phv

The next (in fact three) macros to take care of `\"e` as well as `\"{e}` situations. The latter one is always handled by TEX's `\accent` primitive, but the former one can put the accents on top of characters as well as use `\char` to call for a character directly.

30
```
\unexpanded\def\dohandleaccent#1%
  {\def\dodohandleaccent%
     {\ifx\next\bgroup
        \def\next{\getvalue{\nocharactercoding#1}}%
      \else
        \def\next{\dododohandleaccent#1}%
      \fi
      \next}%
   \futurelet\next\dodohandleaccent}
```

31
```
\def\dododohandleaccent#1#2%
  {\bgroup
   \ifundefined{\charactercoding#1\string#2}%
     \def\\{\getvalue{\nocharactercoding#1}#2\egroup}%
   \else
     \def\\{\getvalue{\charactercoding#1\string#2}\egroup}%
   \fi
   \\}
```

The trick with `\\` is needed to prevent spaces from being gobbled after the accented character, should we have `\next`, we should have ended up with gobbled spaces.

`\redefinecommand` Redefinition of encoding dependant commands like `\b` and `\c` can be triggered by:

```
   \redefinecommand  b  % something math
   \redefinecommand  c  % something math
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

Handling of characters is easier than handling accents because here we don't have to take care of arguments. We just call for the right glyph in the right place.

The **\next** construction permits handling of commands that take arguments. This means that we can use this command to redefine accent handling commands too.

32
```
\def\redefinecommand#1 %
  {\def\!!stringa{\nocharactercoding#1}%
   \@EA\letvalue\@EA\!!stringa\@EA=\csname#1\endcsname
   \setvalue{#1}{\dohandlecommand{#1}}}%
```

33
```
\unexpanded\def\dohandlecommand#1%
  {\doifdefinedelse{\charactercoding#1}
     {\def\next{\getvalue{\charactercoding#1}}}
     {\def\next{\getvalue{\nocharactercoding#1}}}%
   \next}
```

\redefinecharacter

Special characters, which differ from accented characters in that they are to be presented as they are, are redefined by

```
   \redefinecharacter  ae  % ae
   \redefinecharacter  cc  % ccedilla
```

To keep things simple, we just copy this command:

34
```
\let\redefinecharacter=\redefinecommand
```

\magfactor
\magfactorhalf

There are several ways to specify a font. Three of them are pure TEX ones, the fourth one is new:

```
\font\name=cmr12
\font\name=cmr12 at 10pt
\font\name=cmr12 scaled \magstep2
\font\name=cmr12 sa 1.440
```

The non–TEX alternative `sa` stands for *scaled at*. This means as much as: scale the corpssize with this factor. The value 1.440 in this example is derived from the `\magstep`'s as mentioned in table 6.1. We therefore introduce `\magfactor` as an alternative for `\magstep`.

magstep	equivalent	factor
1	`\magfactor1`	1.200
2	`\magfactor2`	1.440
3	`\magfactor3`	1.728
4	`\magfactor4`	2.074
5	`\magfactor5`	2.488

Table 6.1 Factors to be used with `sa`.

35
```
\def\magfactor#1%
  {\ifcase#1 1.000\or 1.200\or 1.440\or 1.728\or 2.074\or 2.488\or 1\fi}
```

36
```
\def\magfactorhalf%
  {1.095}
```

These macros enable the use of definitions like `sa \magfactor3` which saves us both (mis)calculations and potential mistypings.

font-ini
font-ans
font-ibm
font-cmr
font-con
font-eul
font-ams
font-lbr
font-pos
font-ptm
font-pcr
font-phv

contents register context syst mult supp lang font colo spec core cont m s exit go back

Because **sa** is not a TEX supported alternative, we have to test for it ourselves. In doing so, we need an auxiliary ⟨*dimension*⟩. We cannot use `\scratchdimen` because font loading can happen at any moment due to postponed loading. We could instead have used dirty grouping tricks, but this one works too.

37
```
\newdimen\scaledfont
```

38
```
\def\docalculatefont#1 sa #2sa#3*#4*#5*% The spaces are needed!
  {\edef\fontscale{#2}%
   \ifx\fontscale\empty
     \expandafter\font\csname#4#5\endcsname=#1\relax
   \else
     \scaledfont=#4\relax
     \expandafter\font\csname#4#5\endcsname=#1 at \fontscale\scaledfont\relax
   \fi}
```

I considered checking for mistakenly use of PLAIN's `\magstep`'s but although it would take only a few lines of code, this would not add to consistent use. I therefore removed this check.

39
```
\def\dodoloadfont#1#2#3%
  {\expanded{\docalculatefont\getvalue{\??ft#1#2} sa sa*#1*#2*}%
   #3\relax
   \getvalue{#1#2}}
```

A more ugly but correct alternative for this is:

```
   \def\docalculatefont#1sa #2sa#3*#4*#5*%
     {\edef\fontscale{#2}%
      \scaledfont=#4\relax
      \expandafter\font\csname#4#5\endcsname=#1
        \ifx\fontscale\empty\else at \fontscale\scaledfont\fi}
```

This one saves a few bytes of memory, but is not particular faster due to the often unneeded assignment.

The loading macro is used in two macros. One of them takes care of fixed width teletype fonts.

40
```
\def\doloadfont#1#2%
  {\debuggerinfo{\m!fonts}{loaded #1#2}%
   \dodoloadfont{#1}{#2}{}}
```

41
```
\def\doloadttfont#1#2%
  {\debuggerinfo{\m!fonts}{loaded fixed #1#2}%
   \dodoloadfont{#1}{#2}{\expandafter\hyphenchar\csname#1#2\endcsname=-1}}
```

`\getfontname` The names of the fonts can be called with the rather simple macro `\getfontname`. When for instance we pass `12ptrmtf` as argument, we get cmr12.

42
```
\def\getfontname#1%
  {\getvalue{\??ft#1}}
```

Now we enter the area of font switching. The switching mechanism has to take care of several situations, like:

- changing the overal document fonts (including margins, headers and footers)
- changing local fonts (only the running text)
- smaller and even more smaller alternatives (super- and subscripts)

TEX offers a powerfull family mechanism for super- and subscripts in math mode. In text mode however, we don't use families for the smaller alternatives, and therefore have to take care of it otherwise.

\definecorpsenvironment

The relationship between the several sizes of a font, is defined by:

[setup definieerkorpsomgeving is niet gedefinieerd]

Later on we will see how these parameters are used, so for the moment we stick with an example:

```
\definecorpsenvironment
  [12pt]
  [        text=12pt,
         script=9pt,
   scriptscript=7pt,
              x=10pt,
             xx=8pt,
            big=12pt,
          small=10pt]
```

Due to the fact that \c!text and \s!text can have a similar meaning, and therefore can lead to an unwanted loop, we temporary redefine \c!text. For the moment this in only place that some trickery is needed to fool the multilingual interface.

43
```
\def\definecorpsenvironment%
  {\dodoubleempty\dodefinecorpsenvironment}
```

44
```
\def\dodefinecorpsenvironment[#1][#2]%
  {\let\c!savedtext=\c!text
   \let\c!text=\s!text
   \doifundefined{\??ft#1\s!text}
     {\getparameters[\??ft#1]
        [\s!text=#1,\s!script=#1,\s!scriptscript=#1,
         \c!x=#1,\c!xx=#1,
         \c!groot=#1,\c!klein=#1]}%
\getparameters[\??ft#1][#2]%
\let\c!text=\c!savedtext
\setvalue{\@size@#1}{\docompletefontswitch[#1]}}
```

We default all parameters to the main corps size (begin `#1`), so the next setup is valid too:

```
\definecorpsenvironment[24pt]
```

All parameters can be redefined when needed, so one does not have to stick to the default ones.

`\definecorps` The next step in defining a corps involves the actual font files, which can be recognized by their extension `tfm`. Installing those file is often beyond the scope of the user and up to the system administrator.

```
\definieerkorps[.1.][.2.][..,..=..,..]

.1.      5pt ... 12pt
.2.      rm ss tt mm hw cg
tf       file
bf       file
sl       file
it       file
bs       file
bi       file
sc       file
ex       file
mi       file
sy       file
ma       file
mb       file
mc       file
```

We show two examples, that show all the alternative scaling options. The `\tfa` alternatives can be extended with `\bfa`, `\slb`, etc. or even `e` and higher alternatives.

```
\definecorps [12pt] [rm]
  [tf=cmr12,
   bf=cmbx12,
   it=cmti12,
   sl=cmsl12,
   bi=cmbxti10 at 12pt,
   bs=cmbxsl10 at 12pt,
  tfa=cmr12    scaled \magstep1,
  tfb=cmr12    scaled \magstep2,
  tfc=cmr12    scaled \magstep3,
  tfd=cmr12    scaled \magstep4,
   sc=cmcsc10  at 12pt]

\definecorps [12pt,11pt,10pt,9pt,8pt] [rm]
  [tf=lbr  sa 1,
   bf=lbd  sa 1,
   it=lbi  sa 1,
   sl=lbsl sa 1,
   bi=lbdi sa 1,
   bs=lbdi sa 1,
  tfa=lbr  sa 1.200,
  tfb=lbr  sa 1.440,
  tfc=lbr  sa 1.728,
  tfd=lbr  sa 2.074,
   sc=lbr  sa 0.833]
```

font-ini
font-ans
font-ibm
font-cmr
font-con
font-eul
font-ams
font-lbr
font-pos
font-ptm
font-pcr
font-phv

The second example shows that we can define more sizes at once. The main difference between these examples is that the Computer Modern Roman come in many design sizes. This means that there we cannot define them in bulk using `sa`. Instead of `rm` (roman) one can define `ss` (sans serif), `tt` (teletype), `hw` (hand written), `cg` (calygraphic) and whatever styles.

45
```
\def\definecorps%
  {\dotripleargument\dodefinecorps}
```

The first argument may be a comma separated list. This, combined with specifications using `sa` can save a lot of typing. Although all arguments should be specified, we treat the second argument as optional.

46
```
\def\dodefinecorps[#1][#2][#3]%
  {\ifthirdargument
     \def\dododefinecorps##1%
       {\dodododefinecorps[##1][#2][#3]}%
     \processcommalist[#1]\dododefinecorps
   \else
     \definecorps[#1][\c!rm][#2]%
   \fi}
```

Defining a corps involves two actions: defining the specific style related alternatives, like `\rma`, `\bfa` and `\rmsla`, and storing the definitions of their corps size related fonts. The first step is corps independant but executed every time. This permits user definitions like `\tfw` or `\bfq` for real large alterbatives.

47
```
\def\dodododefinecorps[#1][#2][#3]%
  {\geteparameters[\??ft#1#2][#3]% We expand them!
   \dodefinecorpsenvironment[#1][]% Just to be sure.
   \def\doiffamily##1##2##3\\%
     {\rawdoifinsetelse{##1##2}{\familylist}
```

```
        {\doifsomething{##3}
           {\setvalue{#2##3}% eg: \rma, \ssa
              {\donottest\switchtofontstyle{#2}{##3}}%
            \setvalue{##1##2##3}% eg: \tfa, \bfa
              {\donottest\switchtofontalternative{##1##2}{##3}}%
            \setvalue{#2##1##2##3}% eg: \rmtfa, \ssbfa
              {\donottest\switchtofontstylealternative{#2}{##1##2}{##3}}}}
        {}}%
   \def\dodefinefont##1%
     {\doifdefined{\??ft#1#2##1}
        {\letvalue{@#1#2##1@}=\charactercoding
         \doifelse{#2}{\c!tt}
           {\setvalue{#1#2##1}{\donottest\doloadttfont{#1}{#2##1}}}
           {\setvalue{#1#2##1}{\donottest\doloadfont{#1}{#2##1}}}%
         \bgroup
         \let\relax=\empty
         \debuggerinfo
           {\m!fonts}{\getvalue{\??ft#1#2##1} defined as #1 #2 ##1}%
         \egroup}%
      \expandafter\doiffamily##1\\}%
   \processassignlist[#3]\dodefinefont}
```

These macros show that quite some definitions take place. Fonts are not loeaded yet! This means that at format generation time, no font files are preloaded.

We could have use `\unexpanded\setvalue` instead of the `\donottest` prefixes. However, this would lead to about 400 extra entries in the hash table.

We can save ourselved some 400 csnames by packing the name and the encoding. But ..... not done yet .....

\everycorps
\EveryCorps

Every change in corps size has conseqences for the baseline distance and skips between paragraphs. These are initialized in other modules. Here we only provide the hooks that garantees their handling.

48
```
\newevery \everycorps \EveryCorps
```

At the system level one can initialize thing like:

```
   \appendtoks \setupspacing \to \everycorps
```

While users can add their own non standard commands like:

```
   \EveryCorps{\message{changing to corps \the\corpssize}}
```

Personnaly I never felt the need for such extensions, but at least its possible.

\globalcorpssize
\localcorpssize

Next we'll do the tough job of font switching. Here we have to distinguish between the global (overal) corps size and the local (sometimes in the textflow) size. We store these dimensions in two ⟨*dimension*⟩ registers.

49
```
\newdimen\globalcorpssize  \globalcorpssize=12pt
\newdimen\localcorpssize   \localcorpssize =\globalcorpssize
```

\corpssize

These two registers are not to be misused in calculations. For this purpose we keep a copy:

50
```
\newdimen\corpssize  \corpssize=\globalcorpssize
```

\outputresolution

Sometimes (to be honest: not in this module) we need to take the system resolution into account. Therefore we also define a macro:

51
```
\def\outputresolution {300}
```

font-ini
font-ans
font-ibm
font-cmr
font-con
font-eul
font-ams
font-lbr
font-pos
font-ptm
font-pcr
font-phv

contents register context syst mult supp lang font colo spec core cont m s exit go back

\corpsfactor
\corpspoints

For multiplication purposes we keep an auxiliary counter and macro (here the expansion is not explicitly needed):

52
```
\newcount\corpspoints \dimensiontocount\corpssize\corpspoints
```

53
```
\edef\corpsfactor{\withoutpt\the\corpssize}
```

When we assign for instance 12pt to a ⟨*dimension*⟩ register the `\the`'d value comes out as 12.0pt, which is often not the way users specifies the corps size. Therefore we also store normalized value.

54
```
\def\normalizecorpssize#1\to#2%
  {\scratchdimen=#1\relax
   \doifinstringelse{.0}{\withoutpt\the\scratchdimen}
     {\dimensiontocount\scratchdimen\scratchcounter
      \edef#2{\the\scratchcounter pt}}
     {\edef#2{\the\scratchdimen}}}
```

55
```
\normalizecorpssize\corpssize\to\normalizedglobalcorpssize
\normalizecorpssize\corpssize\to\normalizedlocalcorpssize
\normalizecorpssize\corpssize\to\normalizedcorpssize
```

To be internationalized:

56
```
\def\korpsgrootte {\corpssize}
\def\korpspunten  {\corpspoints}
```

some day.

\fontsize
\fontstyle

Within a corps, fonts can come in different sizes. For instance `\tf` is accompanied by `\tfa`, `\tfb` etc. The third character in these sequences represents the size. The actual size is saved in a macro

57
```
\let\fontsize  = \empty
```

The style, being roman (`\rm`), sans serif (`\ss`) etc. is also available in a macro in `rm`, `ss` etc. form:

58
```
\let\fontstyle = \empty
```

All things related to fonts are grouped into files with names like `font- cmr`. These files are loaded by:

59
```
\def\doreadfontdefinitionfile#1%
  {\doifundefined{\c!file\f!fontprefix#1}%
     {\setvalue{\c!file\f!fontprefix#1}{}%
      \edef\saveddqcatcode{\the\catcode`"}%
      \catcode`\"=\@@other
      \readsysfile{\f!fontprefix#1}
        {\showmessage{\m!fonts}{2}{#1}}
        {\showmessage{\m!fonts}{3}{#1}}%
      \catcode`\"=\saveddqcatcode}}
```

Such files are only loaded once! This permits redundant loading, but at the same time forced grouping when we want continuously mix all kind of font, which of course is a kind of typographically sin. The `"` is made inactibe if needed to prevent problems with loading files that use this character in numbers.

60
```
\def\doswitchpoints[#1]%
  {\expanded{\dodoswitchpoints{#1}}}
```

61
```
\def\dodoswitchpoints#1%
  {\doifdefinedelse{\@size@#1}
```

font-ini
font-ans
font-ibm
font-cmr
font-con
font-eul
font-ams
font-lbr
font-pos
font-ptm
font-pcr
font-phv

```
      {\getvalue{\@size@#1}%
       \localcorpssize=#1\relax
       \normalizecorpssize\localcorpssize\to\normalizedcorpssize
       \the\everycorps}
      {\showmessage{\m!fonts}{4}{#1}}}
```

62
```
\def\doswitchstyle[#1]%
  {\doifdefinedelse{\@style@#1}
      {\getvalue{\@style@#1}%
       \edef\fontstyle{#1}}
      {\showmessage{\m!fonts}{5}{#1}}}
```

TEX loads font metric files like `cmr10.tfm` and `tir.tfm` only once. In PLAIN TEX some font files are *preloaded*. This means that the font files are loaded, but not accessible yet by name. This is accomplished by saying:

```
   \font\preloaded=cmr10 at 11pt
```

and using the name `\preloaded` again and again, so fonts are indeed loaded, but unnamed, and therefore unaccessible. In CONTEXT we don't preload fonts, not even the PLAIN TEX ones, although users can access them. Now why is this done?

Defining fonts using `\definecorps` takes time, so we prefer to predefine at least the Computer Modern Roman fonts. However, loading all those fonts at definition time would take both time and space. But even worse, once fonts are loaded into memory, their encoding vector is fixed, which is a handicap when we want to distribute the compact `fmt` files. So what we want to do is defining fonts in a way that postpones the loading. We accomplish this by only loading the fonts when we switch to another corps size. Among the other alternatives, such as loading the font at the moment of activation and redefining the activation macro afterwards, this proved to be the most efficient alternative.

contents register context syst mult supp lang font colo spec core cont m s exit go back

The next few macros take care of the one exeption on this scheme. When at format generation time we load the default font file, the one that defines the Computer Modern Fonts, we don't want the fonts metrics to end up in the format file, so we temporary prohibit loading. This means that at runtime we have to load the default corps size just before we start typesetting.

Therefore we have to signal the font switching macros that we are preloading fonts. As long as the next boolean is, true, no loading is done.

63
```
\newif\ifloadingfonts \loadingfontstrue
```

\preloadfonts Preloading is only called for once, during the startup sequence of a session. After the loading job is done, the macro relaxes itself and reset the signal.

64
```
\def\preloadfonts%
  {\showmessage{\m!fonts}{6}{\normalizedcorpssize\normalspace\fontstyle}%
   \doswitchpoints[\normalizedcorpssize]%
   \doswitchstyle[\fontstyle]%
   \global\let\preloadfonts=\relax
   \global\loadingfontsfalse}
```

Here comes the main font switching macros. These macros handle changes in size as well as returning to the global corps size.

65
```
\def\dosetfont[#1]%
  {\doifelse{#1}{\v!globaal}
     {\restoreglobalcorps}
     {\processcommalist[#1]\dodosetfont
      \ifloadingfonts\else
        \doswitchpoints[\normalizedcorpssize]%
        \doswitchstyle[\fontstyle]%
      \fi}}
```

font-ini
font-ans
font-ibm
font-cmr
font-con
font-eul
font-ams
font-lbr
font-pos
font-ptm
font-pcr
font-phv

66
```
\def\dodosetfont#1%
  {\dododosetfont{#1}{\showmessage{\m!fonts}{4}{#1}}}
```

67
```
\def\dododosetfont#1#2%
  {\doifnumberelse{#1}
     {\scratchdimen=#1\relax
      \normalizecorpssize\scratchdimen\to\normalizedsetfont
      \doifdefinedelse{\@size@\normalizedsetfont}
        {\localcorpssize=\normalizedsetfont
         \let\normalizedcorpssize=\normalizedsetfont}
        {#2\dosetsubstitutefont{#1}}}
     {\doifdefinedelse{\@style@#1}
        {\edef\fontstyle{#1}}
        {\doreadfontdefinitionfile{#1}}}}
```

When users specify for instance a 13 point corps while no such corps is defined, the system automatically tries to find a best fit, that is the nearest smaller defined corpszize. A smaller one is definitely better than a larger one, simply because otherwise a lot of overfull box messages are more probable to occur. By taking a value slightly smaller than half a point, we can use the next method.

68
```
\def\dosetsubstitutefont#1%
  {\scratchdimen=#1\relax
   \advance\scratchdimen by .499pt
   \dimensiontocount\scratchdimen\scratchcounter
   \advance\scratchcounter by -1
   \ifnum\scratchcounter>3
     \dododosetfont{\the\scratchcounter pt}{}%
   \fi}
```

Next we're going to implement some switching macros we already used in when defining fonts. The first one takes care of the style and is used by commands like `\rma`.

contents register context syst mult supp lang font colo spec core cont m s exit go back

69
```
\def\switchtofontstyle#1#2%
  {\getvalue{#1}%
   \getvalue{\c!tf#2}}
```

The second one is responsible for commands like `\bfa` and the third one handles the combined `\rmbfa` alternatives.

70
```
\def\switchtofontalternative#1#2%
  {\def\fontsize{#2}%
   \setfontstyle{\fontstyle}{\fontstyle}%
   \donottest\getvalue{#1}}
```

71
```
\def\switchtofontstylealternative#1#2#3%
  {\getvalue{\normalizedcorpssize#1#2#3}}
```

Setting the normal sized as well as the x and xx smaller sizes is accomplished by the next set of macros. When in math mode, the commands `\tx` and `\txx` are just a switch to the script and double script styles, but in text mode the values defined by the corpsenvironment are used.

72
```
\def\dosetsomextypeface#1%
  {\doifdefinedelse{#1}
     {\getvalue{#1}}
     {\showmessage{\m!fonts}{7}{#1}}}%
```

73
```
\def\dosetxtypeface#1%
  {\ifmmode
     \scriptstyle
   \else
     \dosetsomextypeface
       {\getvalue{\??ft\normalizedcorpssize\c!x}%  % pt
        \getvalue{\@shortstyle@\fontstyle}%        % rm
        #1}%                                        % tf
     \def\tx{\dosetxxtypeface{#1}}%
   \fi}
```

74
```
\def\dosetxxtypeface#1%
  {\ifmmode
     \scriptscriptstyle
   \else
     \dosetsomextypeface
       {\getvalue{\??ft\normalizedcorpssize\c!xx}%   % pt
        \getvalue{\@shortstyle@\fontstyle}%          % rm
        #1}%                                         % tf
     \let\tx=\relax
     \let\txx=\relax
   \fi}
```

These macros also show us that when we call for `\tx`, this macro is redefined to be `\txx`. Therefore calls like:

```
   {small \tx  is \tx  beautiful}
   {small \tx  is \txx beautiful}
   {small \txx is \tx  beautiful}
   {small \txx is \txx beautiful}
```

result in:

small is beautiful
small is beautiful
small is beautiful
small is beautiful

Setting the main size involves the style list and therefore takes a bit more time. Keep in mind that the fontsize is represented by a character or empty.

75
```
\def\settextfont#1%
  {\def\dosettextfont##1%
     {\doifdefinedelse{\textface#1##1\fontsize}
        {\setvalue{#1##1}{\donottest\getvalue{\textface#1##1\fontsize}}}
        {\setvalue{#1##1}{\donottest\getvalue{\textface#1##1}}}}%
   \rawprocesscommalist[\familylist]\dosettextfont}
```

76
```
\def\settextfonts%
  {\rawprocesscommalist[\stylelist]\settextfont}
```

All three sizes come together in the macro:

77
```
\def\settypefaces#1#2%
  {\setvalue{#2}%
     {\donottest\dosettypeface{#1}{#2}}%
   \setvalue{#2\c!x}%
     {\donottest\dosetxtypeface{#2}}%
   \setvalue{#2\c!xx}%
     {\donottest\dosetxxtypeface{#2}}}%
```

Earlier in this module we defined some TEX families. Here we introduce the macros that are responsible for setting them. The first argument of the next macro takes the style in its short form (rm, ss, etc). The second argument is the alternative (tf, bf, etc).

Before actually assigning the font to a family we activate it. This is needed because loading of fonts is postponed until the first time it's called for. This also forces us to set the text family after we've set the script ones, else the latter one would be in force after executing this macro.

78
```
\def\settextfamily#1#2%
  {\def\setfamily##1##2%
     {\doifdefinedelse{##2#1#2}
        {\debuggerinfo{\m!fonts}{defined ##2#1#2}%
         \getvalue{##2#1#2}\relax % activate font
         \expandafter##1\getvalue{#2\s!fam}=\getvalue{##2#1#2}}
        {\doifdefinedelse{##2#1\c!tf}
           {\debuggerinfo{\m!fonts}{##2#1#2 replaced by ##2#1\c!tf}%
            \getvalue{##2#1\c!tf}\relax % activate font
            \expandafter##1\getvalue{#2\s!fam}=\getvalue{##2#1\c!tf}}
           {\debuggerinfo{\m!fonts}{not defined ##2#1#2}}}}%
   \scriptscriptfont\getvalue{#2\s!fam}=\scriptfont\getvalue{#2\s!fam}%
   \setfamily\scriptscriptfont\scriptscriptface
   \scriptfont\getvalue{#2\s!fam}=\textfont\getvalue{#2\s!fam}%
   \setfamily\scriptfont\scriptface
   \textfont\getvalue{#2\s!fam}=\textfont\tffam
   \setfamily\textfont\textface}
```

\defineoverallstyle When setting of switching the overall style we can use the short identifier like rm and ss, but when defined we can also use more verbose names like roman or sansserif. Such names are defined by:

```
\defineoverallstyle [roman, rm] [rm]
\defineoverallstyle [sansserif, ss] [ss]
```

79
```
\def\dodefineoverallstyle[#1][#2]%
  {\rawdoifinsetelse{#2}{\stylelist}
     {\debuggerinfo{\m!fonts}{unknown style #2}}
     {\addtocommalist{#2}\stylelist
      \showmessage{\m!fonts}{8}{#2}}%
   \setvalue{#2\c!x}%
```

font-ini font-ans font-ibm font-cmr font-con font-eul font-ams font-lbr font-pos font-ptm font-pcr font-phv

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
    {\getvalue{#2}\getvalue{\c!tf\c!x}}%
  \setvalue{#2\c!xx}%
    {\getvalue{#2}\getvalue{\c!tf\c!xx}}%
  \def\docommando##1%
    {\setvalue{\@shortstyle@##1}{#2}%
     \setvalue{\@style@##1}{\getvalue{#2}}%
     \setvalue{#2}{\donottest\setfontstyle{##1}{#2}}}%
  \processcommalist[#1]\docommando}
```

80
```
\def\defineoverallstyle%
  {\dodoubleargument\dodefineoverallstyle}
```

CHECKEN WAT `\fontstyle` HIER DOET

81
```
\def\setfontstyle#1#2%  #1:name (roman, romaan)  #2:style (rm)
  {\edef\fontstyle{#1}%
   \def\dosettextfamily##1%
     {\settextfamily{#2}{##1}%
      \settypefaces{#2}{##1}}%
   \rawprocesscommalist[\familylist]\dosettextfamily
   \fam\tffam\relax
   \def\tx{\tfx}%
   \donottest\tf}
```

Setting the math families looks much like setting the texts ones. This time however we use the 12 point font as a default when nothing is defined. This enables us to implement partial schemes. Here we also set the `\skewchar`, which takes care of accents in math mode (actually it's the largest accent). The first family needs a bit different treatment because it can be set to the default roman as well as a user defined font.

font-ini
font-ans
font-ibm
font-cmr
font-con
font-eul
font-ams
font-lbr
font-pos
font-ptm
font-pcr
font-phv

contents register context syst mult supp lang font colo spec core cont m s exit go back

82
```
\def\setmathfamilies%
  {\setskewchar{\textface\c!mm\c!mi}{'177}%
   \setskewchar{\textface\c!mm\c!sy}{'60}%
   \setmathfamily\mrfam\textface\scriptface\scriptscriptface
     {\c!mm\c!mr}{\c!rm\c!tf}%
   \setmathfamily\mifam\textface\scriptface\scriptscriptface
     {\c!mm\c!mi}{}%
   \setmathfamily\syfam\textface\scriptface\scriptscriptface
     {\c!mm\c!sy}{}%
   \setmathfamily\exfam\textface\textface\textface
     {\c!mm\c!ex}{}%
   \setmathfamily\mafam\textface\scriptface\scriptscriptface
     {\c!mm\c!ma}{}%
   \setmathfamily\mbfam\textface\scriptface\scriptscriptface
     {\c!mm\c!mb}{}%
   \setmathfamily\mcfam\textface\scriptface\scriptscriptface
     {\c!mm\c!mc}{}}
```

When setting the `\skewchar` we need to test on the availability first.

83
```
\def\setskewchar#1#2%
  {\doifdefined{#1}
     {\getvalue{#1}\expandafter\skewchar\getvalue{#1}=#2\relax}}
```

First we try to set the font at the math specific one (the fifth argment), next we take the alternative the last argument, which of often empty, and finally we default to the 12 point alternative.

84
```
\def\setmathfamily#1#2#3#4#5#6%
  {\def\dosetmathfamily##1##2%
     {\doifdefinedelse{##2#5}
        {\getvalue{##2#5}\relax
```

```
          ##1#1=\getvalue{##2#5}\relax}
         {\doifdefinedelse{##2#6}
            {\getvalue{##2#6}\relax
             ##1#1=\getvalue{##2#6}\relax}
            {\doifdefinedelse{12pt#5}
               {\getvalue{12pt#5}\relax
                ##1#1=\getvalue{12pt#5}\relax}
               {##1#1=\nullfont}}}}%
    \dosetmathfamily\scriptscriptfont{#4}%
    \dosetmathfamily\scriptfont{#3}%
    \dosetmathfamily\textfont{#2}}
```

The previous macros show that it's is not always neccessary to define the whole bunch of fonts, take for instance the sequence:

```
\setupcorps
  [ams]

\definecorps [24pt] [mm]
  [ma=msam10 at 24pt,
   mb=msbm10 at 24pt]

\switchtocorps
  [24pt]

This is a 24pt $\blacktriangleleft$
```

Here we didn't define the 24 point corps environment, so it's defined automatically. Of course one can always use the TEX primitive `\font` to switch to whatever font needed.

When asking for a complete font switch, for instance from 10 to 12 points, the next macro does the job. First we normalize the size, nect we define the current range of text, script and scriptscript sizes, then we set the text fonts and the math families and finally we activate the default typeface and also set the font specific parameters assigned to `\everycorps`

85
```
\def\docompletefontswitch[#1]%
  {\corpssize=#1\relax
   \dimensiontocount\corpssize\corpspoints
   \edef\corpsfactor{\withoutpt\the\corpssize}%
   \normalizecorpssize\corpssize\to\normalizedcorpssize
   \edef\textface{\getvalue{\??ft\normalizedcorpssize\s!text}}%
   \edef\scriptface{\getvalue{\??ft\normalizedcorpssize\s!script}}%
   \edef\scriptscriptface{\getvalue{\??ft\normalizedcorpssize\s!scriptscript}}%
   \settextfonts
   \setmathfamilies
   \rmtf
   \the\everycorps}
```

`\setupcorps` `\switchtocorps`

The next two macros are user ones. With `\setupcorps` one can set the document corps size, font family, style and/or options defined in files, for example:

```
   \setupcorps[cmr,ams,12pt,roman]
```

This command affects the document as a whole: text, headers and footers. The second macro however affects only the text:

```
   \switchtocorps[10pt]
```

So we've got:

font-ini
font-ans
font-ibm
font-cmr
font-con
font-eul
font-ams
font-lbr
font-pos
font-ptm
font-pcr
font-phv

```
\stelkorpsin[..,...,..]

...       naam romaan schreefloos teletype handschrift calligrafie 5pt ... 12pt
```

```
\switchnaarkorps[..,...,..]

...       5pt ... 12pt klein groot globaal
```

Both macros look alike. The second one also has to take keywords into account.

86
```
\def\setupcorps[#1]%
  {\doifsomething{#1}
     {\dosetfont[#1]%
      \globalcorpssize=\localcorpssize
      \normalizecorpssize\globalcorpssize\to\normalizedglobalcorpsize
      \let\globalfontstyle=\fontstyle
      \the\everycorps}}
```

87
```
\unexpanded\def\switchtocorps[#1]%
  {\doifsomething{#1}
     {\doifdefinedelse{\??ft\normalizedcorpssize\interfaced{#1}}
        {\doswitchpoints[\getvalue{\??ft\normalizedcorpssize\interfaced{#1}}]%
         \doswitchstyle[\fontstyle]}
        {\dosetfont[#1]}%
      \the\everycorps}}
```

Because the last macro can appear in arguments or be assigned to parameters, we protect this one for unwanted expansion.

contents register context syst mult supp lang font colo spec core cont m s exit go back

88
```
\def\dosetmathfont#1%
  {\def\rm{\fam0}%
   \edef\mffam{\getvalue{#1\s!fam}}%
   \textfont\mrfam=\textfont\mffam
   \scriptfont\mrfam=\scriptfont\mffam
   \scriptscriptfont\mrfam=\scriptscriptfont\mffam}
```

89
```
\def\dosettypeface#1#2%
  {\doifdefinedelse{@\normalizedcorpssize#1#2@}
     {\edef\charactercoding{\getvalue{@\normalizedcorpssize#1#2@}}}
     {\let\charactercoding=\@plain@}%
   \def\tx%
     {\dosetxtypeface{#2\fontsize}}%
   \def\txx%
     {\dosetxxtypeface{#2\fontsize}}%
   \expandafter\fam\getvalue{#2\s!fam}%
   \def\mf{\donottest{\dosetmathfont{#2}}}%
   \donottest\getvalue{#1#2}}
```

`\os` Old style numerals can be typeset with `\os` and look like 1234567890 instead of the more common looking 1234567890.

Some day this macro will be made more geneal. For the moment it's behavior is tigthly coupled to the Computer Modern Roman.

90
```
\def\os%
  {\getvalue{\normalizedcorpssize\c!mm\c!mi}}
```

font-ini
font-ans
font-ibm
font-cmr
font-con
font-eul
font-ams
font-lbr
font-pos
font-ptm
font-pcr
font-phv

contents register context syst mult supp lang font colo spec core cont m s exit go back

\definecorpsswitch

PLAIN TEX defines some macro's like `\tenpoint` to switch to a specific corpssize. Just for the sake of compatibility we can define them like:

```
\definecorpsswitch [twelvepoint] [12pt]
```

We don't support language specific synonyms here, mainly because PLAIN TEX is english anyway.

91
```
\def\dodefinecorpsswitch[#1][#2]%
  {\def\docommando##1%
     {\setvalue{##1}{\switchtocorps[#2]}}%
   \processcommalist[#1]\docommando}
```

92
```
\def\definecorpsswitch%
  {\dodoubleargument\dodefinecorpsswitch}
```

\setsmallcorps
\setmaincorps
\setbigcorps

When we're typesetting at for instance 10pt, we can call for the `small` as well as the `big` alternative, related to this main size, using `\switchtocorps[small]`. The three alternatives can be activated by the next three system calls and are defined by the corpsenvironment.

93
```
\def\setsmallcorps%
  {\doswitchpoints[\getvalue{\??ft\normalizedcorpssize\v!klein}]%
   \doswitchstyle[\fontstyle]}
```

94
```
\def\setmaincorps%
  {\doswitchpoints[\normalizedcorpssize]%
   \doswitchstyle[\fontstyle]}
```

95
```
\def\setbigcorps%
  {\doswitchpoints[\getvalue{\??ft\normalizedcorpssize\v!groot}]%
   \doswitchstyle[\fontstyle]}
```

\restoreglobalcorps

Users can set whatever font available while typesetting text. Pagenumnbers, footers, headers etc. however must be typeset in the main corps and style of the document. Returning to the global state can be done with the next macro:

96
```
\def\restoreglobalcorps%
  {\doswitchpoints[\normalizedglobalcorpsize]%
   \doswitchstyle[\globalfontstyle]%
   \let\fontsize=\empty
   \tf}
```

This macro has to be called when entering the pagebody handling routine as well as the footnote insert routine. Users can access this feature —for instance when one wants to typeset tables and alike in the main corps and style while the running text is temporary set to a smaller one— by saying `\switchtocorps[global]`.

\rasterfont

There are (at the moment) two situations in which we want to have fast access to a particular font. When we are using TEX to typeset rasters, we use small .'s (a rather small period indeed), the same as PICTEX uses for drawing purposes.

97
```
\def\rasterfont%
  {\getvalue{\!!fivepoint\c!rm\c!tf}}
```

\infofont

The second situation occurs when we enable the info mode, and put all kind of status information in the margin. We don't want huge switches to the main corps and style, so here too we use a direct method.

98
```
\def\infofont%
  {\getvalue{\!!sixpoint\c!tt\c!tf}%
   \the\everycorps}
```

font-ini font-ans font-ibm font-cmr font-con font-eul font-ams font-lbr font-pos font-ptm font-pcr font-phv

contents register context syst mult supp lang font colo spec core cont m s exit go back

\definealternativestyle

In the main modules we are going to implement lots of parameterized commands and one of these parameters will concern the font to use. To suit consistent use of fonts we here implement a mechanism for defining the keywords that present a particular style or alternative.

```
\definealternativestyle [keywords] [\style] [\nostyle]
```

The first command is used in the normal textflow, while the second command takes care of headings and alike. Consider the next two definitions:

```
\definealternativestyle [bold] [\bf]  []
\definealternativestyle [cap]  [\kap] [\kap]
```

A change `\bf` in a heading which is to be set in `\tfd` does not look that well, so therefore we leave the second argument of `\definealternativestyle` empty. When we capatalize characters using the pseudo small cap command `\kap`, we want this to take effect in both text and headings, which is accomplished by assigning both arguments.

99
```
\def\dodefinealternativestyle[#1][#2][#3]%
  {\def\docommando##1%
     {\doifundefined{##1}
        {\expandafter\ifx\csname##1\endcsname#2\else
           \setvalue{##1}{\groupedcommand{#2}{}}%
         \fi}%
      \setvalue{\@letter@##1}{#2}%
      \setvalue{\@noletter@##1}{#3}}%
   \processcommalist[#1]\docommando}
```

100
```
\def\definealternativestyle%
  {\dotripleargument\dodefinealternativestyle}
```

This command also defines the keyword as command. This means that the example definition of `bold` we gave before, results in a command `\bold` which can be used as:

```
He's a \bold{man} man with a {\bold head}.
```

or

He's a **man** man with a **head**.

Such definitions are of course unwanted for `\kap` because this would result in an endless recursive call. Therefore we check on the existance of both the command and the substitution. The latter is needed because for instance `\type` is an entirely diferent command. That command handles verbatim, while the style command would just switch to teletype font. This is just an example of a tricky naming coincidence.

\doconvertfont
\noconvertfont
\dontconvertfont

After having defined such keywords, we can call for them by using

```
\doconvertfont{keyword}{text}
```

We deliberately pass an argument. This enables us to assign converters that handle one agrument, like `\kap`.

By default the first specification is used to set the style, exept when we say `\dontconvertfont`, after which the second specification is used. We can also directly call for `\noconvertfont`.

101

```
\unexpanded\def\doconvertfont#1#2%
  {\doifdefinedelse{\@letter@#1}
     {\doifelsenothing{#1}
        {\def\next{}}
        {\def\next{\getvalue{\@letter@#1}}}}
     {\doifdefinedelse{#1}
        {\def\next{\getvalue{#1}}}
        {\def\next{#1}}}%
   \next{#2}}
```

102
```
\def\noconvertfont#1#2%
  {\doifdefinedelse{\@noletter@#1}
     {\doifelsenothing{#1}
        {\def\next{}}
        {\def\next{\getvalue{\@noletter@#1}}}}
     {\def\next{#1}}%
   \next{#2}}
```

103
```
\unexpanded\def\dontconvertfont%
  {\let\doconvertfont=\noconvertfont}
```

These commands are not grouped! Grouping is most probably done by the calling macro's and would lead to unnecessary overhead.

\em \emphasistypeface \emphasisboldface

The next macro started as a copy of Donald Arseneau's `\em` (TUG NEWS Vol. 3, no. 1, 1994). His implementation was a bit more sophisticated version of the standard LaTeX one. We furter enhanced the macro, so now it also adapts itself to boldface mode. Because we favor *slanted* type over *italic*, we made the emphasis adaptable, for instance:

```
   \def\emphasistypeface {\it}
   \def\emphasisboldface {\bi}
```

But we prefer:

104
```
\def\emphasistypeface {\sl}
\def\emphasisboldface {\bs}
```

105
```
\unexpanded\def\em%
  {\ifnum\fam=\itfam
     \def\emphasistypeface{\it}\tf
   \else\ifnum\fam=\slfam
```

font-ini font-ans font-ibm font-cmr font-con font-eul font-ams font-lbr font-pos font-ptm font-pcr font-phv

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
     \def\emphasistypeface{\sl}\tf
   \else\ifnum\fam=\bffam
     \emphasisboldface
   \else\ifnum\fam=\bsfam
     \def\emphasisboldface{\bs}\bf
   \else\ifnum\fam=\bifam
     \def\emphasisboldface{\bi}\bf
   \else
     \emphasistypeface
   \fi\fi\fi\fi
   \ifdim\fontdimen1\font>\!!zeropoint
     \expandafter\aftergroup
   \fi
   \emphasiscorrection}
```

Donald's (adapted) macros take the next character into account when placing italic correction. As a bonus we also look for something that looks like a dash, in which case we don't correct.

106
```
\def\emphasiscorrection%
  {\ifhmode
     \expandafter\emphasislook
   \fi}
```

107
```
\def\emphasislook%
  {\begingroup
   \futurelet\next\emphasistest}
```

108
```
\def\emphasistest%
  {\ifcat\noexpand\next,%
     \setbox\scratchbox=\hbox{\next}%
     \ifdim\ht\scratchbox<.3ex
```

```
        \let\doemphasiscorrection\endgroup
      \fi
    \fi
    \doemphasiscorrection}
```

109
```
\def\doemphasiscorrection%
  {\scratchskip=\lastskip
   \ifdim\scratchskip=\!!zeropoint
     \/\relax
   \else
     \unskip\/\hskip\scratchskip
   \fi
   \endgroup}
```

We end with some examples which show the behavior when some punctuation is met. We also show how the mechanism adapts itself to bold, italic and slanted typing.

```
   test {test}test       \par
   test {\em test}test   \par
   test {\em test}--test \par

   test {test}, test     \par
   test {\em test}, test \par

   test {\em test {\em test {\em test} test} test} test \par
   test {\bf test {\em test {\em test} test} test} test \par
   test {\sl test {\em test {\em test} test} test} test \par
   test {\it test {\em test {\em test} test} test} test \par
```

We get:

font-ini
font-ans
font-ibm
font-cmr
font-con
font-eul
font-ams
font-lbr
font-pos
font-ptm
font-pcr
font-phv

contents register context syst mult supp lang font colo spec core cont m s exit go back

test testtest
test *test*test
test *test*–test
test test, test
test *test*, test
test *test* test *test* test *test* test
test **test** ***test*** **test** ***test*** **test** test
test *test* test *test* test *test* test
test *test* test *test* test *test* test

`\showcorps` One can call for a rather simple overview of a corps and the relations between its alternative fonts.

```
\toonkorps[..,...,..]

...      zie
```

The current corps (here we omitted the argument) looks like:

[9.0pt]													
	\tf	\sc	\sl	\it	\bf	\bs	\bi	\tfx	\tfxx	\tfa	\tfb	\tfc	\tfd
\rm	Ag	AG	*Ag*	*Ag*	**Ag**	***Ag***	***Ag***	Ag	Ag	Ag	Ag	Ag	Ag
\ss	Ag	Ag	*Ag*	*Ag*	**Ag**	**Ag**	**Ag**	Ag	Ag	Ag	Ag	Ag	Ag
\tt	Ag	Ag	*Ag*	*Ag*	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag

The implementation is rather straightforward in using `\halign`.

110

```
\def\doshowcorps[#1]%
  {\startruledboxcorrection
   \vbox
     {\def\bigstrut##1##2%
        {\hbox{\vrule
           \!!height ##1\ht\strutbox
           \!!depth  ##2\dp\strutbox
           \!!width  \!!zeropoint}}
      \doifelsenothing{#1}
        {\def\title{\the\korpsgrootte}}
        {\switchtocorps[#1]\def\title{#1}}
      \tabskip\!!zeropoint
      \parindent\!!zeropoint
      \def\next##1##2##3%
        {&&##1&&##2\tf##3&&##2\sc##3%
         &&##2\sl##3&&##2\it##3&&##2\bf##3&&##2\bs##3&&##2\bi##3%
         &&##2\tfx##3&&##2\tfxx##3%
         &&##2\tfa##3&&##2\tfb##3&&##2\tfc##3&&##2\tfd##3&\cr}%
      \setlocalhsize
      \halign to \localhsize
        {\bigstrut{1.5}{2}##&\vrule##
         \tabskip=\!!zeropoint \!!plus 1fill
         &\hfil##\hfil&\vrule##&\hfil##\hfil&\vrule##
         &\hfil##\hfil&\vrule##&\hfil##\hfil&\vrule##
         &\hfil##\hfil&\vrule##&\hfil##\hfil&\vrule##
         &\hfil##\hfil&\vrule##&\hfil##\hfil&\vrule##
         &\hfil##\hfil&\vrule##&\hfil##\hfil&\vrule##
         &\hfil##\hfil&\vrule##&\hfil##\hfil&\vrule##
         &\hfil##\hfil&\vrule##&\hfil##\hfil&##\vrule
```

```
          \tabskip=\!!zeropoint\cr
          \noalign{\hrule}
          &\multispan{29}{\vrule\hfil\tttf\strut[\title]\hfil\vrule}\cr
          \noalign{\hrule}\next{}{\tt\string}{}
          \noalign{\hrule}\next{\tt\string\rm}{\rm}{Ag}
          \noalign{\hrule}\next{\tt\string\ss}{\ss}{Ag}
          \noalign{\hrule}\next{\tt\string\tt}{\tt}{Ag}
          \noalign{\hrule}}
    \stopruledboxcorrection}
```

111
```
\def\showcorps%
  {\dosingleempty\doshowcorps}
```

\showcorpsenvironment

The current corpsenvironment is:

[9.0pt]						
text	**script**	**scriptscript**	**x**	**xx**	**klein**	**groot**
12pt	9pt	7pt	10pt	8pt	10pt	14.4pt
11pt	8pt	6pt	9pt	7pt	9pt	12pt
10pt	7pt	5pt	8pt	6pt	8pt	12pt
9pt	7pt	5pt	7pt	5pt	7pt	11pt
8pt	6pt	5pt	6pt	5pt	6pt	10pt
7pt	6pt	5pt	6pt	5pt	5pt	9pt
6pt	5pt	5pt	5pt	5pt	5pt	8pt
5pt	5pt	5pt	5pt	5pt	5pt	7pt
4pt	4pt	4pt	4pt	4pt	4pt	6pt

This overview is generated using:

```
\toonkorpsomgeving[..,...,..]

...     zie
```

112
```
\def\doshowcorpsenvironment[#1]%
  {\startruledboxcorrection
   \vbox
     {\tabskip\!!zeropoint
      \parindent\!!zeropoint
      \doifelsenothing{#1}
        {\def\title{\the\korpsgrootte}}
        {\switchtocorps[#1]\def\title{#1}}
      \def\do##1##2%
        {\getvalue{\??ft##1##2}}
      \def\next##1##2%
        {&&##1{##2}{\s!text}&&##1{##2}{\s!script}&&##1{##2}{\s!scriptscript}%
         &&##1{##2}{\c!x}&&##1{##2}{\c!xx}%
         &&##1{##2}{\v!klein}&&##1{##2}{\v!groot}&\cr
         \noalign{\hrule}}
      \def\donext##1%
        {\doifdefined{\??ft##1\s!text}{\next\do##1}}
      \setlocalhsize
      \halign to \localhsize
        {##&\vrule##\strut
         \tabskip=\!!zeropoint \!!plus 1fill
         &\hfil##\hfil&\vrule##&\hfil##\hfil&\vrule##&\hfil##\hfil&\vrule##
         &\hfil##\hfil&\vrule##&\hfil##\hfil&\vrule##&\hfil##\hfil&\vrule##
         &\hfil##\hfil&##\vrule
        \tabskip=\!!zeropoint\cr
        \noalign{\hrule}
        &\multispan{15}{\vrule\hfil\tttf\strut[\title]\hfil}\vrule\cr
        \noalign{\hrule}
        \next\bf\relax
        \donext\!!twelvepoint\donext\!!elevenpoint\donext\!!tenpoint
        \donext\!!ninepoint  \donext\!!eightpoint \donext\!!sevenpoint
        \donext\!!sixpoint   \donext\!!fivepoint  \donext\!!fourpoint}}
   \stopruledboxcorrection}
```

113
```
\def\showcorpsenvironment%
  {\dosingleempty\doshowcorpsenvironment}
```

Fonts can only be used when loaded. In CONTEXT we postpone the loading of fonts, even when we load PLAIN. This means that we have to redefine one of the PLAIN macros. Let's tell that to the user first:

114
```
\writestatus{loading}{Postponed Plain TeX Font Definitions}
```

`\bordermatrix` In PLAIN TEX the width of a parenthesis is stored in the ⟨*dimension*⟩ `\p@renwd`. This value is derived from the width of `\tenrm B`, so let's take care of it now:

115
```
\let\normalbordermatrix=\bordermatrix
```

116
```
\def\bordermatrix%
  {\bgroup
   \setbox0=\hbox{\getvalue{\textface\c!mm\c!ex}B}%
   \global\p@renwd=\wd0\relax
   \egroup
   \normalbordermatrix}
```

Because we want to be as PLAIN compatible as possible, we make most of PLAIN's font mechanisme available to the CONTEXT user.

117
```
\def\setplainfonts#1#2%
  {\setvalue{ten#1}{\getvalue{\!!tenpoint#2}}%
   \setvalue{seven#1}{\getvalue{\!!sevenpoint#2}}%
   \setvalue{five#1}{\getvalue{\!!fivepoint#2}}}
```

118
```
\setplainfonts {\c!rm} {\c!rm\c!tf}
\setplainfonts {\c!bf} {\c!rm\c!bf}
\setplainfonts {\c!sl} {\c!rm\c!sl}
\setplainfonts {\c!it} {\c!rm\c!it}
\setplainfonts {\c!tt} {\c!rm\c!tt}
\setplainfonts {\c!sy} {\c!mm\c!sy}
\setplainfonts {\c!ex} {\c!mm\c!ex}
\setplainfonts {\c!i}  {\c!mm\c!mi}
```

119
```
\let\setplainfonts=\undefined
```

\ss
\SS

We are going to redefine `\ss` but for those wo still want to have access to the german ß, we save it's value in `\SS`. Ok, I should have used `\sf` insead of `\ss` in the first place.

120
```
\let\SS=\ss
```

\dotlessi
\dotlessj

We also save both dotless ı and ȷ. This way we still have them were we expect them, even when macros of font providers redefine them.

121
```
\let\dotlessi=\i
\let\dotlessj=\j
```

Here come the definitions.

122
```
\redefineaccent    '   % grave
\redefineaccent    `   % acute
\redefineaccent    "   % dieresis
\redefineaccent    ^   % circumflex
\redefineaccent    ~   % tilde
\redefineaccent    v   % caron
\redefineaccent    u   % breve
\redefineaccent    .   % dotaccent
\redefineaccent    H   % hungarumlaut
\redefineaccent    t   % ........
```

123
```
\redefinecharacter ae  % ae
\redefinecharacter AE  % AE
\redefinecharacter oe  % oe
\redefinecharacter OE  % OE
\redefinecharacter o   % oslash
\redefinecharacter O   % Oslash
\redefinecharacter ss  % germandbls
\redefinecharacter SS  % germandbls
\redefinecharacter aa  % aring
\redefinecharacter AA  % Aring
\redefinecharacter cc  % ccedilla
\redefinecharacter CC  % Ccedilla
```

124
```
\redefinecommand   b
\redefinecommand   c
```

125
```
\definecorpsenvironment
  [14.4pt]
  [        \s!text=14.4pt,
```

font-ini
font-ans
font-ibm
font-cmr
font-con
font-eul
font-ams
font-lbr
font-pos
font-ptm
font-pcr
font-phv

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
           \s!script=\!!elevenpoint,
     \s!scriptscript=\!!ninepoint,
                \c!x=\!!twelvepoint,
               \c!xx=\!!tenpoint,
            \c!groot=14.4pt,
            \c!klein=\!!twelvepoint]
```

126
```
\definecorpsenvironment
  [\!!elevenpoint]
  [         \s!text=\!!elevenpoint,
          \s!script=\!!eightpoint,
    \s!scriptscript=\!!sixpoint,
               \c!x=\!!ninepoint,
              \c!xx=\!!sevenpoint,
           \c!groot=\!!twelvepoint,
           \c!klein=\!!ninepoint]
```

127
```
\definecorpsenvironment
  [\!!tenpoint]
  [         \s!text=\!!tenpoint,
          \s!script=\!!sevenpoint,
    \s!scriptscript=\!!fivepoint,
               \c!x=\!!eightpoint,
              \c!xx=\!!sixpoint,
           \c!groot=\!!twelvepoint,
           \c!klein=\!!eightpoint]
```

128
```
\definecorpsenvironment
  [\!!ninepoint]
  [         \s!text=\!!ninepoint,
         \s!script=\!!sevenpoint,
   \s!scriptscript=\!!fivepoint,
              \c!x=\!!sevenpoint,
             \c!xx=\!!fivepoint,
          \c!groot=\!!elevenpoint,
          \c!klein=\!!sevenpoint]
```

129
```
\definecorpsenvironment
  [\!!eightpoint]
  [        \s!text=\!!eightpoint,
         \s!script=\!!sixpoint,
   \s!scriptscript=\!!fivepoint,
              \c!x=\!!sixpoint,
             \c!xx=\!!fivepoint,
          \c!groot=\!!tenpoint,
          \c!klein=\!!sixpoint]
```

130
```
\definecorpsenvironment
  [\!!sevenpoint]
  [        \s!text=\!!sevenpoint,
         \s!script=\!!sixpoint,
   \s!scriptscript=\!!fivepoint,
              \c!x=\!!sixpoint,
             \c!xx=\!!fivepoint,
          \c!groot=\!!ninepoint,
          \c!klein=\!!fivepoint]
```

131
```
\definecorpsenvironment
  [\!!sixpoint]
  [        \s!text=\!!sixpoint,
          \s!script=\!!fivepoint,
    \s!scriptscript=\!!fivepoint,
               \c!x=\!!fivepoint,
              \c!xx=\!!fivepoint,
           \c!groot=\!!eightpoint,
           \c!klein=\!!fivepoint]
```

132
```
\definecorpsenvironment
  [\!!fivepoint]
  [         \s!text=\!!fivepoint,
          \s!script=\!!fivepoint,
    \s!scriptscript=\!!fivepoint,
               \c!x=\!!fivepoint,
              \c!xx=\!!fivepoint,
           \c!groot=\!!sevenpoint,
           \c!klein=\!!fivepoint]
```

133
```
\definecorpsenvironment
  [\!!fourpoint]
  [         \s!text=\!!fourpoint,
          \s!script=\!!fourpoint,
    \s!scriptscript=\!!fourpoint,
               \c!x=\!!fourpoint,
              \c!xx=\!!fourpoint,
           \c!groot=\!!sixpoint,
           \c!klein=\!!fourpoint]
```

134
```
\definecorpsswitch [twelvepoint] [\!!twelvepoint]
\definecorpsswitch [elevenpoint] [\!!elevenpoint]
\definecorpsswitch [tenpoint]    [\!!tenpoint]
```

font-ini
font-ans
font-ibm
font-cmr
font-con
font-eul
font-ams
font-lbr
font-pos
font-ptm
font-pcr
font-phv

```
    \definecorpsswitch [ninepoint]   [\!!ninepoint]
    \definecorpsswitch [eightpoint]  [\!!eightpoint]
    \definecorpsswitch [sevenpoint]  [\!!sevenpoint]
    \definecorpsswitch [sixpoint]    [\!!sixpoint]
    \definecorpsswitch [fivepoint]   [\!!fivepoint]
    \definecorpsswitch [fourpoint]   [\!!fourpoint]

135 \definecorpsswitch [xii]         [\!!twelvepoint]
    \definecorpsswitch [xi]          [\!!elevenpoint]
    \definecorpsswitch [x]           [\!!tenpoint]
    \definecorpsswitch [ix]          [\!!ninepoint]
    \definecorpsswitch [viii]        [\!!eightpoint]
    \definecorpsswitch [vii]         [\!!sevenpoint]
    \definecorpsswitch [vi]          [\!!sixpoint]

136 \defineoverallstyle [\v!romaan,      \c!rm] [\c!rm]
    \defineoverallstyle [\v!schreefloos, \c!ss] [\c!ss]
    \defineoverallstyle [\v!type,        \c!tt] [\c!tt]
    \defineoverallstyle [\v!handschrift, \c!hw] [\c!hw]
    \defineoverallstyle [\v!calligrafie, \c!cg] [\c!cg]

137 \definealternativestyle [\v!mediaeval]                    [\os]  []
    \definealternativestyle [\v!normaal]                      [\tf]  []
    \definealternativestyle [\v!vet]                          [\bf]  []
    \definealternativestyle [\v!type]                         [\tt]  []
    \definealternativestyle [\v!schuin]                       [\sl]  []
    \definealternativestyle [\v!vetschuin,\v!schuinvet]       [\bs]  []
    \definealternativestyle [\v!klein,\v!kleinnormaal]        [\tfx] []
    \definealternativestyle [\v!kleinvet]                     [\bfx] []
    \definealternativestyle [\v!kleintype]                    [\ttx] []
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
\definealternativestyle [\v!kleinschuin]                       [\slx] []
\definealternativestyle [\v!kleinvetschuin,\v!kleinschuinvet] [\bsx] []
\definealternativestyle [\v!kap,\v!kapitaal]                   [\kap] [\kap]
```

By default we load the Computer Modern Roman fonts and activate the 12pt roman corps. Sans serif and teletype are also available and can be called for by `\ss` and `\tt`.

We also load the high ASCII waarde as defined by the standard IBM PC codepage. Finaly we load the POSTSCRIPT standard predefined accented characters encoding vector as provided by Y&Y named **texnansi**. These are for instance used when we load Lucida Bright (**lbr**) or POSTSCRIPT Times Roman (**ptr**), Helvetica (**phv**) and Courier (**pcr**) which are also available as whole (`\setupcorps[pos]`).

```
\setupcorps [cmr, 12pt, \v!romaan, ibm, ans]
```

```
\protect
```

6.2 Y&Y texnansi Encoding

This is Y&Y's texnansi encoding vector, which combines the best of the ansi encoding vector (prebuilt accented characters etc.) and some of TEX's vectors.

1
```
\startcoding[texnansi]
```

2
```
\defineaccent " a 228
\defineaccent " e 235
\defineaccent " i 239
\defineaccent " o 246
\defineaccent " u 252
\defineaccent " y 255
```

3
```
\defineaccent " A 196
\defineaccent " E 203
\defineaccent " I 207
\defineaccent " O 214
\defineaccent " U 220
\defineaccent " Y 159
```

4
```
\defineaccent ' a 225
\defineaccent ' e 233
\defineaccent ' i 237
\defineaccent ' o 243
\defineaccent ' u 250
\defineaccent ' y 253
```

5
```
\defineaccent ' A 193
\defineaccent ' E 201
\defineaccent ' I 205
```

```
   \defineaccent ' O 211
   \defineaccent ' U 218
   \defineaccent ' Y 221

6  \defineaccent ` a 224
   \defineaccent ` e 232
   \defineaccent ` i 236
   \defineaccent ` o 242
   \defineaccent ` u 249

7  \defineaccent ` A 192
   \defineaccent ` E 200
   \defineaccent ` I 204
   \defineaccent ` O 210
   \defineaccent ` U 217

8  \defineaccent ^ a 226
   \defineaccent ^ e 234
   \defineaccent ^ i 238
   \defineaccent ^ o 244
   \defineaccent ^ u 251

9  \defineaccent ^ A 194
   \defineaccent ^ E 202
   \defineaccent ^ I 206
   \defineaccent ^ O 212
   \defineaccent ^ U 219

10 \defineaccent ~ a 227
   \defineaccent ~ n 241
   \defineaccent ~ o 245
```

11
```
\defineaccent ~ A 195
\defineaccent ~ N 209
\defineaccent ~ O 213
```

12
```
\defineaccent ` {\dotlessi} 236
\defineaccent ' {\dotlessi} 237
\defineaccent " {\dotlessi} 239
\defineaccent ^ {\dotlessi} 238
```

13
```
\definecharacter ae 230
\definecharacter oe 156
\definecharacter o  248
\definecharacter AE 198
\definecharacter OE 140
\definecharacter O  216
\definecharacter ss 223
```

14
```
\definecharacter aa 229
\definecharacter AA 197
```

15
```
\definecharacter cc 231
\definecharacter CC 199
```

Het onderstaande zou toch fraaier moeten kunnen, maar ja ...

16
```
\def\newansib#1{\oalign{#1\crcr\hidewidth
  \vbox to.2ex{\hbox{\char175}\vss}\hidewidth}}
```

17
```
\def\newansic#1{\setbox0\hbox{#1}\ifdim\ht0=1ex\accent184 #1%
  \else{\ooalign{\hidewidth\char184\hidewidth\crcr\unhbox0}}\fi}
```

18
```
\definecommand b {\newansib}
\definecommand c {\newansic}
```

We still have to take care of:

```
   \bgroup
     \catcode146=\active
     \gdef^^92{{^\bgroup\prim@s}}
   \egroup

   \mathcode146="8000

   \chardef\i=105
```

We have to redefine some commands too:

19
```
\redefinecommand grave  \definecommand grave {\mathaccent"7060 }
\redefinecommand acute  \definecommand acute {\mathaccent"70B4 }
\redefinecommand hat    \definecommand hat   {\mathaccent"7088 }
\redefinecommand tilde  \definecommand tilde {\mathaccent"7098 }
\redefinecommand ddot   \definecommand ddot  {\mathaccent"70A8 }
\redefinecommand bar    \definecommand bar   {\mathaccent"70AF }
```

20
```
\stopcoding
```

21
```
\endinput
```

6.3 IBM Keys

This module activates the IBM PC high ASCII characters, such as ë and ß.

```
\unprotect

\catcode`\ë=\@@active \unexpanded\defë{\"e}
\catcode`\é=\@@active \unexpanded\defé{\'e}
\catcode`\è=\@@active \unexpanded\defè{\`e}
\catcode`\ê=\@@active \unexpanded\defê{\^e}

\catcode`\ä=\@@active \unexpanded\defä{\"a}
\catcode`\á=\@@active \unexpanded\defá{\'a}
\catcode`\à=\@@active \unexpanded\defà{\`a}
\catcode`\â=\@@active \unexpanded\defâ{\^a}

\catcode`\ö=\@@active \unexpanded\defö{\"o}
\catcode`\ó=\@@active \unexpanded\defó{\'o}
\catcode`\ò=\@@active \unexpanded\defò{\`o}
\catcode`\ô=\@@active \unexpanded\defô{\^o}

\catcode`\ï=\@@active \unexpanded\defï{\"\dotlessi}
\catcode`\í=\@@active \unexpanded\defí{\'\dotlessi}
\catcode`\ì=\@@active \unexpanded\defì{\`\dotlessi}
\catcode`\î=\@@active \unexpanded\defî{\^\dotlessi}

\catcode`\ü=\@@active \unexpanded\defü{\"u}
\catcode`\ú=\@@active \unexpanded\defú{\'u}
\catcode`\ù=\@@active \unexpanded\defù{\`u}
\catcode`\û=\@@active \unexpanded\defû{\^u}
```

7
```
\catcode`\É=\@@active \unexpanded\defÉ{\'E}
\catcode`\Ä=\@@active \unexpanded\defÄ{\"A}
\catcode`\Ü=\@@active \unexpanded\defÜ{\"U}
```

8
```
\catcode`\ç=\@@active \unexpanded\defç{\c c}
\catcode`\Ç=\@@active \unexpanded\defÇ{\c C}
```

9
```
\catcode`\ñ=\@@active \unexpanded\defñ{\~n}
```

10
```
\catcode`\ß=\@@active \unexpanded\defß{\SS}
```

11
```
\catcode`\«=\@@active
\catcode`\»=\@@active
```

12
```
\unexpanded\def«{\ifvmode\leavevmode\fi\leftguillemot\prewordbreak}
\unexpanded\def»{\prewordbreak\rightguillemot}
```

13
```
\protect
```

6.4 Computer Modern

The Computer Modern Roman is derived from the Monotype 8a Times Roman. In this module, that is loaded by default, we define all relevant alternatives.

1
```
\definecorps [12pt] [rm]
  [tf=cmr12,
   bf=cmbx12,
   it=cmti12,
   sl=cmsl12,
   bi=cmbxti10 at 12pt,
   bs=cmbxsl10 at 12pt,
  tfa=cmr12    scaled \magstep1,
  tfb=cmr12    scaled \magstep2,
  tfc=cmr12    scaled \magstep3,
  tfd=cmr12    scaled \magstep4,
   sc=cmcsc10  at 12pt]
```

2
```
\definecorps [12pt] [ss]
  [tf=cmss12,
   bf=cmssbx10 at 12pt,
   it=cmssi12,
   sl=cmssi12,
   bi=cmssbx10 at 12pt,
   bs=cmssbx10 at 12pt,
  tfa=cmss12   scaled \magstep1,
  tfb=cmss12   scaled \magstep2,
  tfc=cmss12   scaled \magstep3,
  tfd=cmss12   scaled \magstep4,
   sc=cmss10]
```

3
```
\definecorps [12pt] [tt]
  [tf=cmtt12,
   sl=cmsltt10 at 12pt,
   it=cmitt10  at 12pt,
  tfa=cmtt12   scaled \magstep1,
  tfb=cmtt12   scaled \magstep2,
  tfc=cmtt12   scaled \magstep3,
  tfd=cmtt12   scaled \magstep4]
```

4
```
\definecorps [12pt] [mm]
  [ex=cmex10   at 12pt,
   mi=cmmi12,
   sy=cmsy10   at 12pt]
```

5
```
\definecorps [12pt] [hw]
  [tf=cmtt12]
```

6
```
\definecorps [12pt] [cg]
  [tf=cmtt12]
```

7
```
\definecorps [11pt] [rm]
  [tf=cmr10    at 11pt,
   bf=cmbx10   at 11pt,
   sl=cmsl10   at 11pt,
   it=cmti10   at 11pt,
   bi=cmbxti10 at 11pt,
   bs=cmbxsl10 at 11pt,
  tfa=cmr9     scaled \magstep2,
  tfb=cmr9     scaled \magstep3,
  tfc=cmr9     scaled \magstep4,
  tfd=cmr9     scaled \magstep5,
   sc=cmcsc10  at 11pt]
```

8
```
\definecorps [11pt] [ss]
  [tf=cmss10   at 11pt,
   bf=cmssbx10 at 11pt,
   it=cmssi10  at 11pt,
   sl=cmssi10  at 11pt,
   bi=cmssbx10 at 11pt,
   bs=cmssbx10 at 11pt,
  tfa=cmss9    scaled \magstep2,
  tfb=cmss9    scaled \magstep3,
  tfc=cmss9    scaled \magstep4,
  tfd=cmss9    scaled \magstep5,
   sc=cmss9]
```

9
```
\definecorps [11pt] [tt]
  [tf=cmtt10   at 11pt,
   sl=cmsltt10 at 11pt,
   it=cmitt10  at 11pt]
```

10
```
\definecorps [11pt] [mm]
  [ex=cmex10   at 11pt,
   mi=cmmi10   at 11pt,
   sy=cmsy10   at 11pt]
```

11
```
\definecorps [11pt] [hw]
  [tf=cmtt10   at 11pt]
```

12
```
\definecorps [11pt] [cg]
  [tf=cmtt10   at 11pt]
```

13
```
\definecorps [10pt] [rm]
  [tf=cmr10,
   bf=cmbx10,
   it=cmti10,
   sl=cmsl10,
   bi=cmbxti10,
   bs=cmbxsl10,
  tfa=cmr10   scaled \magstep1,
  tfb=cmr10   scaled \magstep2,
  tfc=cmr10   scaled \magstep3,
  tfd=cmr10   scaled \magstep4,
   sc=cmcsc10]
```

14
```
\definecorps [10pt] [ss]
  [tf=cmss10,
   bf=cmssbx10,
   it=cmssi10,
   sl=cmssi10,
   bi=cmssbx10,
   bs=cmssbx10,
  tfa=cmss10   scaled \magstep1,
  tfb=cmss10   scaled \magstep2,
  tfc=cmss10   scaled \magstep3,
  tfd=cmss10   scaled \magstep4,
   sc=cmss8]
```

15
```
\definecorps [10pt] [tt]
  [tf=cmtt10,
   sl=cmsltt10,
   it=cmitt10,
  tfa=cmtt10   scaled \magstep1,
  tfb=cmtt10   scaled \magstep2,
  tfc=cmtt10   scaled \magstep3,
  tfd=cmtt10   scaled \magstep4]
```

16
```
\definecorps [10pt] [mm]
  [ex=cmex10,
   mi=cmmi10,
   sy=cmsy10]
```

17
```
\definecorps [10pt] [hw]
  [tf=cmtt10]
```

18
```
\definecorps [10pt] [cg]
  [tf=cmtt10]
```

19
```
\definecorps [9pt] [rm]
  [tf=cmr9,
   bf=cmbx9,
   it=cmti9,
   sl=cmsl9,
   bi=cmbxti10 at 9pt,
   bs=cmbxsl10 at 9pt,
  tfa=cmr9     scaled \magstep1,
  tfb=cmr9     scaled \magstep2,
  tfc=cmr9     scaled \magstep3,
  tfd=cmr9     scaled \magstep4,
   sc=cmcsc10  at 9pt]
```

20
```
\definecorps [9pt] [ss]
  [tf=cmss9,
    bf=cmssbx10 at 9pt,
    it=cmssi9,
    sl=cmssi9,
    bi=cmssbx10 at 9pt,
    bs=cmssbx10 at 9pt,
   tfa=cmss9    scaled \magstep1,
   tfb=cmss9    scaled \magstep2,
   tfc=cmss9    scaled \magstep3,
   tfd=cmss9    scaled \magstep4,
    sc=cmss10   at 7pt]
```

21
```
\definecorps [9pt] [tt]
  [tf=cmtt9,
   sl=cmsltt10 at 9pt,
   it=cmitt10  at 9pt]
```

22
```
\definecorps [9pt] [mm]
  [ex=cmex10   at 9pt,
   mi=cmmi9,
   sy=cmsy9]
```

23
```
\definecorps [9pt] [hw]
  [tf=cmtt9]
```

24
```
\definecorps [9pt] [cg]
  [tf=cmtt9]
```

25
```
\definecorps [8pt] [rm]
  [tf=cmr8,
   bf=cmbx8,
   it=cmti8,
    sl=cmsl8,
    bi=cmbxti10 at 8pt,
    bs=cmbxsl10 at 8pt,
   tfa=cmr8     scaled \magstep1,
   tfb=cmr8     scaled \magstep2,
   tfc=cmr8     scaled \magstep3,
   tfd=cmr8     scaled \magstep4,
    sc=cmcsc10  at 8pt]
```

26
```
\definecorps [8pt] [ss]
  [tf=cmss8,
    bf=cmssbx10 at 8pt,
    it=cmssi8,
    sl=cmssi8,
    bi=cmssbx10 at 8pt,
    bs=cmssbx10 at 8pt,
   tfa=cmss8   scaled \magstep1,
   tfb=cmss8   scaled \magstep2,
   tfc=cmss8   scaled \magstep3,
   tfd=cmss8   scaled \magstep4,
    sc=cmss10   at 6pt]
```

27
```
\definecorps [8pt] [tt]
  [tf=cmtt8,
    sl=cmsltt10 at 8pt,
    it=cmitt10  at 8pt]
```

28
```
\definecorps [8pt] [mm]
  [ex=cmex10   at 8pt,
    mi=cmmi8,
   sy=cmsy8]
```

29
```
\definecorps [8pt] [hw]
  [tf=cmtt8]
```

30
```
\definecorps [8pt] [cg]
  [tf=cmtt8]
```

31
```
\definecorps [7pt] [rm]
  [tf=cmr7,
   bf=cmbx7,
   it=cmti10   at 7pt,
   sl=cmsl10   at 7pt,
   bi=cmbxti10 at 7pt,
   bs=cmbxsl10 at 7pt,
  tfa=cmr7     scaled \magstep1,
  tfb=cmr7     scaled \magstep2,
  tfc=cmr7     scaled \magstep3,
  tfd=cmr7     scaled \magstep4,
   sc=cmcsc10  at 7pt]
```

32
```
\definecorps [7pt] [ss]
  [tf=cmss10   at 7pt,
   bf=cmssbx10 at 7pt,
   it=cmssi10  at 7pt,
   sl=cmssi10  at 7pt,
   bs=cmssbx10 at 7pt,
   bi=cmssbx10 at 7pt,
  tfa=cmss8, % scaled 1000,
  tfb=cmss8    scaled \magstep1,
  tfc=cmss8    scaled \magstep2,
  tfd=cmss8    scaled \magstep3,
   sc=cmss10   at 5pt]
```

33
```
\definecorps [7pt] [tt]
  [tf=cmtt10   at 7pt,
   sl=cmsltt10 at 7pt,
   it=cmitt10  at 7pt]
```

34
```
\definecorps [7pt] [mm]
  [ex=cmex10   at 7pt,
   mi=cmmi7,
   sy=cmsy7]
```

35
```
\definecorps [6pt] [rm]
  [tf=cmr6,
   bf=cmbx6,
   it=cmti10   at 6pt,
   sl=cmsl10   at 6pt,
   bi=cmbxti10 at 6pt,
   bs=cmbxsl10 at 6pt,
  tfa=cmr6     scaled \magstep1,
  tfb=cmr6     scaled \magstep2,
  tfc=cmr6     scaled \magstep3,
  tfd=cmr6     scaled \magstep4,
   sc=cmcsc10  at 6pt]
```

36
```
\definecorps [6pt] [ss]
  [tf=cmss10   at 6pt,
   bf=cmssbx10 at 6pt,
   it=cmssi10  at 6pt,
   sl=cmssi10  at 6pt,
```

font-ini
font-ans
font-ibm
font-cmr
font-con
font-eul
font-ams
font-lbr
font-pos
font-ptm
font-pcr
font-phv

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
    bs=cmssbx10 at 6pt,
    bi=cmssbx10 at 6pt,
    sc=cmss10   at 4pt]
```

37
```
\definecorps [6pt] [tt]
   [tf=cmtt10   at 6pt,
    sl=cmsltt10 at 6pt,
    it=cmitt10  at 6pt]
```

38
```
\definecorps [6pt] [mm]
   [ex=cmex10   at 6pt,
    mi=cmmi6,
    sy=cmsy6]
```

39
```
\definecorps [5pt] [rm]
   [tf=cmr5,
    bf=cmbx5,
    it=cmti10   at 5pt,
    sl=cmsl10   at 5pt,
    bi=cmbxti10 at 5pt,
    bs=cmbxsl10 at 5pt,
   tfa=cmr5     scaled \magstep1,
   tfb=cmr5     scaled \magstep2,
   tfc=cmr5     scaled \magstep3,
   tfd=cmr5     scaled \magstep4,
    sc=cmcsc10  at 5pt]
```

40
```
\definecorps [5pt] [ss]
   [tf=cmss10   at 5pt,
    bf=cmssbx10 at 5pt,
    it=cmssi10  at 5pt,
    sl=cmssi10  at 5pt,
    bs=cmssbx10 at 5pt,
    bi=cmssbx10 at 5pt,
    sc=cmss10   at 3pt]
```

41
```
\definecorps [5pt] [tt]
   [tf=cmtt10   at 5pt,
    sl=cmsltt10 at 5pt,
    it=cmitt10  at 5pt]
```

42
```
\definecorps [5pt] [mm]
   [ex=cmex10   at 5pt,
    mi=cmmi5,
    sy=cmsy5]
```

43
```
\definecorps [4pt] [rm]
   [tf=cmr10    at 4pt,
    bf=cmbx10   at 4pt,
    it=cmti10   at 4pt,
    sl=cmsl10   at 4pt,
    bi=cmbxti10 at 4pt,
    bs=cmbxsl10 at 4pt,
    sc=cmr10    at 4pt]
```

44
```
\definecorps [4pt] [ss]
   [tf=cmss10   at 4pt,
    bf=cmssbx10 at 4pt,
    it=cmssi10  at 4pt,
    sl=cmssi10  at 4pt,
    bs=cmssbx10 at 4pt,
    bi=cmssbx10 at 4pt,
   sc=cmss10   at 4pt]
```

45
```
\definecorps [4pt] [tt]
  [tf=cmtt10   at 4pt,
   sl=cmsltt10 at 4pt,
   it=cmitt10  at 4pt]
```

46
```
\definecorps [4pt] [mm]
  [ex=cmex10   at 4pt,
   mi=cmmi10   at 4pt,
   sy=cmsy10   at 4pt]
```

We also define some large alternatives that can be used for titlepages and section headings.

47
```
\definecorps [12pt] [rm]
  [bfa=cmbx12 scaled \magstep1,
   bfb=cmbx12 scaled \magstep2,
   bfc=cmbx12 scaled \magstep3,
   bfd=cmbx12 scaled \magstep4,
   sla=cmsl12 scaled \magstep1,
   slb=cmsl12 scaled \magstep2,
   slc=cmsl12 scaled \magstep3,
   sld=cmsl12 scaled \magstep4,
   bsa=cmbxsl12 scaled \magstep1,
   bsb=cmbxsl12 scaled \magstep2,
   bsc=cmbxsl12 scaled \magstep3,
   bsd=cmbxsl12 scaled \magstep4]
```

48
```
\definecorps [11pt] [rm]
  [bfa=cmbx9 scaled \magstep2,
   bfb=cmbx9 scaled \magstep3,
   bfc=cmbx9 scaled \magstep4,
   bfd=cmbx9 scaled \magstep5,
   sla=cmsl9 scaled \magstep2,
   slb=cmsl9 scaled \magstep3,
   slc=cmsl9 scaled \magstep4,
   sld=cmsl9 scaled \magstep5]
```

49
```
\definecorps [10pt] [rm]
  [bfa=cmbx10 scaled \magstep1,
   bfb=cmbx10 scaled \magstep2,
   bfc=cmbx10 scaled \magstep3,
   bfd=cmbx10 scaled \magstep4,
   sla=cmsl10 scaled \magstep1,
   slb=cmsl10 scaled \magstep2,
   slc=cmsl10 scaled \magstep3,
   sld=cmsl10 scaled \magstep4,
   bsa=cmbxsl10 scaled \magstep1,
   bsb=cmbxsl10 scaled \magstep2,
   bsc=cmbxsl10 scaled \magstep3,
   bsd=cmbxsl10 scaled \magstep4]
```

50
```
\definecorps [9pt] [rm]
  [bfa=cmbx9 scaled \magstep1,
   bfb=cmbx9 scaled \magstep2,
   bfc=cmbx9 scaled \magstep3,
   bfd=cmbx9 scaled \magstep4,
   sla=cmsl9 scaled \magstep1,
   slb=cmsl9 scaled \magstep2,
   slc=cmsl9 scaled \magstep3,
   sld=cmsl9 scaled \magstep4]
```

51
```
\definecorps [8pt] [rm]
  [bfa=cmbx8 scaled \magstep1,
   bfb=cmbx8 scaled \magstep2,
   bfc=cmbx8 scaled \magstep3,
   bfd=cmbx8 scaled \magstep4]
```

52
```
\definecorps [7pt] [rm]
  [bfa=cmbx7 scaled \magstep1,
   bfb=cmbx7 scaled \magstep2,
   bfc=cmbx7 scaled \magstep3,
   bfd=cmbx7 scaled \magstep4]
```

53
```
\definecorps [6pt] [rm]
  [bfa=cmbx6 scaled \magstep1,
   bfb=cmbx6 scaled \magstep2,
   bfc=cmbx6 scaled \magstep3,
   bfd=cmbx6 scaled \magstep4]
```

54
```
\definecorps [12pt] [ss]
  [bfa=cmss12 scaled \magstep1,
   bfb=cmss12 scaled \magstep2,
   bfc=cmss12 scaled \magstep3,
   bfd=cmss12 scaled \magstep4,
   sla=cmssi10 scaled \magstep2,
   slb=cmssi10 scaled \magstep3,
   slc=cmssi10 scaled \magstep4,
   sld=cmssi10 scaled \magstep5,
   bsa=cmssi10 scaled \magstep2,
   bsb=cmssi10 scaled \magstep3,
   bsc=cmssi10 scaled \magstep4,
   bsd=cmssi10 scaled \magstep5]
```

55
```
\definecorps [10pt] [ss]
  [bfa=cmss10 scaled \magstep1,
   bfb=cmss10 scaled \magstep2,
   bfc=cmss10 scaled \magstep3,
   bfd=cmss10 scaled \magstep4,
   sla=cmssi10 scaled \magstep1,
   slb=cmssi10 scaled \magstep2,
   slc=cmssi10 scaled \magstep3,
   sld=cmssi10 scaled \magstep4,
   bsa=cmssi10 scaled \magstep1,
   bsb=cmssi10 scaled \magstep2,
   bsc=cmssi10 scaled \magstep3,
   bsd=cmssi10 scaled \magstep4]
```

56
```
\definecorps [12pt] [tt]
  [sla=cmsltt10 scaled \magstep2,
   slb=cmsltt10 scaled \magstep3,
   slc=cmsltt10 scaled \magstep4,
   sld=cmsltt10 scaled \magstep5]
```

57
```
\definecorps [10pt] [tt]
  [sla=cmsltt10 scaled \magstep1,
   slb=cmsltt10 scaled \magstep2,
   slc=cmsltt10 scaled \magstep3,
   sld=cmsltt10 scaled \magstep4]
```

6.5 Concrete Roman

The Concrete Modern Roman is just an alternative Computer Modern Roman.

1
```
\definecorps [12pt] [rm]
  [tf=ccr10   at 12pt,   % scaled \magstep1
   it=ccti10  at 12pt,   % scaled \magstep1
   sl=ccsl10  at 12pt,   % scaled \magstep1
   sc=cccsc10 at 12pt]   % scaled \magstep1
```

2
```
\definecorps [11pt] [rm]
  [tf=ccr10   at 11pt,   % scaled \magstephalf
   it=ccti10  at 11pt,   % scaled \magstephalf
   sl=ccsl10  at 11pt,   % scaled \magstephalf
   sc=cccsc10 at 11pt]   % scaled \magstephalf
```

3
```
\definecorps [10pt] [rm]
  [tf=ccr10,
   it=ccti10,
   sl=ccsl10,
   sc=cccsc10]
```

4
```
\definecorps [9pt] [rm]
  [tf=ccr9,
   it=ccr9,
   sl=ccr9,
   sc=ccr9]
```

5
```
\definecorps [8pt] [rm]
  [tf=ccr8,
   it=ccr8,
   sl=ccr8,
   sc=ccr8]
```

6
```
\definecorps [7pt] [rm]
  [tf=ccr7,
   it=ccr7,
   sl=ccr7,
   sc=ccr7]
```

7
```
\definecorps [6pt] [rm]
  [tf=ccr6,
   it=ccr6,
   sl=ccr6,
   sc=ccr6]
```

8
```
\definecorps [5pt] [rm]
  [tf=ccr5,
   it=ccr5,
   sl=ccr5,
   sc=ccr5]
```

6.6 Euler

The Euler Fonts are designed by Herman Zapf and can be used with the Concrete Fonts defined elsewhere.

1
```
\definecorps [12pt] [mm]      % scaled \magstep1
  [mi=eurm10 at 12pt,
   ex=euex10 at 12pt,
   ma=euex10 at 12pt,
   mb=eusm10 at 12pt,
   mc=eufm10 at 12pt]
```

2
```
\definecorps [11pt] [mm]      % scaled \magstephalf
  [mi=eurm10 at 11pt,
   ex=euex10 at 11pt,
   ma=euex10 at 11pt,
   mb=eusm10 at 11pt,
   mc=eufm10 at 11pt]
```

3
```
\definecorps [10pt] [mm]
  [mi=eurm10,
   ex=euex10,
   ma=euex10,
   mb=eusm10,
   mc=eufm10]
```

4
```
\definecorps [9pt] [mm]
  [mi=eurm10 at 9pt,
   ex=euex10 at 9pt,
   ma=euex10 at 9pt,
   mb=eusm10 at 9pt,
   mc=eufm10 at 9pt]
```

5
```
\definecorps [8pt] [mm]
  [mi=eurm7  at 8pt,
   ex=euex10 at 8pt,
   ma=euex10 at 8pt,
   mb=eusm7  at 8pt,
   mc=eufm7  at 8pt]
```

6
```
\definecorps [7pt] [mm]
  [mi=eurm7,
   ex=euex10 at 7pt,
   ma=euex10 at 7pt,
   mb=eusm7,
   mc=eufm7]
```

7
```
\definecorps [6pt] [mm]
  [mi=eurm7  at 6pt,
   ex=euex10 at 6pt,
   ma=euex10 at 6pt,
   mb=eusm7  at 6pt,
   mc=eufm7  at 6pt]
```

8
```
\definecorps [5pt] [mm]
  [mi=eurm5,
   ex=euex10 at 5pt,
   ma=euex10 at 5pt,
   mb=eusm5,
   mc=eufm5]
```

font-ini
font-ans
font-ibm
font-cmr
font-con
font-eul
font-ams
font-lbr
font-pos
font-ptm
font-pcr
font-phv

contents register context syst mult supp lang font colo spec core cont m s exit go back

Here we copy part of the files that are distributed along with these fonts, but first we define some extra families.

9
```
\let\exfam=\mafam   % was A
\let\smfam=\mbfam   % was 8
\let\fmfam=\mcfam   % was 9
```

10
```
\let\hexexfam=\hexmafam
\let\hexsmfam=\hexmbfam
\let\hexfmfam=\hexmcfam
```

Now we're up to the redefinitions.

11
```
\mathcode`0="7130
\mathcode`1="7131
\mathcode`2="7132
\mathcode`3="7133
\mathcode`4="7134
\mathcode`5="7135
\mathcode`6="7136
\mathcode`7="7137
\mathcode`8="7138
\mathcode`9="7139
```

12
```
\mathchardef\intop          ="1\hexexfam 52
\mathchardef\ointop         ="1\hexexfam 48
\mathchardef\coprod         ="1\hexexfam 60
\mathchardef\prod           ="1\hexexfam 51
\mathchardef\sum            ="1\hexexfam 50
\mathchardef\braceld        ="\hexexfam 7A
\mathchardef\bracerd        ="\hexexfam 7B
```

```
\mathchardef\bracelu         ="\hexexfam 7C
\mathchardef\braceru         ="\hexexfam 7D
\mathchardef\infty           ="0\hexexfam 31
```

13
```
\mathchardef\nearrow         ="3\hexexfam 25
\mathchardef\searrow         ="3\hexexfam 26
\mathchardef\nwarrow         ="3\hexexfam 2D
\mathchardef\swarrow         ="3\hexexfam 2E
\mathchardef\Leftrightarrow ="3\hexexfam 2C
\mathchardef\Leftarrow       ="3\hexexfam 28
\mathchardef\Rightarrow      ="3\hexexfam 29
\mathchardef\leftrightarrow ="3\hexexfam 24
\mathchardef\leftarrow       ="3\hexexfam 20
\mathchardef\rightarrow      ="3\hexexfam 21
```

14
```
\let\gets =\leftarrow
\let\to   =\rightarrow
```

15
```
\mathcode`\^^W               ="3\hexexfam 24
\mathcode`\^^X               ="3\hexexfam 20
\mathcode`\^^Y               ="3\hexexfam 21
\mathcode`\^^K               ="3\hexexfam 22
\mathcode`\^^A               ="3\hexexfam 23
```

16
```
\def\uparrow                 {\delimiter"3\hexexfam 22378 }
\def\downarrow               {\delimiter"3\hexexfam 23379 }
\def\updownarrow             {\delimiter"3\hexexfam 6C33F }
\def\Uparrow                 {\delimiter"3\hexexfam 2A37E }
\def\Downarrow               {\delimiter"3\hexexfam 2B37F }
\def\Updownarrow             {\delimiter"3\hexexfam 6D377 }
```

17
```
\mathchardef\leftharpoonup    ="3\hexexfam 18
\mathchardef\leftharpoondown  ="3\hexexfam 19
\mathchardef\rightharpoonup   ="3\hexexfam 1A
\mathchardef\rightharpoondown ="3\hexexfam 1B
```

18
```
\mathcode`+="2\hexfmfam 2B
\mathcode`-="2\hexfmfam 2D
\mathcode`!="0\hexfmfam 21
\mathcode`(="4\hexfmfam 28    \delcode`(="\hexfmfam 28300
\mathcode`)="5\hexfmfam 29    \delcode`)="\hexfmfam 29301
\mathcode`[="4\hexfmfam 5B    \delcode`[="\hexfmfam 5B302
\mathcode`]="5\hexfmfam 5D    \delcode`]="\hexfmfam 5D303
\mathcode`=="3\hexfmfam 3D
```

19
```
\mathchardef\Relbar   ="303D % we need the old = to match \Arrows
\mathchardef\Gamma    ="7100
\mathchardef\Delta    ="7101
\mathchardef\Theta    ="7102
\mathchardef\Lambda   ="7103
\mathchardef\Xi       ="7104
\mathchardef\Pi       ="7105
\mathchardef\Sigma    ="7106
\mathchardef\Upsilon  ="7107
\mathchardef\Phi      ="7108
\mathchardef\Psi      ="7109
\mathchardef\Omega    ="710A
```

20
```
\let\varsigma         =\sigma % Euler doesn't have these
\let\varrho           =\rho   % Euler doesn't have these
\mathchardef\aleph    ="0D40
```

```
21  \def\rbrace           {\delimiter"5\hexsmfam 67A09 } \let\}=\rbrace
    \def\lbrace           {\delimiter"4\hexsmfam 66A08 } \let\{=\lbrace

22  \mathchardef\leq      ="3\hexsmfam 14 \let\le=\leq
    \mathchardef\geq      ="3\hexsmfam 15 \let\ge=\geq
    \mathchardef\Re       ="0\hexsmfam 3C
    \mathchardef\Im       ="0\hexsmfam 3D

23  \def\vert             {\delimiter"\hexsmfam 6A30C }
    \def\backslash        {\delimiter"\hexsmfam 6E30F }
```

6.7 AMS Math Symbols

Here we implement the symbol fonts as provided by the American Mathematical Society. The names of these symbols can be found in The Joy of TEX by M. Spivak.

First we extend the already defined font sets a bit. We make use of the **sa** option.

1
```
\definecorps [14.4pt,12pt,11pt,10pt,9pt] [mm]
  [ma=msam10 sa 1,
   mb=msbm10 sa 1]
```

2
```
\definecorps [8pt,7pt] [mm]
  [ma=msam7 sa 1,
   mb=msbm7 sa 1]
```

3
```
\definecorps [6pt,5pt,4pt] [mm]
  [ma=msam5 sa 1,
   mb=msbm5 sa 1]
```

We already have defined some additional math families. This means that do not have to do this again. It would exhaust our limited pool of `\fam`'s anyway.

4
```
\unprotect
```

5
```
\let\msafam@=\hexmafam
\let\msbfam@=\hexmbfam
```

6
```
\protect
```

The following piece of TEX is part of the distribution of the AMS fonts. The macros are slightly adapted to the CONTEXT way of font handling, which means that we have commented out some sections. The comments are original.

```
%% @texfile{
%%     filename="amssym.def",
%%     version="2.1",
%%     date="5-APR-1991",
%%     filetype="TeX: option",
%%     copyright="Copyright (C) American Mathematical Society,
%%            all rights reserved.  Copying of this file is
%%            authorized only if either:
%%            (1) you make absolutely no changes to your copy
%%                including name; OR
%%            (2) if you do make changes, you first rename it to some
%%                other name.",
%%     author="American Mathematical Society",
%%     address="American Mathematical Society,
%%            Technical Support Department,
%%            P. O. Box 6248,
%%            Providence, RI 02940,
%%            USA",
%%     telephone="401-455-4080 or (in the USA) 800-321-4AMS",
%%     email="Internet: Tech-Support@Math.AMS.org",
%%     codetable="ISO/ASCII",
%%     checksumtype="line count",
%%     checksum="108",
%%     keywords="amsfonts, tex",
%%     abstract="This file contains definitions that perform the same
%%            functions as similar ones in AMS-TeX, so that the file
%%            AMSSYM.TEX can be used outside of AMS-TeX. Instructions
%%            for using this file and the AMS symbol fonts are
%%            included in the AMSFonts 2.0 User's Guide."
```

font-ini
font-ans
font-ibm
font-cmr
font-con
font-eul
font-ams
font-lbr
font-pos
font-ptm
font-pcr
font-phv

```
  %%      }
```

7
```
\expandafter\ifx\csname amssym.def\endcsname\relax \else\endinput\fi
```

Store the catcode of the @ in the csname so that it can be restored later.

8
```
\expandafter\edef\csname amssym.def\endcsname%
  {\catcode`\noexpand\@=\the\catcode`\@\normalspace}
```

Set the catcode to 11 for use in private control sequence names.

9
```
\catcode`\@=11
```

Include all definitions related to the fonts msam, msbm and eufm, so that when this file is used by itself, the results with respect to those fonts are equivalent to what they would have been using $\mathcal{AMS}$-TEX. Most symbols in fonts msam and msbm are defined using `\newsymbol`; however, a few symbols that replace composites defined in plain must be defined with `\mathchardef`.

10
```
\def\undefine#1%
  {\let#1\undefined}
```

11
```
\def\newsymbol#1#2#3#4#5%
  {\let\next@\relax
   \ifnum#2=\@ne
     \let\next@\msafam@
   \else
     \ifnum#2=\tw@
       \let\next@\msbfam@
     \fi
   \fi
   \mathchardef#1="#3\next@#4#5}
```

font-ini
font-ans
font-ibm
font-cmr
font-con
font-eul
font-ams
font-lbr
font-pos
font-ptm
font-pcr
font-phv

contents register context syst mult supp lang font colo spec core cont m s exit go back

12
```
\def\mathhexbox@#1#2#3%
  {\relax
   \ifmmode
     \mathpalette{}{\m@th\mathchar"#1#2#3}%
   \else
     \leavevmode
     \hbox{$\m@th\mathchar"#1#2#3$}%
   \fi}

   \def\hexnumber@#1%
     {\ifcase#1
        0\or 1\or 2\or 3\or
        4\or 5\or 6\or 7\or
        8\or 9\or A\or B\or
        C\or D\or E\or F\fi}

   \font\tenmsa=msam10
   \font\sevenmsa=msam7
   \font\fivemsa=msam5
   \newfam\msafam
   \textfont\msafam=\tenmsa
   \scriptfont\msafam=\sevenmsa
   \scriptscriptfont\msafam=\fivemsa

   \edef\msafam@%
     {\hexnumber@\msafam}
```

13
```
\mathchardef\dabar@"0\msafam@39
```

14
```
\def\dashrightarrow  {\mathrel{\dabar@\dabar@\mathchar"0\msafam@4B}}
\def\dashleftarrow   {\mathrel{\mathchar"0\msafam@4C\dabar@\dabar@}}
```

```
\let\dasharrow       \dashrightarrow
\def\ulcorner        {\delimiter"4\msafam@70\msafam@70 }
\def\urcorner        {\delimiter"5\msafam@71\msafam@71 }
\def\llcorner        {\delimiter"4\msafam@78\msafam@78 }
\def\lrcorner        {\delimiter"5\msafam@79\msafam@79 }
\def\yen             {{\mathhexbox@\msafam@55 }}
\def\checkmark       {{\mathhexbox@\msafam@58 }}
\def\circledR        {{\mathhexbox@\msafam@72 }}
\def\maltese         {{\mathhexbox@\msafam@7A }}

   \font\tenmsb=msbm10
   \font\sevenmsb=msbm7
   \font\fivemsb=msbm5
   \newfam\msbfam
   \textfont\msbfam=\tenmsb
   \scriptfont\msbfam=\sevenmsb
   \scriptscriptfont\msbfam=\fivemsb

   \edef\msbfam@%
     {\hexnumber@\msbfam}
```

15
```
\def\Bbb#1%
  {{\fam\msbfam\relax#1}}
```

16
```
\def\widehat#1%
  {\setbox\z@\hbox{$\m@th#1$}%
   \ifdim\wd\z@>\tw@ em%
     \mathaccent"0\msbfam@5B{#1}%
   \else
     \mathaccent"0362{#1}%
```

```
    \fi}
```

17
```
\def\widetilde#1%
  {\setbox\z@\hbox{$\m@th#1$}%
    \ifdim\wd\z@>\tw@ em%
      \mathaccent"0\msbfam@5D{#1}%
    \else
      \mathaccent"0365{#1}%
    \fi}

    \font\teneufm=eufm10
    \font\seveneufm=eufm7
    \font\fiveeufm=eufm5
    \newfam\eufmfam
    \textfont\eufmfam=\teneufm
    \scriptfont\eufmfam=\seveneufm
    \scriptscriptfont\eufmfam=\fiveeufm
    \def\frak#1{{\fam\eufmfam\relax#1}}
    \let\goth\frak
```

Restore the catcode value for @ that was previously saved.

18
```
\csname amssym.def\endcsname

    %% @texfile{
    %%     filename="amssym.tex",
    %%     version="2.1a",
    %%     date="31-OCT-1991",
    %%     filetype="TeX: option",
    %%     copyright="Copyright (C) American Mathematical Society,
    %%            all rights reserved.  Copying of this file is
```

font-ini
font-ans
font-ibm
font-cmr
font-con
font-eul
font-ams
font-lbr
font-pos
font-ptm
font-pcr
font-phv

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
%%             authorized only if either:
%%             (1) you make absolutely no changes to your copy
%%                 including name; OR
%%             (2) if you do make changes, you first rename it to some
%%                 other name.",
%%      author="American Mathematical Society",
%%      address="American Mathematical Society,
%%             Technical Support Department,
%%             P. O. Box 6248,
%%             Providence, RI 02940,
%%             USA",
%%      telephone="401-455-4080 or (in the USA) 800-321-4AMS",
%%      email="Internet: Tech-Support@Math.AMS.org",
%%      codetable="ISO/ASCII",
%%      checksumtype="line count",
%%      checksum="279",
%%      keywords="amstex, ams-tex, tex, amsfonts, math symbols",
%%      abstract="This file defines names for all the math symbols in
%%             the math symbol fonts of the AMSFonts package (MSAM and
%%             MSBM). If this file is not used by way of the AMS-TeX
%%             \UseAMSsymbols command, it must be used in conjunction
%%             with AMSSYM.DEF, which provides a definition of the
%%             \newsymbol and \undefine commands.
%%             Instructions for using the AMS symbol fonts are
%%             included in: AMS-TeX 2.1 User's Guide; AMSFonts 2.1
%%             User's Guide; The Joy of TeX, editions dated 1990 or
%%             later."
%%      }
```

font-ini
font-ans
font-ibm
font-cmr
font-con
font-eul
font-ams
font-lbr
font-pos
font-ptm
font-pcr
font-phv

contents register context syst mult supp lang font colo spec core cont m s exit go back

Save the current value of the @-sign catcode so that it can be restored afterwards. This allows us to call amssym.tex either within an $\mathcal{AMS}$-TEX document style file or by itself, in addition to providing a means of testing whether the file has been previously loaded. We want to avoid inputting this file twice because when $\mathcal{AMS}$-TEX is being used `\newsymbol` will give an error message if used to define a control sequence name that is already defined.

If the csname is not equal to `\relax`, we assume this file has already been loaded and `\endinput` immediately.

19
```
\expandafter\ifx\csname pre amssym.tex at\endcsname\relax \else \endinput\fi
```

Otherwise we store the catcode of the @ in the csname.

20
```
\expandafter\chardef\csname pre amssym.tex at\endcsname=\the\catcode`\@
```

Set the catcode to 11 for use in private control sequence names.

21
```
\catcode`\@=11
```

Most symbols in fonts msam and msbm are defined using `\newsymbol`. A few that are delimiters or otherwise require special treatment have already been defined as soon as the fonts were loaded. Finally, a few symbols that replace composites defined in plain must be undefined first.

22
```
\newsymbol\boxdot 1200
\newsymbol\boxplus 1201
\newsymbol\boxtimes 1202
\newsymbol\square 1003
\newsymbol\blacksquare 1004
\newsymbol\centerdot 1205
\newsymbol\lozenge 1006
\newsymbol\blacklozenge 1007
\newsymbol\circlearrowright 1308
```

```
\newsymbol\circlearrowleft 1309
\undefine\rightleftharpoons
\newsymbol\rightleftharpoons 130A
\newsymbol\leftrightharpoons 130B
\newsymbol\boxminus 120C
\newsymbol\Vdash 130D
\newsymbol\Vvdash 130E
\newsymbol\vDash 130F
\newsymbol\twoheadrightarrow 1310
\newsymbol\twoheadleftarrow 1311
\newsymbol\leftleftarrows 1312
\newsymbol\rightrightarrows 1313
\newsymbol\upuparrows 1314
\newsymbol\downdownarrows 1315
\newsymbol\upharpoonright 1316
 \let\restriction\upharpoonright
\newsymbol\downharpoonright 1317
\newsymbol\upharpoonleft 1318
\newsymbol\downharpoonleft 1319
\newsymbol\rightarrowtail 131A
\newsymbol\leftarrowtail 131B
\newsymbol\leftrightarrows 131C
\newsymbol\rightleftarrows 131D
\newsymbol\Lsh 131E
\newsymbol\Rsh 131F
\newsymbol\rightsquigarrow 1320
\newsymbol\leftrightsquigarrow 1321
\newsymbol\looparrowleft 1322
\newsymbol\looparrowright 1323
```

```
\newsymbol\circeq 1324
\newsymbol\succsim 1325
\newsymbol\gtrsim 1326
\newsymbol\gtrapprox 1327
\newsymbol\multimap 1328
\newsymbol\therefore 1329
\newsymbol\because 132A
\newsymbol\doteqdot 132B
 \let\Doteq\doteqdot
\newsymbol\triangleq 132C
\newsymbol\precsim 132D
\newsymbol\lesssim 132E
\newsymbol\lessapprox 132F
\newsymbol\eqslantless 1330
\newsymbol\eqslantgtr 1331
\newsymbol\curlyeqprec 1332
\newsymbol\curlyeqsucc 1333
\newsymbol\preccurlyeq 1334
\newsymbol\leqq 1335
\newsymbol\leqslant 1336
\newsymbol\lessgtr 1337
\newsymbol\backprime 1038
\newsymbol\risingdotseq 133A
\newsymbol\fallingdotseq 133B
\newsymbol\succcurlyeq 133C
\newsymbol\geqq 133D
\newsymbol\geqslant 133E
\newsymbol\gtrless 133F
\newsymbol\sqsubset 1340
```

font-ini
font-ans
font-ibm
font-cmr
font-con
font-eul
font-ams
font-lbr
font-pos
font-ptm
font-pcr
font-phv

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
\newsymbol\sqsupset 1341
\newsymbol\vartriangleright 1342
\newsymbol\vartriangleleft 1343
\newsymbol\trianglerighteq 1344
\newsymbol\trianglelefteq 1345
\newsymbol\bigstar 1046
\newsymbol\between 1347
\newsymbol\blacktriangledown 1048
\newsymbol\blacktriangleright 1349
\newsymbol\blacktriangleleft 134A
\newsymbol\vartriangle 134D
\newsymbol\blacktriangle 104E
\newsymbol\triangledown 104F
\newsymbol\eqcirc 1350
\newsymbol\lesseqgtr 1351
\newsymbol\gtreqless 1352
\newsymbol\lesseqqgtr 1353
\newsymbol\gtreqqless 1354
\newsymbol\Rrightarrow 1356
\newsymbol\Lleftarrow 1357
\newsymbol\veebar 1259
\newsymbol\barwedge 125A
\newsymbol\doublebarwedge 125B
\undefine\angle
\newsymbol\angle 105C
\newsymbol\measuredangle 105D
\newsymbol\sphericalangle 105E
\newsymbol\varpropto 135F
\newsymbol\smallsmile 1360
```

```
\newsymbol\smallfrown 1361
\newsymbol\Subset 1362
\newsymbol\Supset 1363
\newsymbol\Cup 1264
 \let\doublecup\Cup
\newsymbol\Cap 1265
 \let\doublecap\Cap
\newsymbol\curlywedge 1266
\newsymbol\curlyvee 1267
\newsymbol\leftthreetimes 1268
\newsymbol\rightthreetimes 1269
\newsymbol\subseteqq 136A
\newsymbol\supseteqq 136B
\newsymbol\bumpeq 136C
\newsymbol\Bumpeq 136D
\newsymbol\lll 136E
 \let\llless\lll
\newsymbol\ggg 136F
 \let\gggtr\ggg
\newsymbol\circledS 1073
\newsymbol\pitchfork 1374
\newsymbol\dotplus 1275
\newsymbol\backsim 1376
\newsymbol\backsimeq 1377
\newsymbol\complement 107B
\newsymbol\intercal 127C
\newsymbol\circledcirc 127D
\newsymbol\circledast 127E
\newsymbol\circleddash 127F
```

```
\newsymbol\lvertneqq 2300
\newsymbol\gvertneqq 2301
\newsymbol\nleq 2302
\newsymbol\ngeq 2303
\newsymbol\nless 2304
\newsymbol\ngtr 2305
\newsymbol\nprec 2306
\newsymbol\nsucc 2307
\newsymbol\lneqq 2308
\newsymbol\gneqq 2309
\newsymbol\nleqslant 230A
\newsymbol\ngeqslant 230B
\newsymbol\lneq 230C
\newsymbol\gneq 230D
\newsymbol\npreceq 230E
\newsymbol\nsucceq 230F
\newsymbol\precnsim 2310
\newsymbol\succnsim 2311
\newsymbol\lnsim 2312
\newsymbol\gnsim 2313
\newsymbol\nleqq 2314
\newsymbol\ngeqq 2315
\newsymbol\precneqq 2316
\newsymbol\succneqq 2317
\newsymbol\precnapprox 2318
\newsymbol\succnapprox 2319
\newsymbol\lnapprox 231A
\newsymbol\gnapprox 231B
\newsymbol\nsim 231C
```

```
\newsymbol\ncong 231D
\newsymbol\diagup 231E
\newsymbol\diagdown 231F
\newsymbol\varsubsetneq 2320
\newsymbol\varsupsetneq 2321
\newsymbol\nsubseteqq 2322
\newsymbol\nsupseteqq 2323
\newsymbol\subsetneqq 2324
\newsymbol\supsetneqq 2325
\newsymbol\varsubsetneqq 2326
\newsymbol\varsupsetneqq 2327
\newsymbol\subsetneq 2328
\newsymbol\supsetneq 2329
\newsymbol\nsubseteq 232A
\newsymbol\nsupseteq 232B
\newsymbol\nparallel 232C
\newsymbol\nmid 232D
\newsymbol\nshortmid 232E
\newsymbol\nshortparallel 232F
\newsymbol\nvdash 2330
\newsymbol\nVdash 2331
\newsymbol\nvDash 2332
\newsymbol\nVDash 2333
\newsymbol\ntrianglerighteq 2334
\newsymbol\ntrianglelefteq 2335
\newsymbol\ntriangleleft 2336
\newsymbol\ntriangleright 2337
\newsymbol\nleftarrow 2338
\newsymbol\nrightarrow 2339
```

font-ini
font-ans
font-ibm
font-cmr
font-con
font-eul
font-ams
font-lbr
font-pos
font-ptm
font-pcr
font-phv

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
\newsymbol\nLeftarrow 233A
\newsymbol\nRightarrow 233B
\newsymbol\nLeftrightarrow 233C
\newsymbol\nleftrightarrow 233D
\newsymbol\divideontimes 223E
\newsymbol\varnothing 203F
\newsymbol\nexists 2040
\newsymbol\Finv 2060
\newsymbol\Game 2061
\newsymbol\mho 2066
\newsymbol\eth 2067
\newsymbol\eqsim 2368
\newsymbol\beth 2069
\newsymbol\gimel 206A
\newsymbol\daleth 206B
\newsymbol\lessdot 236C
\newsymbol\gtrdot 236D
\newsymbol\ltimes 226E
\newsymbol\rtimes 226F
\newsymbol\shortmid 2370
\newsymbol\shortparallel 2371
\newsymbol\smallsetminus 2272
\newsymbol\thicksim 2373
\newsymbol\thickapprox 2374
\newsymbol\approxeq 2375
\newsymbol\succapprox 2376
\newsymbol\precapprox 2377
\newsymbol\curvearrowleft 2378
\newsymbol\curvearrowright 2379
```

font-ini
font-ans
font-ibm
font-cmr
font-con
font-eul
font-ams
font-lbr
font-pos
font-ptm
font-pcr
font-phv

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
\newsymbol\digamma 207A
\newsymbol\varkappa 207B
\newsymbol\Bbbk 207C
\newsymbol\hslash 207D
\undefine\hbar
\newsymbol\hbar 207E
\newsymbol\backepsilon 237F
```

Restore the catcode value for @ that was previously saved.

23

```
\catcode`\@=\csname pre amssym.tex at\endcsname
```

6.8 Lucida Bright

font-ini
font-ans
font-ibm
font-cmr
font-con
font-eul
font-ams
font-lbr
font-pos
font-ptm
font-pcr
font-phv

The Lucida Bright fonts are both good looking and and complete. These fonts have prebuilt accented characters, which means that we use another encoding vector: Y&Y texnansi. These fonts are a good illustration that a 12 point corps is indeed never that size. The Lucida Bright fonts come in one design size.

1
```
\startcoding[texnansi]
```

2
```
\definecorps [14.4pt,12pt,11pt,10pt,9pt,8pt] [rm]
  [tf=lbr sa 1,
   bf=lbd sa 1,
   it=lbi sa 1,
   sl=lbsl sa 1,
   bi=lbdi sa 1,
   bs=lbdi sa 1,
  tfa=lbr sa 1.200,
  tfb=lbr sa 1.440,
  tfc=lbr sa 1.728,
  tfd=lbr sa 2.074,
   sc=lbr sa 0.833]
```

3
```
\definecorps [14.4pt,12pt,11pt,10pt,9pt,8pt] [ss]
  [tf=lsr sa 1,
   bf=lsd sa 1,
   it=lsi sa 1,
   sl=lsi sa 1,
   bi=lsdi sa 1,
   bs=lsdi sa 1,
  tfa=lsr sa 1.200,
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
  tfb=lsr sa 1.440,
  tfc=lsr sa 1.728,
  tfd=lsr sa 2.074,
   sc=lsr sa 0.833]
```

4
```
\definecorps [14.4pt,12pt,11pt,10pt,9pt,8pt] [tt]
  [tf=lstr sa 1,
   sl=lsto sa 1,
  tfa=lstr sa 1.200,
  tfb=lstr sa 1.440,
  tfc=lstr sa 1.728,
  tfd=lstr sa 2.074]
```

5
```
\definecorps [14.4pt,12pt,11pt,10pt,9pt,8pt] [hw]
  [tf=lbh sa 1]
```

6
```
\definecorps [14.4pt,12pt,11pt,10pt,9pt,8pt] [cg]
  [tf=lbc sa 1]
```

7
```
\definecorps [14.4pt,12pt,11pt,10pt,9pt,8pt] [mm]
  [ex=lbme sa 1,
   mi=lbmo sa 1,
   sy=lbms sa 1,
   ma=lbma sa 1]
```

8
```
\definecorps [7pt,6pt,5pt] [rm]
  [tf=lbr sa 1,
   bf=lbd sa 1,
   sl=lbi sa 1,
   it=lbi sa 1]
```

9
```
\definecorps [7pt,6pt,5pt] [ss]
  [tf=lsr sa 1,
   sl=lsd sa 1,
   it=lsi sa 1,
   bf=lsi sa 1]
```

10
```
\definecorps [7pt,6pt,5pt] [tt]
  [tf=lstr sa 1,
   sl=lsto sa 1]
```

11
```
\definecorps [7pt,6pt,5pt] [mm]
  [ex=lbme sa 1,
   mi=lbmo sa 1,
   sy=lbms sa 1,
   ma=lbma sa 1]
```

Defining the larger alternatives takes only a few commands, thanks to `sa`.

12
```
\definecorps [14.4pt,12pt,11pt,10pt,9pt,8pt] [rm]
  [bfa=lbd sa 1.200,
   bfb=lbd sa 1.440,
   bfc=lbd sa 1.728,
   bfd=lbd sa 2.074,
   sla=lbsl sa 1.200,
   slb=lbsl sa 1.440,
   slc=lbsl sa 1.728,
   sld=lbsl sa 2.074,
   bsa=lbdi sa 1.200,
   bsb=lbdi sa 1.440,
   bsc=lbdi sa 1.728,
   bsd=lbdi sa 2.074]
```

13
```
\definecorps [14.4pt,12pt,11pt,10pt,9pt,8pt] [ss]
  [bfa=lsr sa 1.200,
   bfb=lsr sa 1.440,
   bfc=lsr sa 1.728,
   bfd=lsr sa 2.074,
   sla=lsd sa 1.200,
   slb=lsd sa 1.440,
   slc=lsd sa 1.728,
   sld=lsd sa 2.074,
   bsa=lsi sa 1.200,
   bsb=lsi sa 1.440,
   bsc=lsi sa 1.728,
   bsd=lsi sa 2.074]
```

14
```
\definecorps [14.4pt,12pt,10pt] [tt]
  [sla=lsto sa 1.220,
   slb=lsto sa 1.440,
   slc=lsto sa 1.728,
   sld=lsto sa 2.074]
```

15
```
\stopcoding
```

Firt we implement some alternatives for AMS symbols. These can be overrules by loading the AMS font module afterwards.

16
```
\mathchardef\blacktriangleleft ="01F0
\mathchardef\blacktriangleright="01F1
\mathchardef\boxtimes          ="02EC
```

Here I copied the definition file that is part of the Y&Y distribution.

This part of the definition is adapted bij J. Hagen. There is already an extra family: `\mafam` (Math A). Also, the loading of fonts is done somewhere else.

```
\unprotect
```

```
\let\arfam    = \mafam
\let\thearfam = \hexmafam
```

This part is adapted to the CONTEXT font–naming method. Also, we use `\setskewchar`, which activates the not yet loaded font.

The next definitions are already taken care of.

```
% \setskewchar{12ptmmmi}='177
% \setskewchar{11ptmmmi}='177
% \setskewchar{10ptmmmi}='177
% \setskewchar{9ptmmmi}='177
% \setskewchar{8ptmmmi}='177
% \setskewchar{7ptmmmi}='177
% \setskewchar{6ptmmmi}='177
% \setskewchar{5ptmmmi}='177

% \setskewchar{12ptmmsy}='60
% \setskewchar{11ptmmsy}='60
% \setskewchar{10ptmmsy}='60
% \setskewchar{9ptmmsy}='60
% \setskewchar{8ptmmsy}='60
% \setskewchar{7ptmmsy}='60
% \setskewchar{6ptmmsy}='60
```

font-ini
font-ans
font-ibm
font-cmr
font-con
font-eul
font-ams
font-lbr
font-pos
font-ptm
font-pcr
font-phv

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
  % \setskewchar{5ptmmsy}='60
```

Adjusted for LucidaNewMath–Extension at 10pt and math axis at 3.13pt Note: delimiter increments are 5.5pt (as opposed to 6pt in CM).

19
```
\def\big    #1{{\hbox{$\left#1\vbox to8.20\p@{}\right.\n@space$}}}
\def\Big    #1{{\hbox{$\left#1\vbox to10.80\p@{}\right.\n@space$}}}
\def\bigg   #1{{\hbox{$\left#1\vbox to13.42\p@{}\right.\n@space$}}}
\def\Bigg   #1{{\hbox{$\left#1\vbox to16.03\p@{}\right.\n@space$}}}
\def\biggg  #1{{\hbox{$\left#1\vbox to17.72\p@{}\right.\n@space$}}}
\def\Biggg  #1{{\hbox{$\left#1\vbox to21.25\p@{}\right.\n@space$}}}
\def\n@space  {\nulldelimiterspace\z@ \m@th}
```

Define some extra large sizes. It's always done using extensible parts.

20
```
\def\bigggl{\mathopen\biggg}
\def\bigggr{\mathclose\biggg}
\def\Bigggl{\mathopen\Biggg}
\def\Bigggr{\mathclose\Biggg}
```

The following is needed if the roman text font is *not* just LBR.

Draw the small sizes of [and] from LBMO instead of LBR.

21
```
\mathcode`\[="4186 \delcode`\[="186302
\mathcode`\]="5187 \delcode`\]="187303
```

Draw the small sizes of (and) from LBMO instead of LBR.

22
```
\mathcode`\(="4184 \delcode`\(="184300
\mathcode`\)="5185 \delcode`\)="185301
```

The small sizes of { and } are already drawn from LBMS.

Draw small / from LBMO instead of LBR.

23
```
\mathcode`\/="013D \delcode`\/="13D30E
```

Draw = and + from LBMS instead of LBR.

24
```
\mathcode`\=="3283 \mathcode`\+="2282
```

Make open face brackets accessible, i.e. [[and]].

25
```
\def\ldbrack{\delimiter"4182382}
\def\rdbrack{\delimiter"5183383}
```

Provide access to surface integral signs (linked from text to display size).

26
```
\mathchardef\surfintop="1390
\def\surfint{\surfintop\nolimits}
```

Make medium size integrals available (*not* linked to display size).

27
```
\mathchardef\midintop="1392
\def\midint{\midintop\nolimits}
```

28
```
\mathchardef\midointop="1393
\def\midoint{\midointop\nolimits}
```

29
```
\mathchardef\midsurfintop="1394
\def\midsurfint{\midsurfintop\nolimits}
```

Extensible integral (use with \bigg, \Bigg, \biggg, \Biggg etc).

30
```
\def\largeint{\delimiter"135A395}
```

Various types of small integrals.

31
```
\mathchardef\dblint  ="0188
\mathchardef\trplint ="0189
\mathchardef\contint ="018A
\mathchardef\surfint ="018B
\mathchardef\volint  ="018C
\mathchardef\clwint  ="018D
\mathchardef\cclwcint="018E
\mathchardef\clwcint ="018F
```

To close up gaps in special math characters constructed from pieces.

32
```
\def\joinrel{\mathrel{\mkern-4mu}}
```

Some characters that need construction in CM exist complete in LBMO or LBMS.

33
```
\mathchardef\bowtie="31F6
\mathchardef\models="32EE
\mathchardef\doteq ="32C9
\mathchardef\cong  ="329B
\mathchardef\angle ="028B
```

Some more characters.

34
```
\mathchardef\hbar              ="019D
\mathchardef\neq               ="3\thearfam 94
\mathchardef\rightleftharpoons="3\thearfam 7A
\mathchardef\leftrightharpoons="3\thearfam 79
\mathchardef\hookleftarrow     ="3\thearfam 3C
```

```
\mathchardef\hookrightarrow   ="3\thearfam 3E
\mathchardef\mapsto           ="3\thearfam 2C
```

The (is not large enough for strut in LBMO.

35
```
\def\mathstrut{\vphantom{f}}
```

In $n^{\rm th}$ root, don't want the n to come too close to the radical.

36
```
\def\r@@t#1#2%
  {\setbox\z@\hbox{$\m@th#1\sqrt{#2}$}
   \dimen@\ht\z@ \advance\dimen@-\dp\z@
   \mkern5mu\raise.6\dimen@\copy\rootbox \mkern-7.5mu \box\z@}
```

Draw upper case upright greek from LucidaNewMath–Extension.

37
```
\mathchardef\Gamma  ="03D0
\mathchardef\Delta  ="03D1
\mathchardef\Theta  ="03D2
\mathchardef\Lambda ="03D3
\mathchardef\Xi     ="03D4
\mathchardef\Pi     ="03D5
\mathchardef\Sigma  ="03D6
\mathchardef\Upsilon="03D7
\mathchardef\Phi    ="03D8
\mathchardef\Psi    ="03D9
\mathchardef\Omega  ="03DA
```

Draw upper case italic greek from LucidaNewMath–Italic.

38
```
\mathchardef\varGamma  ="0100
\mathchardef\varDelta  ="0101
```

```
\mathchardef\varTheta  ="0102
\mathchardef\varLambda ="0103
\mathchardef\varXi     ="0104
\mathchardef\varPi     ="0105
\mathchardef\varSigma  ="0106
\mathchardef\varUpsilon="0107
\mathchardef\varPhi    ="0108
\mathchardef\varPsi    ="0109
\mathchardef\varOmega  ="010A
```

`\matrix` is changed because LBMO is not at 10pt.

39
```
\def\matrix#1%
  {\null\,\vcenter{\normalbaselines\m@th
   \ialign{\hfil$##$\hfil&&\quad\hfil$##$\hfil\crcr
   \mathstrut\crcr\noalign{\kern-0.9\baselineskip}
   #1\crcr\mathstrut\crcr\noalign{\kern-0.9\baselineskip}}}\,}
```

40
```
\protect
```

6.9 Standard Postscript

This file load the Adobe Times Roman, Helvetica and Courier.

```
\input font-ptm
\input font-phv
\input font-pcr
```

6.10 Adobe Times Roman

This module defines the Standard Adobe Times Roman. We use the Y&Y texnansi encoding vector.

1
```
\startcoding [texnansi]
```

2
```
\definecorps [14.4pt,12pt,11pt,10pt,9pt,8pt] [rm]
  [tf=tir sa 1,
   bf=tib sa 1,
   it=tii sa 1,
   sl=tii sa 1,         % tio
   bi=tibi sa 1,
   bs=tib sa 1,         % tibio
  tfa=tir sa 1.200,
  tfb=tir sa 1.440,
  tfc=tir sa 1.728,
  tfd=tir sa 2.074,
   sc=tir sa 0.833]
```

3
```
\definecorps [7pt,6pt,5pt] [rm]
  [tf=tir sa 1,
   bf=tib sa 1,
   it=tii sa 1,
   sl=tii sa 1,         % tio
   bi=tibi sa 1,
   bs=tib sa 1]         % tibio
```

4
```
\definecorps [14.4pt,12pt,11pt,10pt,9pt,8pt] [rm]
  [bfa=tib sa 1.200,
   bfb=tib sa 1.440,
```

font-ini
font-ans
font-ibm
font-cmr
font-con
font-eul
font-ams
font-lbr
font-pos
font-ptm
font-pcr
font-phv

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
    bfc=tib sa 1.728,
    bfd=tib sa 2.074,
    sla=tio sa 1.200,
    slb=tio sa 1.440,
    slc=tio sa 1.728,
    sld=tio sa 2.074,
    bsa=tib sa 1.200,     % tibio
    bsb=tib sa 1.440,     % tibio
    bsc=tib sa 1.728,     % tibio
    bsd=tib sa 2.074]     % tibio

\stopcoding
```

6.11 Adobe Courier

This module defines the Standard Adobe Courier. We use the Y&Y texnansi encoding vector.

```
\startcoding[texnansi]

\definecorps [14.4pt,12pt,11pt,10pt,9pt,8pt] [tt]
  [tf=com sa 1,
   sl=coo sa 1,
   tfa=com sa 1.200,
   tfb=com sa 1.440,
   tfc=com sa 1.728,
   tfd=com sa 2.074]

\definecorps [7pt,6pt,5pt] [tt]
  [tf=com sa 1,
   sl=coo sa 1]

\definecorps [14.4pt,12pt,11pt,10pt,9pt,8pt,7pt,6pt,5pt] [tt]
  [tf=com sa 1,
   sl=coo sa 1]

\definecorps [14.4pt,12pt,10pt] [tt]
  [sla=coo sa 1.200,
   slb=coo sa 1.440,
   slc=coo sa 1.728,
   sld=coo sa 2.074]

\stopcoding
```

6.12 Adobe Helvetica

This module defines the Standard Adobe Helvetica. We use the Y&Y texnansi encoding vector.

```
\startcoding [texnansi]
```

```
\definecorps [14.4pt,12pt,11pt,10pt,9pt,8pt] [ss]
  [tf=hv sa 1,
   bf=hvb sa 1,
   it=hvo sa 1,
   sl=hvo sa 1,
   bs=hvbo sa 1,
   bi=hvbo sa 1,
  tfa=hv sa 1.200,
  tfb=hv sa 1.440,
  tfc=hv sa 1.728,
  tfd=hv sa 2.074,
   sc=hv sa 0.833]
```

```
\definecorps [7pt,6pt,5pt] [ss]
  [tf=hv sa 1,
   bf=hvb sa 1,
   it=hvo sa 1,
   sl=hvo sa 1,
   bs=hvbo sa 1,
   bi=hvbo sa 1]
```

```
\definecorps [14.4pt,12pt,11pt,10pt,9pt,8pt] [tt]
  [bfa=hvb sa 1.200,
   bfb=hvb sa 1.440,
   bfc=hvb sa 1.728,
   bfd=hvb sa 2.074,
   sla=hvo sa 1.200,
   slb=hvo sa 1.440,
   slc=hvo sa 1.728,
   sld=hvo sa 2.074,
   bsa=hvbo sa 1.200,
   bsb=hvbo sa 1.440,
   bsc=hvbo sa 1.728,
   bsd=hvbo sa 2.074]

\stopcoding
```

7 Color Support

7.1 Initialization

1
```
\writestatus{loading}{Context Color Macros}
```

2
```
\unprotect
```

Color support is not present in TEX. Colorful output can however be accomplished by using specials. This also means that this support depends on the DVI driver used. At the moment this module was written, still no decent standard on color specials has been agreed upon. We therefore decided to implement a mechanism that is as independant as possible of drivers.

Color support shares with fonts that is must be implemented in a way that permits processing of individual DVI pages. Furthermore it should honour grouping. The first condition forces us to use a scheme that keeps track of colors at page boundaries. This can be done by means of TEX's marking mechanism (`\mark`).

When building pages, TEX periodically looks at the accumulated typeset contents and breaks the page when suitable. At that moment, control is transfered to the output routine. This routine takes care of building the pagebody and for instance adds headers and footers. The page can be broken in the middle of some colored text, but headers and footers are often in black upon white or background. If colors are applied there, they definitely are used local, which means that they don't cross page borders.

Boxes are handled as a whole, which means that when we apply colors inside a box, those colors don't cross page boundaries, unless of course boxes are split or unboxed. Especially in interactive texts, colors are often used in such a local way: in boxes (buttons and navigational tools) or in the pagebody (backgrounds).

So we can distinguish local colors, that don't cross pages from global colors, of which we can end many pages later. The color macros will treat both types in a different way, thus gaining some speed.

This module also deals with gray scales. Because similar colors can end up in the same gray scale when printed in black and white, we also implement a palet system that deals with these matters. Because of fundamental differences between color and gray scale printing, in CONTEXT we also differ between these. For historic reasons —we first implemented gray scales using patterns of tiny periods— and therefore called them *rasters*. So don't be surprised if this term shows up.

colo-ini
colo-rgb
colo-xwi
colo-mwi
colo-pra

3
```
\startmessages  dutch  library: colors
  title: kleur
      1: systeem -- is globaal actief
      2: systeem -- is lokaal actief
      3: -- is niet gedefinieerd
      4: systeem -- wordt geladen
      5: onbekend systeem --
      6: palet -- is beschikbaar
      7: palet -- is niet beschikbaar
      8: (verkeerde) specificatie -- bij -- wordt zwart
      9: -- kleurruimte wordt niet ondersteund
     10: -- kleurruimte wordt ondersteund
     11: kleur wordt vertaald in grijs
\stopmessages
```

4
```
\startmessages  english  library: colors
  title: color
      1: system -- is global activated
      2: system -- is local activated
      3: -- is not defined
      4: system -- is loaded
      5: unknown system --
      6: palette -- is available
      7: palette -- is not available
      8: (invalid) specification -- at color -- becomes black
      9: -- color space is not supported
     10: -- color space is supported
     11: color is converted to gray
\stopmessages
```

5

```
\startmessages  german  library: colors
  title: farbe
      1: system -- ist global aktiviert
      2: system -- ist lokal aktiviert
      3: -- ist undefiniert
      4: system -- ist geladen
      5: unbekanntes System --
      6: palette -- ist verfuegbar
      7: palette -- ist nicht verfuegbar
      8: (ungueltige) Spezifikation -- bei Farbe -- wird schwarz
      9: -- Farbraum wird nicht unterstuetzt
     10: -- Farbraum wird unterstuetzt
     11: Farbe wird in Grau umgewandelt
\stopmessages
```

We will enable users to specify colors in RGB and CMYK color spaces or gray scales using

\definecolor

```
\definieerkleur[...][..,..=..,..]
```

...	*naam*
r	*tekst*
g	*tekst*
b	*tekst*
c	*tekst*
m	*tekst*
y	*tekst*
k	*tekst*

For example:

```
\definecolor [SomeKindOfRed] [r=.8,g=.05,b=.05]
```

Such color specifications are saved in a macro in the following way:

```
\setvalue{\??cr name}{R:r:g:b}
\setvalue{\??cr name}{C:c:m:y:k}
\setvalue{\??cr name}{K:k}
```

Gray scales are specified with the `k` parameter, the same as used in CMYK specifications.

6
```
\def\colorlist{}
```

7
```
\def\@@cl@@z{0}
\def\@@cl@@o{1}
```

8
```
\def\definecolor%
  {\dodoubleargument\dodefinecolor}
```

9
```
\def\dodefinecolor[#1][#2]%
  {\redoglobal\addtocommalist{#1}\colorlist
   \doifassignmentelse{#2}
     {\let\@@cl@@r=\@@cl@@z\let\@@cl@@g=\@@cl@@z\let\@@cl@@b=\@@cl@@z
      \let\@@cl@@c=\@@cl@@z\let\@@cl@@m=\@@cl@@z\let\@@cl@@y=\@@cl@@z
      \let\@@cl@@k=\@@cl@@z
      \getparameters[\??cl @@][#2]%
      \doifelse{\@@cl@@r\@@cl@@g\@@cl@@b}{\@@cl@@z\@@cl@@z\@@cl@@z}
        {\doifelse{\@@cl@@c\@@cl@@m\@@cl@@y}{\@@cl@@z\@@cl@@z\@@cl@@z}
           {\doifelse{\@@cl@@k}{\@@cl@@z}
              {\showmessage{\m!colors}{8}{{[#2]},#1}%
               \redoglobal\setevalue{\??cr#1}{K:\@@cl@@z}}
              {\redoglobal\setevalue{\??cr#1}{K:\@@cl@@k}}}
           {\redoglobal\setevalue{\??cr#1}{C:\@@cl@@c:\@@cl@@m:\@@cl@@y:\@@cl@@k}}}
        {\redoglobal\setevalue{\??cr#1}{R:\@@cl@@r:\@@cl@@g:\@@cl@@b}}}
     {\doifdefinedelse{\??cr#2}
        {\redoglobal\setevalue{\??cr#1}{\getvalue{\??cr#2}}}
        {\showmessage{\m!colors}{3}{#1}}}%
   \dodoglobal\setvalue{#1}{\color[#1]}}
```

The names of colors are stored in a comma separated list only for the purpose of showing them with `\showcolor`.

Colors can be defined global by using `\doglobal`, like in

```
   \doglobal\definecolor [SomeKindOfRed] [r=.8,g=.05,b=.05]
```

This color shows up as some kind of red.

==macrossetupcolor

colo-ini
colo-rgb
colo-xwi
colo-mwi
colo-pra

contents register context syst mult supp lang font colo spec core cont m s exit go back

colo-ini
colo-rgb
colo-xwi
colo-mwi
colo-pra

Color definitions can be grouped in files with the name:

```
\f!colorprefix-identifier.tex
```

where `\f!colorprefix` is `colo-`. Loading such a file is done by

```
\stelkleurin[...]

...     naam
```

Some default colors are specified in `colo-rgb.tex`, which is loaded into the format by:

```
\setupcolor[rgb]
```

10
```
\def\colorstyle{}
```

11
```
\def\setupcolor%
  {\dosingleargument\dosetupcolor}
```

12
```
\def\dosetupcolor[#1]%
  {\doifnot{#1}{\colorstyle}
     {\def\colorstyle{#1}%
      \def\dodosetupcolor##1%
        {\readsysfile{\f!colorprefix##1}%
           {\showmessage{\m!colors}{4}{\colorstyle}}
           {\showmessage{\m!colors}{5}{\colorstyle}}}%
      \processcommalist[#1]\dodosetupcolor}}
```

When typesetting for paper, we prefer using the CMYK color space, but for on–screen viewing we prefer RGB (the previous implementation supported only this scheme). Independant of such specifications, we support some automatic conversions:

contents register context syst mult supp lang font colo spec core cont m s exit go back

- convert all colors to RGB
- convert all colors to CMYK
- convert all colors to gray scales

We also support optimization of colors to gray scales.

- reduce gray colors to gray scales

These options are communicated by means of:

13
```
\newif\ifRGBsupported
\newif\ifCMYKsupported
\newif\ifconverttoGRAY
\newif\ifpreferGRAY
\newif\ifGRAYprefered
\newif\ifreduceCMYK
```

The last boolean controls reduction of CMYK to CMY colors. When set to true, the black component is added to the other ones.

Color modes are entered using the next set of commands. The `\stop` alternatives are implemented in a way that permits non–grouped use.

14
```
\def\dostartcolormodeR#1:#2:#3\od%
  {\bgroup
   \def\@@cl@@r{#1}\def\@@cl@@g{#2}\def\@@cl@@b{#3}%
   \ifpreferGRAY\ifx\@@cl@@r\@@cr@@g\ifx\@@cl@@r\@@cl@@b
     \GRAYpreferedtrue
   \fi\fi\fi
   \ifGRAYprefered
     \dostartgraycolormode\@@cl@@r
```

```
    \else\ifRGBsupported
      \dostartrgbcolormode\@@cl@@r\@@cl@@g\@@cl@@b
    \else\ifCMYKsupported
      \convertRGBtoCMYK\@@cl@@r\@@cl@@g\@@cl@@b
      \dostartcmykcolormode\@@cl@@c\@@cl@@m\@@cl@@y\@@cl@@k
    \else
      \convertRGBtoGRAY\@@cl@@r\@@cl@@g\@@cl@@b
      \dostartgraycolormode\@@cl@@k
    \fi\fi\fi
    \egroup}

\def\dostartcolormodeC#1:#2:#3:#4\od%
  {\bgroup
    \def\@@cl@@c{#1}\def\@@cl@@m{#2}\def\@@cl@@y{#3}\def\@@cl@@k{#4}%
    \ifpreferGRAY\ifx\@@cl@@k\@@cl@@z\ifx\@@cl@@c\@@cr@@m\ifx\@@cl@@c\@@cl@@y
      \GRAYpreferedtrue
    \fi\fi\fi\fi
    \ifGRAYprefered
      \dostartgraycolormode\@@cl@@c
    \else\ifCMYKsupported
      \ifreduceCMYK
        \convertCMYKtoCMY\@@cl@@c\@@cl@@m\@@cl@@y\@@cl@@k
        \dostartcmykcolormode\@@cl@@c\@@cl@@m\@@cl@@y\@@cl@@k
      \else
        \dostartcmykcolormode\@@cl@@c\@@cl@@m\@@cl@@y\@@cl@@k
      \fi
    \else\ifRGBsupported
      \convertCMYKtoRGB\@@cl@@c\@@cl@@m\@@cl@@y\@@cl@@k
      \dostartrgbcolormode\@@cl@@r\@@cl@@g\@@cl@@b
    \else
```

```
      \convertCMYKtoGRAY\@@cl@@c\@@cl@@m\@@cl@@y\@@cl@@k
      \dostartgraycolormode\@@cl@@k
    \fi\fi\fi
    \egroup}
```

16
```
\def\dostartcolormodeK#1\od%
  {\dostartgraycolormode{#1}}
```

Prefering gray is not the same as converting to gray. Conversion treats each color components in a different way, while prefering is just a reduction and thus a space–saving option.

\startcolormode
\stopcolormode

We use `\stopcolormode` to reset the color in whatever color space and do so by calling the corresponding special. Both commands can be used for fast color switching, like in colored verbatim,

17
```
\def\dostartcolormode#1:%
  {\getvalue{dostartcolormode#1}}
```

18
```
\def\startcolormode#1%
  {\doifcolorelse{#1}
     {\getcurrentcolorspecs{#1}%
      \expandafter\dostartcolormode\currentcolorspecs\od}
     {\dostopcolormode}}
```

19
```
\def\stopcolormode%
  {\dostopcolormode}
```

We use some reserved names for local color components. Consistent use of these scratch variables saves us unneccessary hash entries.

```
    \@@cl@@r \@@cl@@g \@@cl@@b
    \@@cl@@c \@@cl@@m \@@cl@@y \@@cl@@k
    \@@cl@@k
```

colo-ini
colo-rgb
colo-xwi
colo-mwi
colo-pra

We implement several conversion routines.

```
   \convertRGBtoCMYK  {r} {g} {b}
   \convertRGBtoGRAY  {r} {g} {b}
   \convertCMYKtoRGB  {c} {m} {y} {k}
   \convertCMYKtoGRAY {c} {m} {y} {k}
   \convertCMYKtoCMY  {c} {m} {y} {k}
```

The relation between GRAY, RGB and CMYK is:

$$G = .30r + .59g + .11b = 1.0 - min(1.0,\ .30c + .59m + .11y + k)$$

When converting from CMYK to RGB we use the formula:

$$\begin{aligned} r &= 1.0 - min(1.0,\ c + k) \\ g &= 1.0 - min(1.0,\ m + k) \\ b &= 1.0 - min(1.0,\ y + k) \end{aligned}$$

In the conversion routine the color components are calculated in three digits precision.

20
```
\def\realcolorvalue#1%
  {\ifnum     #1<10   0.00\the#1%
   \else\ifnum#1<100  0.0\the#1%
   \else\ifnum#1<1000 0.\the#1%
   \else              1\fi\fi\fi}
```

21
```
\def\doconvertCMYKtoRGB#1\k#2\to#3%
  {\scratchdimen=#1\s!pt
   \advance\scratchdimen by #2\s!pt\relax
   \ifdim\scratchdimen>1\s!pt
     \scratchdimen=-1\s!pt
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

colo-ini
colo-rgb
colo-xwi
colo-mwi
colo-pra

```
   \else
     \scratchdimen=-\scratchdimen
   \fi
   \advance\scratchdimen by 1\s!pt
   \multiply\scratchdimen by 1000
   \scratchcounter=\scratchdimen
   \advance\scratchcounter by \!!medcard
   \divide\scratchcounter by \!!maxcard
   \edef#3{\realcolorvalue\scratchcounter}}
```

22
```
\def\convertCMYKtoRGB#1#2#3#4%
  {\doconvertCMYKtoRGB#1\k#4\to\@@cl@@r
   \doconvertCMYKtoRGB#2\k#4\to\@@cl@@g
   \doconvertCMYKtoRGB#3\k#4\to\@@cl@@b}
```

23
```
\def\doconvertRGBtoCMYK#1\to#2%
  {\scratchdimen=#1\s!pt
   \multiply\scratchdimen by 1000
   \scratchcounter=\scratchdimen
   \advance\scratchcounter by \!!medcard
   \divide\scratchcounter by \!!maxcard
   \scratchcounter=-\scratchcounter
   \advance\scratchcounter by 1000
   \edef#2{\realcolorvalue\scratchcounter}}
```

24
```
\def\convertRGBtoCMYKvalue#1#2#3%
  {\doconvertRGBtoCMYK#1\to\@@cl@@c
   \doconvertRGBtoCMYK#2\to\@@cl@@m
   \doconvertRGBtoCMYK#3\to\@@cl@@y
   \let\@@cl@@k=\@@cl@@z}
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

25
```
\def\convertRGBtoGRAY#1#2#3%
  {\scratchdimen=#1\s!pt
   \scratchdimen=300\scratchdimen
   \scratchcounter=\scratchdimen
   \scratchdimen=#2\s!pt
   \scratchdimen=590\scratchdimen
   \advance\scratchcounter by \scratchdimen
   \scratchdimen=#3\s!pt
   \scratchdimen=110\scratchdimen
   \advance\scratchcounter by \scratchdimen
   \advance\scratchcounter by \!!medcard
   \divide\scratchcounter by \!!maxcard
   \edef\@@cl@@k{\realcolorvalue\scratchcounter}}
```

26
```
\def\convertCMYKtoGRAY#1#2#3#4%
  {\convertCMYKtoRGB{#1}{#2}{#3}{#4}%
   \convertRGBtoGRAY\@@cl@@r\@@cl@@g\@@cl@@b}
```

27
```
\def\doconvertCMYKtoCMY#1\k#2\to#3%
  {\scratchdimen=#1\s!pt
   \advance\scratchdimen by #2\s!pt\relax
   \ifdim\scratchdimen>1\s!pt
     \scratchdimen=1\s!pt
   \else
     \scratchdimen=\scratchdimen
   \fi
   \multiply\scratchdimen by 1000
   \scratchcounter=\scratchdimen
   \advance\scratchcounter by \!!medcard
   \divide\scratchcounter by \!!maxcard
```

colo-ini
colo-rgb
colo-xwi
colo-mwi
colo-pra

```
   \edef#3{\realcolorvalue\scratchcounter}}
```

28
```
\def\convertCMYKtoCMY#1#2#3#4%
  {\doconvertCMYKtoCMY#1\k#4\to\@@cl@@c
   \doconvertCMYKtoCMY#2\k#4\to\@@cl@@m
   \doconvertCMYKtoCMY#3\k#4\to\@@cl@@y
   \let\@@cl@@k=\@@cl@@z}
```

We already mentioned that colors interfere with building the pagebody. This means that when the page is composed, the colors temporary have to be reset. After the page is shipped out, we have to revive the current color.

We use `\marks` to keep track of colors across page boundaries. Unfortunately standard TEX supports only one mark, and using this one for color support only would be a waste. We therefore use an adapted version of J. Fox's multiple mark mechanism as (re)implemented in `supp-mrk`.

29
```
\doifdefinedelse{newmark}
  {\newmark\colormark}
  {\def\colormark#1{}}
```

Using this mark mechanism with lots of colors has one major drawback: TEX's memory tends to overflow when very colorful text is stored in a global box. Even worse is that the processing time grows considerably. We therefore support local as well as global color switching.

Of the next macros, `\popcolor` is to be used after the actual `\shipout` and `\startcolorpage` and `\stopcolorpage` are called when entering and leaving the `\pagebody` builder. In case of emergencies `\pushcolor` can be used to undo the current color, for instance when insertions are appended to the page.

Before we present the color macros, we first define the setup command. This command takes care of setting up the booleans that control local and global behavior and conversion to other color spaces.

contents register context syst mult supp lang font colo spec core cont m s exit go back

30
```
\newif\ifincolor
\newif\iflocalcolor
```

31
```
\def\setupcolors%
  {\dosingleargument\dosetupcolors}
```

32
```
\def\dosetupcolors[#1]%
 {\getparameters[\??cl][#1]%
  \doifelse{\@@clconversie}{\v!ja}
    {\preferGRAYtrue}
    {\preferGRAYfalse}%
  \doifelse{\@@clrgb}{\v!nee}
    {\showmessage{\m!colors}{9}{\v!rgb}\RGBsupportedfalse}
    {\showmessage{\m!colors}{10}{\v!rgb}\RGBsupportedtrue}%
  \doifelse{\@@clcmyk}{\v!nee}
    {\showmessage{\m!colors}{9}{\v!cmyk}\CMYKsupportedfalse}
    {\showmessage{\m!colors}{10}{\v!cmyk}\CMYKsupportedtrue}%
  \ifRGBsupported
    \converttoGRAYfalse
  \else\ifCMYKsupported
    \converttoGRAYfalse
  \else
    \converttoGRAYtrue
    \showmessage{\m!colors}{11}{}%
  \fi\fi
  \processaction
    [\@@clstatus]
    [\v!globaal=>\incolortrue\localcolorfalse
                 \showmessage{\m!colors}{1}{\colorstyle},
      \v!lokaal=>\incolortrue\localcolortrue
```

colo-ini
colo-rgb
colo-xwi
colo-mwi
colo-pra

```
            \showmessage{\m!colors}{2}{\colorstyle},
  \v!start=>\let\@@clstatus=\v!globaal
            \incolortrue\localcolorfalse
            \showmessage{\m!colors}{1}{\colorstyle},
   \v!stop=>\incolorfalse\localcolorfalse]}
```

\doifcolorelse

Switching to a color is done by means of the following command. Later on we will explain the use of palets. We define ourselves a color conditional first.

33
```
\def\currentpalet{}
```

34
```
\def\doifcolorelse#1%
  {\doifdefinedelse{\??cr\currentpalet#1}}
```

35
```
\def\getcurrentcolorspecs#1%
  {\edef\currentcolorspecs{\getvalue{\??cr\currentpalet#1}}}
```

\localstartcolor
\localstopcolor

Simple color support, that is without nesting, is provided by:

36
```
\def\localstartcolor[#1]%
  {\ifincolor
     \localcolortrue
     \doglobalstartcolor[#1]%
   \fi}
```

37
```
\def\localstopcolor%
 {\ifincolor
    \doglobalstopcolor
  \fi}
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

\startcolor
\stopcolor

The more save method, the one that saves the current color state and returns to this state afterward, is activated by:

```
\startkleur[...] ... \stopkleur

...      naam
...      tekst
```

38
```
\def\startcolor[#1]%
  {\ifincolor
     \doglobalstartcolor[#1]%
   \fi}
```

39
```
\def\stopcolor%
  {\ifincolor
     \doglobalstopcolor
   \fi}
```

This macros call the global color switching ones. Starting a global, i.e. a possible page boundary crossing, color mode also sets a \mark in TEX's internal list.

40
```
\newcount\colorlevel
```

41
```
\setvalue{\??cl0C}{}  % saved color
\setvalue{\??cl0S}{}  % stop command
```

42
```
\def\dodoglobalstartcolor[#1]%
  {\xdef\currentcolor{\getvalue{\??cl\the\colorlevel C}}%
   \global\advance\colorlevel by 1
   \setxvalue{\??cl\the\colorlevel C}{#1}%
```

```
    \debuggerinfo{\m!colors}
      {start #1 at level \the\colorlevel}%
    \doifelsenothing{#1}
      {\setxvalue{\??cl\the\colorlevel C}{\currentcolor}%
       \setgvalue{\??cl\the\colorlevel S}{\donoglobalstopcolor}}
      {\doifelse{#1}{\currentcolor}
         {\setgvalue{\??cl\the\colorlevel S}{\donoglobalstopcolor}}
         {\doifcolorelse{#1}
            {\docolormark{#1}%
             \setgvalue{\??cl\the\colorlevel S}{\dodoglobalstopcolor}%
             \startcolormode{#1}}
         {\setgvalue{\??cl\the\colorlevel S}{\donoglobalstopcolor}%
          \showmessage{\m!colors}{3}{#1}}}}}
```

43
```
\def\doglobalstartcolor[#1]%
  {\ifnum\colorlevel=0
     \doifelsenothing{#1}
       {\setgvalue{\??cl\the\colorlevel S}{}}
       {\dodoglobalstartcolor[#1]}%
   \else
     \dodoglobalstartcolor[#1]%
   \fi
   \ignorespaces}
```

44
```
\def\donoglobalstopcolor%
  {\ifnum\colorlevel>0
     \xdef\currentcolor{\getvalue{\??cl\the\colorlevel C}}%
     \debuggerinfo{\m!colors}
       {stop \currentcolor\normalspace at level \the\colorlevel}%
     \global\advance\colorlevel by -1
```

colo-ini
colo-rgb
colo-xwi
colo-mwi
colo-pra

contents register context syst mult supp lang font colo spec core cont m s exit go back

colo-ini
colo-rgb
colo-xwi
colo-mwi
colo-pra

```
   \fi}
```

45
```
\def\dodoglobalstopcolor%
  {\ifnum\colorlevel>0
     \donoglobalstopcolor
     \xdef\previouscolor{\getvalue{\??cl\the\colorlevel C}}%
     \ifnum\colorlevel=0
       \docolormark{}%
       \stopcolormode
     \else % let's do a bit redundant testing here
       \docolormark{\previouscolor}%
       \doifelsenothing{\previouscolor}
         {\dostopcolormode}
         {\doifcolorelse{\previouscolor}
            {\doifnot{\currentcolor}{\previouscolor}
               {\startcolormode{\previouscolor}}}
            {\dostopcolormode}}%
     \fi
   \fi}
```

46
```
\def\doglobalstopcolor%
  {\getvalue{\??cl\the\colorlevel S}}
```

We don't use grouping and save each stop alternative. This permits be especially useful in for instance local color support in verbatim. Using `\bgroup`–`\egroup` pairs could interfere with calling commands

This color mechanism takes care of nested colors, like in:

```
   \kleur[groen]{groen \kleur[groen]{groen \kleur[rood]{rood}} groen}
   \kleur[groen]{groen \kleur[]{groen \kleur[rood]{rood}} groen}
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
\kleur[groen]{groen \kleur[rood]{rood \kleur[rood]{rood}} groen}
\kleur[groen]{groen \kleur[groen]{groen \kleur[]{groen}} groen}
\kleur[groen]{groen \kleur[rood]{rood} groen}
\kleur[groen]{groen \kleur[]{groen} groen}
\kleur[]{zwart \kleur[rood]{rood} zwart}
\kleur[]{zwart}
```

or

groen groen rood groen
groen groen rood groen
groen rood rood groen
groen groen groen groen
groen rood groen
groen groen groen
zwart rood zwart
zwart

Crossing page boundaries is of course also handled. Undefined or empty color specifications are treated as efficient as possible.

[groen] We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, avarage, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and seperate the sheep from the goats. [groen]

[groen] Thus, I came to the conclusion that the designer of a new system must not only be the implementer and first large–scale user; the designer should also write the first user manual.

The seperation of any of these four components would have hurt TEX significantly. If I had not participated fully in all these activities, literally hundreds of improvements would never have been made, because I would never have thought of them or percieved why they were important.

But a system cannot be succesful if it is too strongly influenced by a single person. Once the initial design is complete and fairly robust, the real test begins as people with many different viewpoints undertake their own experiments. [groen]

[rood] We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, avarage, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and seperate the sheep from the goats. [rood]

[geel] Thus, I came to the conclusion that the designer of a new system must not only be the implementer and first large–scale user; the designer should also write the first user manual.

The seperation of any of these four components would have hurt TEX significantly. If I had not participated fully in all these activities, literally hundreds of improvements would never have been made, because I would never have thought of them or percieved why they were important.

But a system cannot be succesful if it is too strongly influenced by a single person. Once the initial design is complete and fairly robust, the real test begins as people with many different viewpoints undertake their own experiments. [geel]

[rood] We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize,

condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, avarage, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and seperate the sheep from the goats. [rood]

[groen] Thus, I came to the conclusion that the designer of a new system must not only be the implementer and first large–scale user; the designer should also write the first user manual.

The seperation of any of these four components would have hurt TEX significantly. If I had not participated fully in all these activities, literally hundreds of improvements would never have been made, because I would never have thought of them or percieved why they were important.

But a system cannot be succesful if it is too strongly influenced by a single person. Once the initial design is complete and fairly robust, the real test begins as people with many different viewpoints undertake their own experiments. [groen]

[groen] We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, avarage, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and seperate the sheep from the goats. [groen]

These quotes are typeset by saying:

```
\startkleur[groen]
  [groen] \input tufte [groen] \par
  \startkleur[]
```

colo-ini
colo-rgb
colo-xwi
colo-mwi
colo-pra

contents register context syst mult supp lang font colo spec core cont m s exit go back

colo-ini
colo-rgb
colo-xwi
colo-mwi
colo-pra

```
        [groen] \input knuth [groen] \par
        \startkleur[rood]
          [rood] \input tufte [rood] \par
          \startkleur[geel]
            [geel] \input knuth [geel] \par
          \stopkleur
          [rood] \input tufte [rood] \par
        \stopkleur
        [groen] \input knuth [groen] \par
      \stopkleur
      [groen] \input tufte [groen] \par
    \stopkleur
```

Out of efficiency we only use marks when needed. The next macro tries to find out if indeed a mark should be set. This macro uses the boolean `\ifinpagebody`, which can be defined and set in the module that handles the pagebody.

47
```
\ifx\ifinpagebody\undefined \newif\ifinpagebodytrue \fi
```

48
```
\def\docolormark#1%
  {\ifinpagebody \else
     \iflocalcolor \else
       \ifinner
         \ifhmode \else
           \dodocolormark{#1}%
         \fi
       \else
         \dodocolormark{#1}%
       \fi
     \fi
```

```
    \fi}
```

49
```
\let\lastcolormark=\empty
```

50
```
\def\dodocolormark#1%
  {\doifnot{#1}{\lastcolormark}
     {\colormark{#1}%
      \xdef\lastcolormark{#1}}}
```

\pushcolor
\popcolor

Pushing the current state in the output routine simply comes to resetting the color to black, while popping restores the color state to that of before the break.

51
```
\def\pushcolor%
  {\stopcolormode}
```

52
```
\def\popcolor%
  {\doifsomething{\botcolormark}
     {\debuggerinfo{\m!colors}{popping \botcolormark}%
      \startcolormode{\botcolormark}}}
```

53
```
\def\popsplitcolor%
  {\getsplitmarks\colormark  % hier wel
   \doifsomething{\botcolormark}
     {\debuggerinfo{\m!colors}{split popping \botcolormark}%
      \startcolormode{\botcolormark}}}
```

colo-ini
colo-rgb
colo-xwi
colo-mwi
colo-pra

contents register context syst mult supp lang font colo spec core cont m s exit go back

\startcolorpage
\stopcolorpage

Local use can be forced with the next two macros. Nesting is still supported but colors are no longer marked.

54
```
\def\startcolorpage%
  {\bgroup
   \def\docolormark##1{}%
   \edef\savedcolorlevel{\the\colorlevel}%
   \ifnum\colorlevel>0
     \stopcolormode
   \fi
   \global\colorlevel=0\relax}
```

55
```
\def\stopcolorpage%
  {\global\colorlevel=\savedcolorlevel\relax
   \egroup}
```

\color
\gray

This leaves the simple color command:

```
\kleur[...]
...      tekst
```

```
\grijs[...]
...      tekst
```

Which can be used straightforward: green as gras. We want color support to be similar to font support and therefore implement `\color` as:

colo-ini
colo-rgb
colo-xwi
colo-mwi
colo-pra

contents register context syst mult supp lang font colo spec core cont m s exit go back

colo-ini
colo-rgb
colo-xwi
colo-mwi
colo-pra

56
```
\unexpanded\def\color[#1]%
  {\groupedcommand
     {\startcolor[#1]}
     {\stopcolor}}
```

57
```
\unexpanded\def\gray[#1]%
  {\groupedcommand
     {\RGBsupportedfalse\CMYKsupportedfalse\startcolor[#1]}
     {\stopcolor}}
```

This implementation enables use of defined colors like:

```
Look at the {\brightgreen bright} side of life and get
yourself no \red{red} head!
```

\colorvalue
\grayvalue

We can typeset the color components using `\colorvalue` and `\grayvalue`. The commands:

```
color value of SomeKindOfRed: \colorvalue{SomeKindOfRed} \crlf
gray value of SomeKindOfRed: \grayvalue{SomeKindOfRed}
```

show us:

color value of SomeKindOfRed:
gray value of SomeKindOfRed:

58
```
\def\realcolorformat#1%
  {\ifnum#1<10        0.00\the#1%
   \else\ifnum#1<100  0.0\the#1%
   \else\ifnum#1<1000 0.\the#1%
   \else              1.000\fi\fi\fi}
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

59
```
\def\colorformatseparator{ }
```

60
```
\def\dodoformatcolor#1%
  {\scratchdimen=#1\s!pt\relax
   \ifdim\scratchdimen>1\s!pt
     \scratchdimen=1\s!pt
   \fi
   \multiply\scratchdimen by 1000
   \scratchcounter=\scratchdimen
   \advance\scratchcounter by \!!medcard
   \divide\scratchcounter by \!!maxcard \relax
   \realcolorformat\scratchcounter}
```

61
```
\def\doformatcolorR#1:#2:#3\od%
  {\dodoformatcolor{#1}\colorformatseparator
   \dodoformatcolor{#2}\colorformatseparator
   \dodoformatcolor{#3}}
```

62
```
\def\doformatcolorC#1:#2:#3:#4\od%
  {\dodoformatcolor{#1}\colorformatseparator
   \dodoformatcolor{#2}\colorformatseparator
   \dodoformatcolor{#3}\colorformatseparator
   \dodoformatcolor{#4}}
```

63
```
\def\doformatcolorK#1\od%
  {\dodoformatcolor{#1}}
```

64
```
\def\doformatcolor#1:%
  {\getvalue{doformatcolor#1}}
```

colo-ini
colo-rgb
colo-xwi
colo-mwi
colo-pra

contents register context syst mult supp lang font colo spec core cont m s exit go back

colo-ini
colo-rgb
colo-xwi
colo-mwi
colo-pra

```
65 \def\colorvalue#1%
     {\doifcolorelse{#1}
        {\getcurrentcolorspecs{#1}%
         \expandafter\doformatcolor\currentcolorspecs\od}
        {}}

66 \def\doformatgrayR#1:#2:#3\od%
     {\convertRGBtoGRAY{#1}{#2}{#3}%
      \dodoformatcolor\@@cl@@k}

67 \def\doformatgrayC#1:#2:#3:#4\od%
     {\convertCMYKtoGRAY{#1}{#2}{#3}{#4}%
      \dodoformatcolor\@@cl@@k}

68 \def\doformatgrayK#1\od%
     {\dodoformatcolor{#1}}

69 \def\doformatgray#1:%
     {\getvalue{doformatgray#1}}

70 \def\grayvalue#1%
     {\doifcolorelse{#1}
        {\getcurrentcolorspecs{#1}%
         \expandafter\doformatgray\currentcolorspecs\od}
        {}}
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

\locatstartraster
\localstopraster
\startraster
\stopraster

colo-ini
colo-rgb
colo-xwi
colo-mwi
colo-pra

The previous conversions are not linear and treat each color component according to human perception curves. Pure gray (we call them rasters) has equal color components. In CONTEXT rasters are only used as backgrounds and these don't cross page boundaries in the way color does. Therefore we don't need stacks and marks. Just to be compatible with color support we offer both 'global' and 'local' commands.

71
```
\def\localstartraster[#1]%
  {\doifinstringelse{.}{#1}
     {\dostartgraymode{#1}}
     {\dostartgraymode{\@@rsraster}}}
```

72
```
\def\localstopraster%
  {\dostopgraymode}
```

73
```
\def\startraster%
  {\localstartraster}
```

74
```
\def\stopraster%
  {\localstopraster}
```

In this documentation we will not go into too much details on palets. Curious users can find more information on this topic in [use of color].

At the moment we implemented color in CONTEXT color printing was not yet on the desktop. In spite of this lack our graphics designer made colorfull illustrations. When printed on a black and white printer, distinctive colors can come out equally gray. We therefore decided to use only colors that were distinctive in colors as well as in black and white print.

Although none of the graphic packages we used supported logical colors and global color redefition, we build this support into CONTEXT. This enabled us to experiment and also prepared us for the future.

contents register context syst mult supp lang font colo spec core cont m s exit go back

\definepalet

Colors are grouped in palets. The colors in such a palet can have colorful names, but best is to use names that specify their use, like *important* or *danger*. As a sort of example CONTEXT has some palets predefined, like:[6]

```
\definepalet
  [alfa]
  [      top=rood:7,
     bottom=groen:6,
         up=blauw:5,
       down=cyaan:4,
    strange=magenta:3,
      charm=geel:2]
```

It's formal definition is:

```
\definieerpalet[..,..=..,..]

naam     naam
```

Visualized, such a palet looks like:

	top	bottom	up	down	strange	charm
alfa	🟥	🟩	🟦	🟦	🟪	🟨
	1.000	0.000	0.150	0.000	1.000	1.000
	0.150	0.800	0.750	0.950	0.650	1.000
	0.150	0.000	1.000	0.950	1.000	0.000

[6] At the time I wrote the palet support, I was reading 'A hort history of time' of S. Hawkins, so that's why we stuck to quarks.

colo-ini
colo-rgb
colo-xwi
colo-mwi
colo-pra

This bar shows both the color and gray alternatives of the palet components (not visible in black and white print).

75
```
\def\definepalet%
  {\dodoubleargument\dodefinepalet}
```

76
```
\def\dodefinepalet[#1][#2]%
  {\setvalue{\??pa#1}{}%
   \showmessage{\m!colors}{6}{#1}%
   \def\dododefinepalet[##1=##2]%
     {\doifvaluesomething{\??pa#1}
        {\setevalue{\??pa#1}{\getvalue{\??pa#1},}}%
      \setevalue{\??pa#1}{\getvalue{\??pa#1}##1}%
     \doifdefinedelse{\??cr##2}
       {\setevalue{\??cr#1:##1}{\getvalue{\??cr##2}}}
       {\setevalue{\??cr#1:##1}{G:0}}}%
   \def\dodefinepalet##1%
     {\dododefinepalet[##1]}%
   \processcommalist[#2]\dodefinepalet}
```

\setuppalet Colors are taken from the current palet, if defined. Setting the current palet is done by:

```
\stelpaletin[...]

...      naam
```

77
```
\def\currentpalet{}
```

78
```
\def\setuppalet%
  {\dosingleempty\dosetuppalet}
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

colo-ini
colo-rgb
colo-xwi
colo-mwi
colo-pra

79
```
\def\dosetuppalet[#1]%
  {\doifelsenothing{#1}
     {\def\currentpalet{}}
     {\doifelsevaluenothing{\??pa#1}
        {\showmessage{\m!colors}{7}{#1}%
         \def\currentpalet{}}
        {\def\currentpalet{#1:}}}}
```

\showpalet

The previous visualization was typeset with:

```
  \showpalet [alfa] [horizontaal,naam,nummer,waarde]
```

This commands is defined as:

```
\toonpalet[.1.][..,.2.,..]

.1.      naam
.2.      horizontaal vertikaal naam waarde
```

80
```
\def\showpalet%
  {\dodoubleargument\doshowpalet}
```

81
```
\def\doshowpalet[#1][#2]%
  {\doifdefined{\??pa#1}
     {\doifinsetelse{\v!vertikaal}{#2}
        {\showverticalpalet[#1][#2]}
        {\showhorizontalpalet[#1][#2]}}}
```

82
```
\def\showhorizontalpalet[#1][#2]%
  {\localvbox
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
{\offinterlineskip
 \!!widtha=\hsize
 \doifinsetelse{\v!naam}{#2}
   {\!!widthb=5em}
   {\!!widthb=\!!zeropoint}
 \advance\!!widtha by -\!!widthb
 \getcommacommandsize[\getvalue{\??pa#1}]
 \divide\!!widtha by \commalistsize
 \setuppalet[#1]
 \doifinset{\v!nummer}{#2}%
   {\def\doshowpalet##1%
      {\hbox to \!!widtha{\hss\strut##1\hss}}
    \hbox{\hskip\!!widthb\processpalet[#1]\doshowpalet}
    \endgraf}
 \def\doshowpalet##1%
   {\color[##1]{\vrule\!!width\!!widtha\!!height\ht\strutbox}}
 \hbox
   {\ifdim\!!widthb>\!!zeropoint\relax
      \hbox to \!!widthb{\hss#1\hskip.75em}%
    \fi
    \processpalet[#1]\doshowpalet}
 \endgraf
 \def\doshowpalet##1%
   {\gray[##1]{\vrule\!!width\!!widtha\!!depth\dp\strutbox}}
 \hbox{\hskip\!!widthb\processpalet[#1]\doshowpalet}
 \endgraf
 \doifinset{\v!waarde}{#2}
   {\def\doshowpalet##1%
      {\vbox
```

```
               {\hsize\!!widtha
                \vskip.25ex
                \everypar{\strut}
                \veryraggedcenter
                \let\colorformatseparator=\endgraf
                \colorvalue{##1}}}
          \hbox{\hskip\!!widthb\processpalet[#1]\doshowpalet}}}}
```

83
```
\def\showverticalpalet[#1][#2]%
  {\localvbox
     {\offinterlineskip
      \setuppalet[#1]
      \def\rule%
        {\vrule\!!width3em\!!height\ht\strutbox\!!depth\dp\strutbox}
      \doifinsetelse{\v!nummer}{#2}
        {\!!widthb=5em}
        {\!!widthb=\!!zeropoint}
      \advance\!!widtha by \!!widthb
      \doifinset{\v!naam}{#2}
        {\hbox{\hskip\!!widthb\hbox to 6em{\hss\strut#1\hss}}
         \endgraf}
      \def\doshowpalet##1%
        {\hbox
           {\ifdim\!!widthb>\!!zeropoint
              \hbox to \!!widthb{\hss##1\hskip.75em}%
            \fi
            \color[##1]{\rule}%
            \gray[##1]{\rule}%
            \doifinset{\v!waarde}{#2}%
              {\hbox to 7em{\hskip.75em\colorvalue{##1}\hss}}}
```

colo-ini
colo-rgb
colo-xwi
colo-mwi
colo-pra

contents register context syst mult supp lang font colo spec core cont m s exit go back

colo-ini
colo-rgb
colo-xwi
colo-mwi
colo-pra

```
        \endgraf}
      \processpalet[#1]\doshowpalet}}
```

84
```
\def\processpalet[#1]%
  {\processcommacommand[\getvalue{\??pa#1}]}
```

\definecolorgroup

The naming of the colors in this palet suggests some ordening, which in turn is suported by color grouping.

```
   \definecolorgroup
     [rood]
     [1.00:0.90:0.90,
      1.00:0.80:0.80,
      1.00:0.70:0.70,
      1.00:0.55:0.55,
      1.00:0.40:0.40,
      1.00:0.25:0.25,
      1.00:0.15:0.15,
      0.90:0.00:0.00]
```

In such a color group colors are numbered from 1 to n.

```
\definieerkleurgroep[.1.][.2.][..,.3.,..]

.1.     naam
.2.     rgb cmyk k
.3.     [x:y:z=,..]
```

This kind of specification is not only more compact than defining each color separate, it also loads faster and takes less bytes.

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
85 \def\definecolorgroup%
     {\dotripleempty\dodefinecolorgroup}

86 \def\dodefinecolorgroup[#1][#2][#3]%
     {\ifthirdargument
        \processaction
          [#2]
          [     \v!cmyk=>\edef\currentcolorspace{C},
                 \v!rgb=>\edef\currentcolorspace{R},
                \v!gray=>\edef\currentcolorspace{K},
             \s!unknown=>\edef\currentcolorspace{R}]%
        \scratchcounter=0
        \def\dododefinecolorgroup##1%
          {\advance\scratchcounter by 1
           \setevalue{\??cr#1:\the\scratchcounter}{\currentcolorspace:##1}}%
        \processcommalist[#3]\dododefinecolorgroup
      \else
        \doifinstringelse{:}{#2}
          {\definecolorgroup[#1][\v!rgb][#2]}
          {\doloop
             {\doifdefinedelse{\??cr#2:\recurselevel}
                {\setevalue{\??cr#1:\recurselevel}%
                   {\getvalue{\??cr#2:\recurselevel}}}
                {\exitloop}}}%
      \fi}
```

colo-ini
colo-rgb
colo-xwi
colo-mwi
colo-pra

\showcolorgroup

We can show the group by:

```
\showcolorgroup [blauw] [horizontaal,naam,nummer,waarde]
```

or in color:

	1	2	3	4	5	6	7	8
blauw	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
	0.900	0.800	0.550	0.300	0.150	0.000	0.000	0.000
	0.950	0.900	0.850	0.800	0.750	0.700	0.550	0.400
	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000

which uses:

```
\toonkleurgroep[.1.][..,.2.,..]

.1.     naam
.2.     horizontaal vertikaal naam waarde nummer
```

87
```
\def\showcolorgroup%
  {\dodoubleargument\doshowcolorgroup}
```

88
```
\def\doshowcolorgroup[#1][#2]%
  {\doifdefined{\??cr#1:1}
     {\doifinsetelse{\v!vertikaal}{#2}
        {\showverticalcolorgroup[#1][#2]}
        {\showhorizontalcolorgroup[#1][#2]}}}
```

89
```
\def\showhorizontalcolorgroup[#1][#2]%
  {\localvbox
```

```
{\offinterlineskip
 \!!widtha=\hsize
 \setuppalet
 \!!counta=0
 \dorecurse{15}
   {\doifdefined{\??cr#1:\recurselevel}{\advance\!!counta by 1}}
 \doifinsetelse{\v!naam}{#2}
   {\!!widthb=5em}
   {\!!widthb=\!!zeropoint}
\advance\!!widtha by -\!!widthb
\divide\!!widtha by \!!counta
\doifinset{\v!nummer}{#2}
  {\hbox
     {\hskip\!!widthb
      \dorecurse{\!!counta}
        {\hbox to \!!widtha{\hss\strut\recurselevel\hss}}}
   \endgraf}
\hbox
  {\ifdim\!!widthb>\!!zeropoint
     \hbox to \!!widthb{\hss#1\hskip.75em}%
   \fi
   \dorecurse{\!!counta}
     {\vbox
        {\hsize\!!widtha
         \color[#1:\recurselevel]
           {\vrule\!!width\!!widtha\!!height\ht\strutbox}
         \endgraf
         \gray[#1:\recurselevel]
           {\vrule\!!width\!!widtha\!!depth\dp\strutbox}}}}%
```

colo-ini
colo-rgb
colo-xwi
colo-mwi
colo-pra

contents register context syst mult supp lang font colo spec core cont m s exit go back

colo-ini
colo-rgb
colo-xwi
colo-mwi
colo-pra

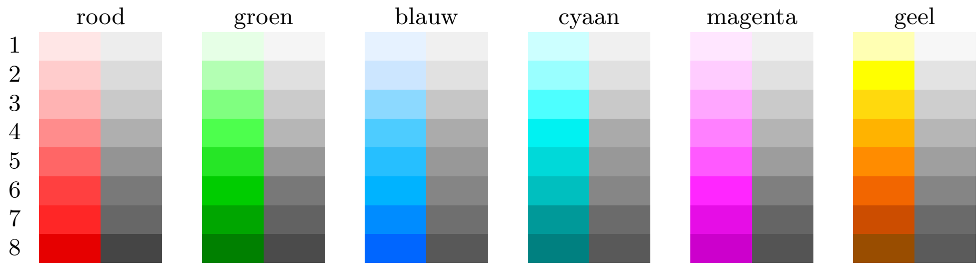
```
     \endgraf
     \doifinset{\v!waarde}{#2}
       {\hbox
          {\hskip\!!widthb
           \dorecurse{\!!counta}
             {\vbox
                {\hsize\!!widtha
                 \vskip.25ex
                 \everypar{\strut}
                 \veryraggedcenter
                 \let\colorformatseparator=\endgraf
                 \colorvalue{#1:\recurselevel}}}}}}}

\def\showverticalcolorgroup[#1][#2]%
  {\localvbox
     {\offinterlineskip
      \setuppalet
      \def\rule%
        {\vrule\!!width2.5em\!!height\ht\strutbox\!!depth\dp\strutbox}
      \doifinsetelse{\v!nummer}{#2}
        {\!!widthb=2em}
        {\!!widthb=\!!zeropoint}
      \doifinset{\v!naam}{#2}
        {\hbox{\hskip\!!widthb\hbox to 5em{\hss\strut#1\hss}}
         \endgraf}
      \dorecurse{15}
        {\doifdefined{\??cr#1:\recurselevel}
           {\hbox
              {\ifdim\!!widthb>\!!zeropoint
                 \hbox to \!!widthb{\hss\recurselevel\hskip.75em}%
```

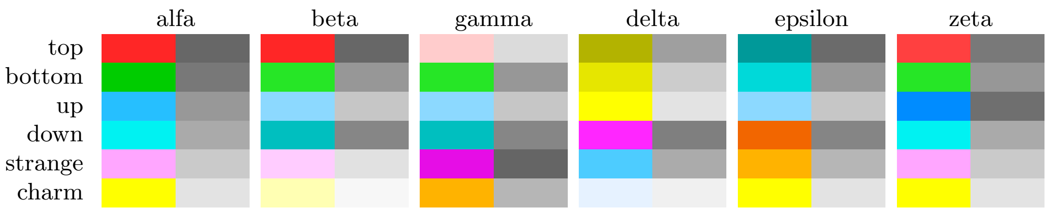
contents register context syst mult supp lang font colo spec core cont m s exit go back

```
        \fi
        \color[#1:\recurselevel]{\rule}%
        \gray[#1:\recurselevel]{\rule}%
        \doifinset{\v!waarde}{#2}%
          {\hbox to 7em{\hskip.75em\colorvalue{#1:\recurselevel}\hss}}}
    \endgraf}}}}
```

There are ten predefined color groups, like *groen*, *rood*, *blauw*, *cyaan*, *magenta* and *geel*.



These groups are used to define palets *alfa* upto *zeta*. As long as we don't use colors from the same row, we get ourselves distinctive palets. By activating such a palet one gains access to its members *top* to *charm* (of course one should use more suitable names than these).

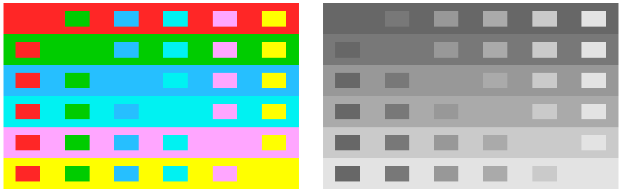


By using the keyword `\v!waarde` the individual color components are shown too. When printed in color, these showcases show both the colors and the gray value.

`\comparepalet` There are some more testing macros available:

```
\comparepalet [alfa]
```

shows the palet colors against a background:



The formal definition is:

```
\vergelijkpalet[...]

...       naam
```

91

```
\def\comparepalet%
  {\dosingleargument\docomparepalet}
```

92

```
\def\docomparepalet[#1]%
  {\doifdefined{\??pa#1}
     {\hbox
        {\dodocomparepalet\color[#1]%
```

colo-ini
colo-rgb
colo-xwi
colo-mwi
colo-pra

```
          \quad
          \dodocomparepalet\gray[#1]}}}
```

93
```
\def\dodocomparepalet#1[#2]%
  {\localvbox
     {\offinterlineskip
      \setuppalet[#2]
      \getcommacommandsize[\getvalue{\??pa#2}]
      \!!widtha=2em\relax
      \hsize=\commalistsize\!!widtha
      \def\rule%
        {\vrule\!!width.5\!!widtha\!!height2.25ex\!!depth-.75ex}
      \def\dododocomparepalet##1%
        {\hbox
           {\setbox0=\hbox
              {#1[##1]{\vrule\!!width\hsize\!!height3ex}}%
            \wd0=\!!zeropoint\box0
            \hbox to \hsize
              {\def\dododocomparepalet####1%
                 {\hbox to \!!widtha
                    {\hss#1[####1]{\rule}\hss}}%
               \processcommacommand[\getvalue{\??pa#2}]\dododocomparepalet}}
          \endgraf}
      \processcommacommand[\getvalue{\??pa#2}]\dododocomparepalet}}
```

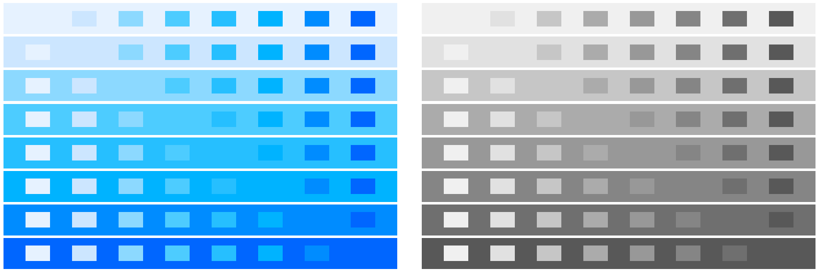
\comparecolorgroup

The similar command:

```
\comparecolorgroup [blauw]
```

shows color groups:

contents register context syst mult supp lang font colo spec core cont m s exit go back



this commands are defined as:

```
\vergelijkkleurgroep[...]

...     naam
```

94
```
\def\comparecolorgroup%
  {\dosingleargument\docomparecolorgroup}
```

95
```
\def\docomparecolorgroup[#1]%
  {\doifdefined{\??cr#1:1}
     {\hbox
        {\dodocomparecolorgroup\color[#1]%
         \quad
         \dodocomparecolorgroup\gray[#1]}}}
```

96
```
\def\dodocomparecolorgroup#1[#2]%
  {\localvbox
```

```
    {\!!counta=0
     \dorecurse{15}
       {\doifdefined{\??cr#2:\recurselevel}{\advance\!!counta by 1}}
     \!!widtha=2em\relax
     \hsize=\!!counta\!!widtha
     \def\rule%
       {\vrule\!!width.5\!!widtha\!!height2.25ex\!!depth-.75ex}
     \def\dododocomparecolorgroup##1%
       {\hbox to \hsize
          {\setbox0=\hbox
             {#1[#2:##1]{\vrule\!!width\hsize\!!height3ex}}%
           \wd0=\!!zeropoint\box0
           \hbox to \hsize
             {\hss\dorecurse{\!!counta}{#1[#2:\recurselevel]{\rule}\hss}}}
        \endgraf}
     \dorecurse{\!!counta}{\dododocomparecolorgroup\recurselevel}}}
```

\showcolor But let's not forget that we also have the more traditional non–related colors. These show up after:

```
    \showcolor [name]
```

Where name for instance can be rgb.

```
\toonkleur[...]

...      naam
```

97
```
\def\showcolor[#1]%
  {\bgroup
```

colo-ini
colo-rgb
colo-xwi
colo-mwi
colo-pra

```
     \setupcolor[#1]
     \stelwitruimtein[\v!geen]
     \def\rule%
       {\vrule\!!width4em\!!height\ht\strutbox\!!depth\dp\strutbox}
     \def\docommand##1%
       {\hbox
          {\gray[##1]{\rule}\quad
           \color[##1]{\rule}\quad
           \grayvalue{##1}\quad
           \hbox to 12em{\colorvalue{##1}\hss}%
           \strut##1}
        \endgraf}
     \processcommacommand[\colorlist]\docommand
   \egroup}
```

We default to the colors defined in `colo-rgb` and support both RGB and CMYK output.

98
```
\setupcolor
  [\v!rgb]
```

99
```
\setupcolors
  [\c!status=\v!stop,
   \c!conversie=\v!ja,
   \c!rgb=\v!ja,
   \c!cmyk=\v!ja]
```

As we can see, color support is turned off by default. Reduction of gray colors to gray scales is turned on.

100
```
\protect
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

7.2 RGB

colo-ini
colo-rgb
colo-xwi
colo-mwi
colo-pra

Just to give users a start we define some colors. While switching fonts is as international as can be, thanks to the mnemonics, naming colors is very interface dependant. To support international setups, we define both english and interface dependant colors. We use the color inheritance mechanisms to implement the interface dependant ones.

First we define some simple primary RGB and CMYK colors. All colors are defined in RGB color space.

1
```
\definecolor [red]           [r=1,   g=0,   b=0]
\definecolor [green]         [r=0,   g=1,   b=0]
\definecolor [blue]          [r=0,   g=0,   b=1]
```

2
```
\definecolor [cyan]          [r=0,   g=1,   b=1]
\definecolor [magenta]       [r=1,   g=0,   b=1]
\definecolor [yellow]        [r=1,   g=1,   b=0]
```

3
```
\definecolor [white]         [r=1,   g=1,   b=1]
\definecolor [black]         [r=0,   g=0,   b=0]
```

4
```
\definecolor [darkred]       [r=.8,  g=0,   b=0]
\definecolor [middlered]     [r=.9,  g=0,   b=0]
\definecolor [lightred]      [r=1,   g=0,   b=0]
```

5
```
\definecolor [darkgreen]     [r=0,   g=.6,  b=0]
\definecolor [middlegreen]   [r=0,   g=.8,  b=0]
\definecolor [lightgreen]    [r=0,   g=1,   b=0]
```

6
```
\definecolor [darkblue]      [r=0,   g=0,   b=.8]
\definecolor [middleblue]    [r=0,   g=0,   b=.9]
```

colo-ini
colo-rgb
colo-xwi
colo-mwi
colo-pra

```
   \definecolor [lightblue]     [r=0,  g=0,  b=1]

7  \definecolor [darkcyan]      [r=.6, g=.8, b=.8]
   \definecolor [middlecyan]    [r=0,  g=.8, b=.8]

8  \definecolor [darkmagenta]   [r=.8, g=.6, b=.8]
   \definecolor [middlemagenta] [r=1,  g=0,  b=.6]

9  \definecolor [darkyellow]    [r=.8, g=.8, b=.6]
   \definecolor [middleyellow]  [r=1,  g=1,  b=.2]

10 \definecolor [darkgray]      [r=.5, g=.5, b=.5]
   \definecolor [middlegray]    [r=.7, g=.7, b=.7]
   \definecolor [lightgray]     [r=.9, g=.9, b=.9]
```

These colors are mapped to interface dependant colornames.

```
11 \startinterface dutch

12   \definecolor [rood]          [red]
     \definecolor [groen]         [green]
     \definecolor [blauw]         [blue]

13   \definecolor [cyaan]         [cyan]
     \definecolor [magenta]       [magenta]
     \definecolor [geel]          [yellow]

14   \definecolor [wit]           [white]
     \definecolor [zwart]         [black]

15   \definecolor [donkerrood]    [darkred]
     \definecolor [middelrood]    [middlered]
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
  \definecolor [lichtrood]     [lightred]

  \definecolor [donkergroen]   [darkgreen]
  \definecolor [middelgroen]   [middlegreen]
  \definecolor [lichtgroen]    [lightgreen]

  \definecolor [donkerblauw]   [darkblue]
  \definecolor [middelblauw]   [middleblue]
  \definecolor [lichtblauw]    [lightblue]

  \definecolor [donkercyaan]   [darkcyan]
  \definecolor [middelcyaan]   [middlecyan]

  \definecolor [donkermagenta] [darkmagenta]
  \definecolor [middelmagenta] [middlemagenta]

  \definecolor [donkergeel]    [darkyellow]
  \definecolor [middelgeel]    [middleyellow]

  \definecolor [donkergrijs]   [darkgray]
  \definecolor [middengrijs]   [middlegray]
  \definecolor [lichtgrijs]    [lightgray]

\stopinterface

\startinterface german

  \definecolor [rot]           [red]
  \definecolor [gruen]         [green]
  \definecolor [blau]          [blue]

  \definecolor [cyan]          [cyan]
  \definecolor [magenta]       [magenta]
```

```
    \definecolor [gelb]          [yellow]

26  \definecolor [weiss]         [white]
    \definecolor [schwarz]       [black]

27  \definecolor [dunkelrot]     [darkred]
    \definecolor [mittelrot]     [middlered]
    \definecolor [hellrot]       [lightred]

28  \definecolor [dunkelgruen]   [darkgreen]
    \definecolor [mittelgruen]   [middlegreen]
    \definecolor [hellgruen]     [lightgreen]

29  \definecolor [dunkelblau]    [darkblue]
    \definecolor [mittelblau]    [middleblue]
    \definecolor [hellblau]      [lightblue]

30  \definecolor [dunkelcyan]    [darkcyan]
    \definecolor [mittelcyan]    [middlecyan]

31  \definecolor [dunkelmagenta] [darkmagenta]
    \definecolor [mittelmagenta] [middlemagenta]

32  \definecolor [dunkelgelb]    [darkyellow]
    \definecolor [mittelgelb]    [middleyellow]

33  \definecolor [dunkelgrau]    [darkgray]
    \definecolor [mittelgrau]    [middlegray]
    \definecolor [hellgrau]      [lightgrey]
```

34
```
\stopinterface
```

Like colors, we first define the english colorgroups. These colorgroups are tuned for distinctive gray scale printing.

35
```
\definecolorgroup
  [gray]
  [0.95:0.95:0.95,
   0.90:0.90:0.90,
   0.80:0.80:0.80,
   0.70:0.70:0.70,
   0.60:0.60:0.60,
   0.50:0.50:0.50,
   0.40:0.40:0.40,
   0.30:0.30:0.30,
   0.20:0.20:0.20,
   0.10:0.10:0.10,
   0.00:0.00:0.00]
```

36
```
\definecolorgroup
  [red]
  [1.00:0.90:0.90,
   1.00:0.80:0.80,
   1.00:0.70:0.70,
   1.00:0.55:0.55,
   1.00:0.40:0.40,
   1.00:0.25:0.25,
   1.00:0.15:0.15,
   0.90:0.00:0.00]
```

colo-ini
colo-rgb
colo-xwi
colo-mwi
colo-pra

37
```
\definecolorgroup
  [green]
  [0.90:1.00:0.90,
   0.70:1.00:0.70,
   0.50:1.00:0.50,
   0.30:1.00:0.30,
   0.15:0.90:0.15,
   0.00:0.80:0.00,
   0.00:0.65:0.00,
   0.00:0.50:0.00]
```

38
```
\definecolorgroup
  [blue]
  [0.90:0.95:1.00,
   0.80:0.90:1.00,
   0.55:0.85:1.00,
   0.30:0.80:1.00,
   0.15:0.75:1.00,
   0.00:0.70:1.00,
   0.00:0.55:1.00,
   0.00:0.40:1.00]
```

39
```
\definecolorgroup
  [cyan]
  [0.80:1.00:1.00,
   0.60:1.00:1.00,
   0.30:1.00:1.00,
   0.00:0.95:0.95,
   0.00:0.85:0.85,
   0.00:0.75:0.75,
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

colo-ini
colo-rgb
colo-xwi
colo-mwi
colo-pra

```
    0.00:0.60:0.60,
    0.00:0.50:0.50]
```

40
```
\definecolorgroup
  [magenta]
  [1.00:0.90:1.00,
   1.00:0.80:1.00,
   1.00:0.65:1.00,
   1.00:0.50:1.00,
   1.00:0.35:1.00,
   1.00:0.15:1.00,
   0.90:0.05:0.90,
   0.80:0.00:0.80]
```

41
```
\definecolorgroup
   [yellow]
   [1.00:1.00:0.70,
    1.00:1.00:0.00,
    1.00:0.85:0.05,
    1.00:0.70:0.00,
    1.00:0.55:0.00,
    0.95:0.40:0.00,
    0.80:0.30:0.00,
    0.60:0.30:0.00]
```

42
```
\definecolorgroup
  [red*]
  [1.00:0.95:0.95,
   1.00:0.90:0.90,
   1.00:0.80:0.80,
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

colo-ini
colo-rgb
colo-xwi
colo-mwi
colo-pra

```
   1.00:0.70:0.70,
   1.00:0.60:0.60,
   1.00:0.50:0.50,
   1.00:0.40:0.40,
   1.00:0.30:0.30]
```

43
```
\definecolorgroup
  [green*]
  [0.95:1.00:0.95,
   0.90:1.00:0.90,
   0.80:1.00:0.80,
   0.70:1.00:0.70,
   0.60:1.00:0.60,
   0.50:1.00:0.50,
   0.40:1.00:0.40,
   0.30:1.00:0.30]
```

44
```
\definecolorgroup
  [blue*]
  [0.95:0.95:1.00,
   0.90:0.90:1.00,
   0.80:0.80:1.00,
   0.70:0.70:1.00,
   0.60:0.60:1.00,
   0.50:0.50:1.00,
   0.40:0.40:1.00,
   0.30:0.30:1.00]
```

45
```
\definecolorgroup
  [yellow*]
  [1.00:1.00:0.10,
   1.00:1.00:0.00,
   0.90:0.90:0.00,
   0.80:0.80:0.00,
   0.70:0.70:0.00,
   0.60:0.60:0.00,
   0.50:0.50:0.00,
   0.40:0.40:0.00]
```

For the sake of implementing interface dependant color groups we support colorgroup duplication.

46
```
\startinterface dutch
  \definecolorgroup [grijs]   [gray]
  \definecolorgroup [rood]    [red]
  \definecolorgroup [groen]   [green]
  \definecolorgroup [blauw]   [blue]
  \definecolorgroup [cyaan]   [cyan]
  \definecolorgroup [magenta] [magenta]
  \definecolorgroup [geel]    [yellow]
  \definecolorgroup [rood*]   [red*]
  \definecolorgroup [groen*]  [green*]
  \definecolorgroup [blauw*]  [blue*]
  \definecolorgroup [geel*]   [yellow*]
\stopinterface
```

47
```
\startinterface german
  \definecolorgroup [grau]    [gray]
  \definecolorgroup [rot]     [red]
  \definecolorgroup [gruen]   [green]
  \definecolorgroup [blau]    [blue]
```

colo-ini
colo-rgb
colo-xwi
colo-mwi
colo-pra

```
  \definecolorgroup [cyan]     [cyan]
  \definecolorgroup [magenta]  [magenta]
  \definecolorgroup [gelb]     [yellow]
  \definecolorgroup [rot*]     [red*]
  \definecolorgroup [gruen*]   [green*]
  \definecolorgroup [blau*]    [blue*]
  \definecolorgroup [gelb*]    [yellow*]
\stopinterface
```

The next set of color palets is quite language independant. These palets are meant as examples.

48
```
\definepalet
  [alfa]
  [     top=red:7,
     bottom=green:6,
         up=blue:5,
       down=cyan:4,
    strange=magenta:3,
      charm=yellow:2]
```

49
```
\definepalet
  [beta]
  [     top=red:7,
     bottom=green:5,
         up=blue:3,
       down=cyan:6,
    strange=magenta:2,
      charm=yellow:1]
```

50
```
\definepalet
  [gamma]
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
  [     top=red:2,
     bottom=green:5,
         up=blue:3,
       down=cyan:6,
    strange=magenta:7,
      charm=yellow:4]
```

51
```
\definepalet
  [delta]
  [     top=yellow*:5,
     bottom=yellow*:3,
         up=yellow*:2,
       down=magenta:6,
    strange=blue:4,
      charm=blue:1]
```

52
```
\definepalet
  [epsilon]
  [     top=cyan:7,
     bottom=cyan:5,
         up=blue:3,
       down=yellow:6,
    strange=yellow:4,
      charm=yellow:2]
```

53
```
\definepalet
  [zeta]
  [     top=red:6,
     bottom=green:5,
         up=blue:7,
```

colo-ini
colo-rgb
colo-xwi
colo-mwi
colo-pra

```
      down=cyan:4,
   strange=magenta:3,
     charm=yellow:2]
```

The next four colors are used for typesetting verbatim TEX in color.

54
```
\definecolor [texcolorone]   [middlered]
\definecolor [texcolortwo]   [middlegreen]
\definecolor [texcolorthree] [middleblue]
\definecolor [texcolorfour]  [darkyellow]
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

7.3 X Windows

colo-ini
colo-rgb
colo-xwi
colo-mwi
colo-pra

I've forgotten where I got these definitions from, but maybe they can be of use.

```
\definieerkleur [aliceblue]          [r=0.94,g=0.97,b=1.00]
\definieerkleur [antiquewhite]       [r=0.98,g=0.92,b=0.84]
\definieerkleur [aquamarine]         [r=0.50,g=1.00,b=0.83]
\definieerkleur [azure]              [r=0.94,g=1.00,b=1.00]
\definieerkleur [beige]              [r=0.96,g=0.96,b=0.86]
\definieerkleur [bisque]             [r=1.00,g=0.89,b=0.77]
\definieerkleur [black]              [r=0.00,g=0.00,b=0.00]
\definieerkleur [blanchedalmond]     [r=1.00,g=0.92,b=0.80]
\definieerkleur [blue]               [r=0.00,g=0.00,b=1.00]
\definieerkleur [blueviolet]         [r=0.54,g=0.17,b=0.89]
\definieerkleur [brown]              [r=0.65,g=0.16,b=0.16]
\definieerkleur [burlywood]          [r=0.87,g=0.72,b=0.53]
\definieerkleur [cadetblue]          [r=0.37,g=0.62,b=0.63]
\definieerkleur [chartreuse]         [r=0.50,g=1.00,b=0.00]
\definieerkleur [chocolate]          [r=0.82,g=0.41,b=0.12]
\definieerkleur [coral]              [r=1.00,g=0.50,b=0.31]
\definieerkleur [cornflowerblue]     [r=0.39,g=0.58,b=0.93]
\definieerkleur [cornsilk]           [r=1.00,g=0.97,b=0.86]
\definieerkleur [cyan]               [r=0.00,g=1.00,b=1.00]
\definieerkleur [darkgoldenrod]      [r=0.72,g=0.53,b=0.04]
\definieerkleur [darkgreen]          [r=0.00,g=0.39,b=0.00]
\definieerkleur [darkkhaki]          [r=0.74,g=0.72,b=0.42]
\definieerkleur [darkolivegreen]     [r=0.33,g=0.42,b=0.18]
\definieerkleur [darkorange]         [r=1.00,g=0.55,b=0.00]
\definieerkleur [darkorchid]         [r=0.60,g=0.20,b=0.80]
\definieerkleur [darksalmon]         [r=0.91,g=0.59,b=0.48]
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
\definieerkleur [darkseagreen]          [r=0.56,g=0.74,b=0.56]
\definieerkleur [darkslateblue]         [r=0.28,g=0.24,b=0.55]
\definieerkleur [darkturquoise]         [r=0.00,g=0.81,b=0.82]
\definieerkleur [darkviolet]            [r=0.58,g=0.00,b=0.83]
\definieerkleur [deeppink]              [r=1.00,g=0.08,b=0.58]
\definieerkleur [deepskyblue]           [r=0.00,g=0.75,b=1.00]
\definieerkleur [dodgerblue]            [r=0.12,g=0.56,b=1.00]
\definieerkleur [firebrick]             [r=0.70,g=0.13,b=0.13]
\definieerkleur [floralwhite]           [r=1.00,g=0.98,b=0.94]
\definieerkleur [forestgreen]           [r=0.13,g=0.55,b=0.13]
\definieerkleur [gainsboro]             [r=0.86,g=0.86,b=0.86]
\definieerkleur [ghostwhite]            [r=0.97,g=0.97,b=1.00]
\definieerkleur [gold]                  [r=1.00,g=0.84,b=0.00]
\definieerkleur [goldenrod]             [r=0.85,g=0.65,b=0.13]
\definieerkleur [green]                 [r=0.00,g=1.00,b=0.00]
\definieerkleur [greenyellow]           [r=0.68,g=1.00,b=0.18]
\definieerkleur [honeydew]              [r=0.94,g=1.00,b=0.94]
\definieerkleur [hotpink]               [r=1.00,g=0.41,b=0.71]
\definieerkleur [indianred]             [r=0.80,g=0.36,b=0.36]
\definieerkleur [ivory]                 [r=1.00,g=1.00,b=0.94]
\definieerkleur [khaki]                 [r=0.94,g=0.90,b=0.55]
\definieerkleur [lavender]              [r=0.90,g=0.90,b=0.98]
\definieerkleur [lavenderblush]         [r=1.00,g=0.94,b=0.96]
\definieerkleur [lawngreen]             [r=0.49,g=0.99,b=0.00]
\definieerkleur [lemonchiffon]          [r=1.00,g=0.98,b=0.80]
\definieerkleur [lightblue]             [r=0.68,g=0.85,b=0.90]
\definieerkleur [lightcoral]            [r=0.94,g=0.50,b=0.50]
\definieerkleur [lightcyan]             [r=0.88,g=1.00,b=1.00]
\definieerkleur [lightgoldenrod]        [r=0.93,g=0.87,b=0.51]
```

```
\definieerkleur [lightgoldenrodyellow] [r=0.98,g=0.98,b=0.82]
\definieerkleur [lightpink]            [r=1.00,g=0.71,b=0.76]
\definieerkleur [lightsalmon]          [r=1.00,g=0.63,b=0.48]
\definieerkleur [lightseagreen]        [r=0.13,g=0.70,b=0.67]
\definieerkleur [lightskyblue]         [r=0.53,g=0.81,b=0.98]
\definieerkleur [lightslateblue]       [r=0.52,g=0.44,b=1.00]
\definieerkleur [lightsteelblue]       [r=0.69,g=0.77,b=0.87]
\definieerkleur [lightyellow]          [r=1.00,g=1.00,b=0.88]
\definieerkleur [limegreen]            [r=0.20,g=0.80,b=0.20]
\definieerkleur [linen]                [r=0.98,g=0.94,b=0.90]
\definieerkleur [magenta]              [r=1.00,g=0.00,b=1.00]
\definieerkleur [maroon]               [r=0.69,g=0.19,b=0.38]
\definieerkleur [mediumaquamarine]     [r=0.40,g=0.80,b=0.67]
\definieerkleur [mediumblue]           [r=0.00,g=0.00,b=0.80]
\definieerkleur [mediumorchid]         [r=0.73,g=0.33,b=0.83]
\definieerkleur [mediumpurple]         [r=0.58,g=0.44,b=0.86]
\definieerkleur [mediumseagreen]       [r=0.24,g=0.70,b=0.44]
\definieerkleur [mediumslateblue]      [r=0.48,g=0.41,b=0.93]
\definieerkleur [mediumspringgreen]    [r=0.00,g=0.98,b=0.60]
\definieerkleur [mediumturquoise]      [r=0.28,g=0.82,b=0.80]
\definieerkleur [mediumvioletred]      [r=0.78,g=0.08,b=0.52]
\definieerkleur [midnightblue]         [r=0.10,g=0.10,b=0.44]
\definieerkleur [mintcream]            [r=0.96,g=1.00,b=0.98]
\definieerkleur [mistyrose]            [r=1.00,g=0.89,b=0.88]
\definieerkleur [moccasin]             [r=1.00,g=0.89,b=0.71]
\definieerkleur [navajowhite]          [r=1.00,g=0.87,b=0.68]
\definieerkleur [navy]                 [r=0.00,g=0.00,b=0.50]
\definieerkleur [navyblue]             [r=0.00,g=0.00,b=0.50]
\definieerkleur [oldlace]              [r=0.99,g=0.96,b=0.90]
```

```
\definieerkleur [olivedrab]           [r=0.42,g=0.56,b=0.14]
\definieerkleur [orange]              [r=1.00,g=0.65,b=0.00]
\definieerkleur [orangered]           [r=1.00,g=0.27,b=0.00]
\definieerkleur [orchid]              [r=0.85,g=0.44,b=0.84]
\definieerkleur [palegoldenrod]       [r=0.93,g=0.91,b=0.67]
\definieerkleur [palegreen]           [r=0.60,g=0.98,b=0.60]
\definieerkleur [paleturquoise]       [r=0.69,g=0.93,b=0.93]
\definieerkleur [palevioletred]       [r=0.86,g=0.44,b=0.58]
\definieerkleur [papayawhip]          [r=1.00,g=0.94,b=0.84]
\definieerkleur [peachpuff]           [r=1.00,g=0.85,b=0.73]
\definieerkleur [peru]                [r=0.80,g=0.52,b=0.25]
\definieerkleur [pink]                [r=1.00,g=0.75,b=0.80]
\definieerkleur [plum]                [r=0.87,g=0.63,b=0.87]
\definieerkleur [powderblue]          [r=0.69,g=0.88,b=0.90]
\definieerkleur [purple]              [r=0.63,g=0.13,b=0.94]
\definieerkleur [red ]                [r=1.00,g=0.00,b=0.00]
\definieerkleur [rosybrown]           [r=0.74,g=0.56,b=0.56]
\definieerkleur [royalblue]           [r=0.25,g=0.41,b=0.88]
\definieerkleur [saddlebrown]         [r=0.55,g=0.27,b=0.07]
\definieerkleur [salmon]              [r=0.98,g=0.50,b=0.45]
\definieerkleur [sandybrown]          [r=0.96,g=0.64,b=0.38]
\definieerkleur [seagreen]            [r=0.18,g=0.55,b=0.34]
\definieerkleur [seashell]            [r=1.00,g=0.96,b=0.93]
\definieerkleur [sienna]              [r=0.63,g=0.32,b=0.18]
\definieerkleur [skyblue]             [r=0.53,g=0.81,b=0.92]
\definieerkleur [slateblue]           [r=0.42,g=0.35,b=0.80]
\definieerkleur [snow]                [r=1.00,g=0.98,b=0.98]
\definieerkleur [springgreen]         [r=0.00,g=1.00,b=0.50]
\definieerkleur [steelblue]           [r=0.27,g=0.51,b=0.71]
```

```
\definieerkleur [tan ]                [r=0.82,g=0.71,b=0.55]
\definieerkleur [thistle]             [r=0.85,g=0.75,b=0.85]
\definieerkleur [tomato]              [r=1.00,g=0.39,b=0.28]
\definieerkleur [turquoise]           [r=0.25,g=0.88,b=0.82]
\definieerkleur [violet]              [r=0.93,g=0.51,b=0.93]
\definieerkleur [violetred]           [r=0.82,g=0.13,b=0.56]
\definieerkleur [wheat]               [r=0.96,g=0.87,b=0.70]
\definieerkleur [white]               [r=1.00,g=1.00,b=1.00]
\definieerkleur [whitesmoke]          [r=0.96,g=0.96,b=0.96]
\definieerkleur [yellow]              [r=1.00,g=1.00,b=0.00]
\definieerkleur [yellowgreen]         [r=0.60,g=0.80,b=0.20]
```

colo-ini
colo-rgb
colo-xwi
colo-mwi
colo-pra

contents register context syst mult supp lang font colo spec core cont m s exit go back

7.4 MS Windows

colo-ini
colo-rgb
colo-xwi
colo-mwi
colo-pra

I cannot imagineanyone using these color, but nevertheless we define them here.

```
\definecolor [DarkRed]      [r=.5,  g=0,   b=0]
\definecolor [DarkGreen]    [r=0,   g=.5,  b=0]
\definecolor [PeaGreen]     [r=.5,  g=.5,  b=0]
\definecolor [DarkBlue]     [r=0,   g=0,   b=.5]
\definecolor [Lavender]     [r=.5,  g=0,   b=.5]
\definecolor [Slate]        [r=0,   g=.5,  b=.5]
\definecolor [LightGrey]    [r=.75, g=.75, b=.75]
\definecolor [DarkGrey]     [r=.5,  g=.5,  b=.5]
\definecolor [BrightRed]    [r=1,   g=0,   b=0]
\definecolor [BrightGreen]  [r=0,   g=1,   b=0]
\definecolor [Yellow]       [r=1,   g=1,   b=0]
\definecolor [BrightBlue]   [r=0,   g=0,   b=1]
\definecolor [Magenta]      [r=1,   g=0,   b=1]
\definecolor [Cyan]         [r=0,   g=1,   b=1]
\definecolor [White]        [r=1,   g=1,   b=1]
\definecolor [Black]        [r=0,   g=0,   b=0]
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

7.5 [to be documented: colo-pra]

This module is not yet fully documented.

colo-ini
colo-rgb
colo-xwi
colo-mwi
colo-pra

contents register context syst mult supp lang font colo spec core cont m s exit go back

8 Special Drivers

8.1 Initialization

Specials are TEX's channel to the outside world. They make TEX even more platform independant and permit easy adaption to new developments. One major drawback of specials is that they have to be supported by printer drivers. We've tried to overcome this problem by implementinmg specials as a sort of drivers themselves.

1
```
\writestatus{loading}{Context Special Macros / Initialization}
```

2
```
\unprotect
```

3
```
\startmessages  dutch  library: specials
  title: specials
      1: -- geladen
      2: verdere nesting is niet toegestaan --
      3: -- gereset
      4: commando -- bestaat niet
      5: definitiefile -- wordt geladen
      6: nesting is niet toegestaan
\stopmessages
```

4
```
\startmessages  english  library: specials
  title: specials
      1: -- loaded
      2: no deeper nesting is permitted --
      3: -- is reset
      4: command -- does not exist
      5: loading definition file --
      6: nesting is not permitted
\stopmessages
```

spec-ini
spec-mis
spec-dvi
spec-ps
spec-yy
spec-tr
spec-tpd
spec-pdf
spec-1p0
spec-1p1
spec-win
spec-etc
spec-mne
spec-htm

contents register context syst mult supp lang font colo spec core cont m s exit go back

5
```
\startmessages  german  library: specials
  title: spezielles
      1: -- geladen
      2: Keine tiefere Verschachtelung erlaubt --
      3: -- ist zurueckgesetzt
      4: Befehl -- existiert nicht
      5: Lade Definitionsdatei --
      6: Verschachtelung nicht erlaubt
\stopmessages
```

6
```
\startmessages  dutch  library: interactions
     21: -- code tussengevoegd
\stopmessages
```

7
```
\startmessages  english  library: interactions
     21: -- code inserted
\stopmessages
```

8
```
\startmessages  german  library: interactions
     21: -- Code eingefuegt
\stopmessages
```

Because there is no standardization in the use of specials, more than one driver or program can be supported. The specials are grouped in libraries. Some of these are general, such as the `postscript` library, some are tuned to a special kind of program, like the `pdf` ones, and some support a specific driver, as we can see in the `yandy` library. A library is build with the commands:

```
   \startspecials[name][inheritance]

   \definespecial\none{...}
   \definespecial\onlyone#1{...}
   \definespecial\alot#1#2#3#4{...}

   \stopspecials
```

Because drivers show some overlap in their support of specials, a mechanism of inheritance is implemented. The predefined libraries show this feature.

Every special has to be predefined first. We do this with the command:

```
   \installspecial [\none]    [and] [0]
   \installspecial [\onlyone] [and] [1]
   \installspecial [\alot]    [or]  [4]
```

This means as much as: there is a special names `\none` which has no arguments and has more than one appearance. The special `\alot` on the other hand has four arguments and is only defined once. Every instance in the libraries of a special of category `and` is executed when called upon, but only one special of category `or` can be active. Most of the `postscript`–specials are of category `or`, because they tend to interfere with driver specific ones. The interactive specials of `dviwindo` and `pdf` are an example of specials that can be called both.

A library is defined in a file with the name `spec-...`. We load a library with the command:

```
   \usespecials [list]
```

where the list can contain one or more file tags, the `...` in the filename. The keyword `reset` resets all loaded specials. This is equivalent to `\resetspecials`.

Although a mechanism of nesting can be implemented, we prefer to use a inheritance mechanism as mentioned. Calls upon `\usespecials` within a `\startspecials` would lead to confusion and errors.

9
```
\newif\ifinheritspecials
```

We define some local constants and variables. They look a bit horrible but we don't want conflicts.

10
```
\def\@@specfil@@{@@specfil@@}
\def\@@speclst@@{@@speclst@@}
\def\@@speccat@@{@@speccat@@}
\def\@@specarg@@{@@specarg@@}
\def\@@specexc@@{@@specexc@@}
```

11
```
\def\currentspecial      {}
\def\currentspecialfile  {}
\def\preloadedspecials   {}
```

\preloadspecials The following command can be used to show the loaded list of specials.

12
```
\def\preloadspecials%
  {\doifsomething{\preloadedspecials}
     {\showmessage{\m!specials}{1}{\preloadedspecials}}}
```

\startspecials Every library has a unique name, which is given as the first argument to `\startspecials`. When another library is defined with the same name, previous specials can be overruled. The name may differ from the file–tag.

The optional second argument can consist of a list of libraries that are to be loaded first. This list can contain file–tags or names of libraries. Names are often more meaningful.

13
```
\def\dostartspecials[#1][#2]%
  {\let\mainspecialfile=\currentspecialfile
   \doifelsenothing{#2}
     {\inheritspecialsfalse}
     {\ifinheritspecials
        \showmessage{\m!specials}{2}{(#2)}%
      \else
        \inheritspecialstrue
```

```
         \processcommalist[#2]\dousespecials
         \inheritspecialsfalse
       \fi}%
    \doifelsenothing{#1}
      {\def\currentspecial{\s!unknown}}
      {\def\currentspecial{#1}}%
    \let\currentspecialfile=\mainspecialfile
    \setevalue{\@@specfil@@\currentspecial}{\currentspecialfile}%
    \unprotect
    \addtocommalist{\currentspecial}\preloadedspecials}
```

14
```
\def\startspecials%
  {\dodoubleempty\dostartspecials}
```

15
```
\def\stopspecials%
  {\def\currentspecial{}%
   \protect}
```

\installspecial
\resetspecials

We have to install specials before we can define and use them. The command itself is defined as a call to another command that executes one or more user–defined specials, depending of it's category: `or` versus `and`.

The command `\installspecial` takes three (non–optional) arguments: the name of the command, the category it belongs to and the number of arguments it takes.

With `\resetspecials` we can unload the predefined specials.

16
```
\def\@@allspecials{}
```

17
```
\def\doinstallspecial[#1][#2][#3]%
  {\setvalue{\@@speclst@@\string#1}{}%
```

```
    \setvalue{\@@speccat@@\string#1}{#2}%
    \setvalue{\@@specarg@@\string#1}{#3}%
    \addtocommalist{\string#1}\@@allspecials
    \def#1{\executespecial#1}}
```

18
```
\def\installspecial%
  {\dotripleargument\doinstallspecial}
```

19
```
\def\resetspecials%
  {\def\docommando##1%
     {\setvalue{\@@speclst@@##1}{}}%
    \processcommacommand[\@@allspecials]\docommando
    \showmessage{\m!specials}{3}{\preloadedspecials}%
    \def\preloadedspecials{}%
    \def\@@allspecials{}}
```

`\definespecial` The command `\definespecial` take the place of `\def` in the definition of a special. Just to be sure, we first check if the command is permitted, i.e. installed. If not, we give a warning and gobble the illegal command in an quite elegant way.

If the command can be combined (`and`) with others, we append it to a list, otherwise (`or`) it becomes the only item in the list.

20
```
\def\definespecial#1%
  {\ifx#1\undefined
     \showmessage{\m!specials}{4}{\string#1}%
     \def\next%
       {\def\@@illegalspecial@@}%
   \else
     \def\next%
       {\doifelse{\getvalue{\@@speccat@@\string#1}}{or}
```

```
          {\edef\@@newspeclst@@{\currentspecial}}
          {\edef\@@newspeclst@@{\getvalue{\@@speclst@@\string#1}}%
           \addtocommalist{\currentspecial}\@@newspeclst@@}%
        \setevalue{\@@speclst@@\string#1}{\@@newspeclst@@}%
        \setvalue{\currentspecial\string#1}}%
   \fi
   \next}
```

\usespecials We use `\usespecials` to load a specific library. This command is only permitted outside de definition part.

21
```
\def\dousespecials#1%
  {\doifelse{#1}{\v!reset}
     {\resetspecials}
     {\doifdefinedelse{\@@specfil@@#1}
        {\edef\currentspecialfile{\getvalue{\@@specfil@@#1}}}
        {\edef\currentspecialfile{#1}}%
      \showmessage{\m!specials}{5}{\f!specialprefix\currentspecialfile}%
      \readsysfile{\f!specialprefix\currentspecialfile}{}{}%
      \showmessage{\m!specials}{1}{\preloadedspecials}}}
```

22
```
\def\usespecials[#1]%
  {\doifelsenothing{\currentspecial}
     {\processcommalist[#1]\dousespecials}
     {\showmessage{\m!specials}{6}{}}}
```

\executespecials

The command **\executespecials** is used to execute the defined specials. Once a special is installed, the special itself calls for this command, so it's not needed outside this module. One can use it if wanted.

A former implementation grouped the execution. Recent additions however —like the specials that implement object handling— asked for non–grouped execution.

23
```
\def\executespecials#1#2%
  {\def\doonespecial##1%
     {\getvalue{##1\string#1}#2\relax}%
   \processcommacommand
     [\getvalue{\@@speclst@@\string#1}]\doonespecial}
```

24
```
\def\executespecial#1%
  {\expandafter\ifcase\getvalue{\@@specarg@@\string#1}\relax
     \def\next%
       {\executespecials#1{}}%
   \or
     \def\next##1%
       {\executespecials#1{{##1}}}%
   \or
     \def\next##1##2%
       {\executespecials#1{{##1}{##2}}}%
   \or
     \def\next##1##2##3%
       {\executespecials#1{{##1}{##2}{##3}}}%
   \or
     \def\next##1##2##3##4%
       {\executespecials#1{{##1}{##2}{##3}{##4}}}%
   \or
```

```
      \def\next##1##2##3##4##5%
        {\executespecials#1{{##1}{##2}{##3}{##4}{##5}}}%
    \or
      \def\next##1##2##3##4##5##6%
        {\executespecials#1{{##1}{##2}{##3}{##4}{##5}{##6}}}%
    \or
      \def\next##1##2##3##4##5##6##7%
        {\executespecials#1{{##1}{##2}{##3}{##4}{##5}{##6}{##7}}}%
    \or
      \def\next##1##2##3##4##5##6##7##8%
        {\executespecials#1{{##1}{##2}{##3}{##4}{##5}{##6}{##7}{##8}}}%
    \or
      \def\next##1##2##3##4##5##6##7##8##9%
        {\executespecials#1{{##1}{##2}{##3}{##4}{##5}{##6}{##7}{##8}{##9}}}%
    \else
      \def\next%
        {\message{illegal special: \string#1}}%
    \fi
    \next}
```

The `{{...}}` are needed because we pass all those arguments to the specials support macro.

25
```
\let\openspecialfile  = \relax
\let\closespecialfile = \relax
```

26
```
\protect
```

The following libraries are defined. Two postscript drivers are supported, as well as two mechanisms for interactive texts.

file	name	calls	support	program / driver
spec-tex	tex		Generic TEX (DVI)	(default)
spec-ps	postscript		Adobe PostScript	(default)
spec-tr	rokicky	postscript	Thomas Rokicky	(dvips)
spec-yy	yandy	postscript	YandY	(dvipsone, dviwindo)
spec-pdf	pdf		Adobe PDF V2.1	(Acrobat)
spec-win	dviwindo	YandY	(dviwindo)	
spec-1p0	pdf		Adobe PDF V 1.0	(Acrobat)
spec-2p0	pdf		Adobe PDF V 2.0	(Acrobat)
spec-htm	html		HTML V 2.0	(dvips)

\dostartgraymode
\dostopgraymode
\dostartrgbcolormode
\dostartcmykcolormode
\dostartgraycolormode
\dostopcolormode

We start with the installation of color and grayscale specials. The values are in the range 0..1 (e.g. 0.25).

```
\dostartgraymode      {gray} ... \dostopgraymode
\dostartrgbcolormode  {red} {green} {blue} ... \dostopcolormode
\dostartcmykcolormode {cyan} {magenta} {yellow} {black} ... \dostopcolormode
\dostartgraycolormode {gray} ... \dostopcolormode
```

Because we can expect conflicts between drivers, we implement them as category `or`. In previous versions of DVIPSONE the use of their color–specials did not interfere with the PostScript ones, but recent versions do.

27
```
\installspecial [\dostartgraymode]      [or] [1]
\installspecial [\dostopgraymode]       [or] [0]
```

28
```
\installspecial [\dostartrgbcolormode]  [or] [3]
\installspecial [\dostartcmykcolormode] [or] [4]
```

```
\installspecial [\dostartgraycolormode] [or] [1]
\installspecial [\dostopcolormode]      [or] [0]
```

\doinsertfile

Probably the most problematic special is the following one. Because we want to be able to support different schemes, we pass a lot of data to it.

```
   \doinsertfile {type,method} {file} {xscale} {yscale} {x} {y} {w} {h} {options}
```

The scale is given percents, the other values are base points.

The special is implemented as `or`. Because DVIPSONE understands them all, a chain of alternatives would generate multiple courrences of the same illustration.

When option 1 is passed, the viewers is asked to present a preview, like the first frame of a movie.

29
```
\installspecial [\doinsertfile] [or] [9]
```

\dostartrotation
\dostoprotation

We support rotation with the special:

```
   \dostartrotation {angle} ... \dostoprotation
```

For the moment these specials are installed as category `or`.

30
```
\installspecial [\dostartrotation] [or] [1]
\installspecial [\dostoprotation]  [or] [0]
```

\doselectfirstpaperbin
\doselectsecondpaperbin

Here are some very printer–specific ones. No further comment.

31
```
\installspecial [\doselectfirstpaperbin]  [or]  [0]
\installspecial [\doselectsecondpaperbin] [or]  [0]
```

\doovalbox When we look at the implementation, this is a complicated one. There are seven arguments.

```
\doovalbox {w} {h} {d} {linewidth} {radius} {stroke} {fill}
```

This command has to return a `\vbox` which can be used to lay over another one (with text). The radius is in degrees, the stroke and fill are `1` (true) of `0` (false).

32
```
\installspecial [\doovalbox] [or] [7]
```

\dosetupidentity We can declare some characteristics of the document with

```
\dosetupidentity {title} {subject} {author} {creator} {date}
```

All data is in string format.

33
```
\installspecial [\dosetupidentity] [and] [5]
```

\dosetuppaper This special can be used to tell the driver what page size to use. The special takes three arguments.

```
\dosetuppaper {type} {width} {height}
```

The type is one of the common identifiers, like A4, A5 or B2.

34
```
\installspecial [\dosetuppaper] [and] [3]
```

\dosetupprinter Some drivers enable the user to specify the paper type used and/or page dimensions to be taken into account.

```
\dosetupprinter {type} {hoffset} {voffset} {width} {height}
```

The first argument is one of `letter`, `legal`, `A4`, `A5` etc. The dimensions are in basepoints.

35
```
\installspecial [\dosetupprinter] [and] [5]
```

\dosetuppage
\dosetupinteraction
\dosetupscreen

Here come some obscure interactive commands. Probably the specs will change with the development of the macros that use them.

The first ones can be used to set up the interaction.

```
\dosetupinteraction
```

Normally this command does nothing but giving a message that some scheme is supported. Postscript prolog files can best be loaded by the printer driver program.

The second one sets up the screen. It takes three arguments:

```
\dosetupscreen {hoffset} {voffset} {width} {height} {options}
```

The first four arguments are in scaled points. Option 1 results in a full screen launch.

36

```
\installspecial [\dosetupinteraction] [and] [0]
\installspecial [\dosetupscreen]      [and] [5]
```

\dostarthide
\dostophide

Not every part of the screen is suitable for paper. Menus for instance have no meaning on an non–interactive medium. These elements are hidden by means of:

```
\dostarthide
\dostophide
```

37

```
\installspecial [\dostarthide]        [or]  [0]
\installspecial [\dostophide]         [or]  [0]
```

\dostartgotolocation
\dostopgotolocation
\dostartgotorealpage
\dostopgotorealpage

The interactive real work is done by the following four specials. The reason for providing the first one with both a label and a number, is a result of the quite poor implementation of `pdfmarks` in version 1.0 of Acrobat. Because only pagenumbers were supported as destination, we had to provide both labels (DVIWINDO) and pagenumbers (PDF). Some drivers use start stop pairs.

```
\dostartgotolocation {w} {h} {file} {label} {page}
\dostartgotorealpage {w} {h} {file} {page}
```

Their counterparts are:

```
\dostopgotolocation
\dostopgotorealpage
```

The internal alternative is used for system–generated links, the external one for user–generated links.

38
```
\installspecial [\dostartgotolocation] [and] [5]
\installspecial [\dostopgotolocation]  [and] [0]
\installspecial [\dostartgotorealpage] [and] [4]
\installspecial [\dostopgotorealpage]  [and] [0]
```

\dostartthisislocation
\dostopthisislocation
\dostartthisisrealpage
\dostopthisisrealpage

The opposite commands of `\dogotosomething` have only one argument:

```
\dostartthisislocation {label}
\dostartthisisrealpage {page}
```

These commands are accompanied by:

```
\dostopthisislocation
\dostopthisisrealpage
```

As with all interactive commands's they are installed as `and` category specials.

39
```
\installspecial [\dostartthisislocation] [and] [1]
\installspecial [\dostopthisislocation]  [and] [0]
\installspecial [\dostartthisisrealpage] [and] [1]
\installspecial [\dostopthisisrealpage]  [and] [0]
```

\dostartexecutecommand
\dostopexecutecommand

The actual behavior of the next pair of commands depends much on the viewing engine. Therefore one cannot depend too much on their support.

```
   \dostartexecutecommand {w} {h} {command} {options}
```

The next commands are supported:

command	action
first	go to the first page
previous	go to the previous page
next	go to the next page
last	go to the last page
backward	go back to the link list
forward	go forward in the link list
print	enter print mode
exit	exit viewer
close	close document
enter	enter viewer
help	show help on the viewer

There are no options yet. Options are to be passed as a comma separated list of assignments.

40
```
\installspecial [\dostartexecutecommand] [and] [4]
\installspecial [\dostopexecutecommand]  [and] [0]
```

\dostartobject
\dostopobject
\doinsertobject

Reuse of object can reduce the output filesize considerably. Reusable objects are implemented with:

```
\dostartobject{name}{width}{height}{depth}
some typeset material
\dostopobject

\doinsertobject{name}
```

The savings can be huge in interactive texts.

41
```
\installspecial [\dostartobject]  [or] [4]
\installspecial [\dostopobject]   [or] [0]
\installspecial [\doinsertobject] [or] [1]
```

\dostartrunprogram
\dostoprunprogram
\dostartgotoprofile
\dostopgotoprofile
\dobeginofprofile
\doendofprofile

These specials are still experimental. They are not yet supported by the programs the way they should be.

— *still undocumented* —

42
```
\installspecial [\dostartrunprogram]    [and] [3]
\installspecial [\dostoprunprogram]     [and] [0]
\installspecial [\dostartgotoprofile]   [and] [3]
\installspecial [\dostopgotoprofile]    [and] [0]
\installspecial [\dobeginofprofile]     [and] [3]
\installspecial [\doendofprofile]       [and] [3]
```

So far for the installation. Finally we preload our favorite set of specials.

43
```
\usespecials[ps,yy,win,pdf]
```

One can overrule this by for instance

```
\usespecials[reset,ps,tr,pdf]
```

8.2 Miscellaneous Macros

Quite some modules in this group are dedicated to supporting PDF directly by means of PDFTEXor indirectly by using Acrobat Distiller. This module implements some common features.

1
```
\writestatus{loading}{Context Special Macros / Miscellaneous Macros}
```

2
```
\unprotect
```

\setPDFdestination

PDF destinations should obey the specifications laid down in the PDF reference manual. The next macro strips illegal characters from the destination name.

3
```
\def\setPDFdestination#1%
  {\bgroup
   \lccode`\/=`-\lccode`\#=`-\lccode`\<=`-\lccode`\>=`-%
   \lccode`\[=`-\lccode`\]=`-\lccode`\(=`-\lccode`\)=`-%
   \lccode`A=`A\lccode`B=`B\lccode`C=`C\lccode`D=`D\lccode`E=`E%
   \lccode`F=`F\lccode`G=`G\lccode`H=`H\lccode`I=`I\lccode`J=`J%
   \lccode`K=`K\lccode`L=`L\lccode`M=`M\lccode`N=`N\lccode`O=`O%
   \lccode`P=`P\lccode`Q=`Q\lccode`R=`R\lccode`S=`S\lccode`T=`T%
   \lccode`U=`U\lccode`V=`V\lccode`W=`W\lccode`X=`X\lccode`Y=`Y\lccode`Z=`Z%
   \stripcharacter{ }\from#1\to\PDFdestination
   \@EA\lowercase\@EA{\@EA\xdef\@EA\PDFdestination\@EA{\PDFdestination}}%
   \egroup}
```

\ifusepagedestinations

In PDF version 1.0 only page references were supported, while in DVIWINDO 1.N only named references were accepted. Therefore CONTEXT supports both methods of referencing. In PDF version 1.1 named destinations arrived. Lack of continuous support of version 1.1 viewers for MS–DOS therefore sometimes forces us to prefer page references. As a bonus, they are faster too and have no limitations. How fortunate we were having both mechanisms available when the version 3.0 (PDF version 1.2) viewers proved to be too bugged to support named destinations.

4
```
\newif\ifusepagedestinations
```

\dodoinsertfile

File insertion depend on the driver or TEX variant used. All driver modules use the same scheme for file insertion, and therefore have the next macro in common:

5
```
\def\dododoinsertfile[#1][#2,#3][#4]%
  {\def\fileinsertionclass{do#1insert}%
   \doifdefinedelse{\fileinsertionclass#2}
     {\def\next{\getvalue{\fileinsertionclass#2}{#4}}}
     {\doifdefinedelse{\fileinsertionclass#3}
        {\def\next{\getvalue{\fileinsertionclass#3}{#4}}}
        {\def\next{\gobblesevenarguments}}}%
   \next}
```

6
```
\def\dodoinsertfile#1#2#3%
  {\dododoinsertfile[#1][#2][#3]}
```

This macro is called with 10 arguments, where the first one specifies the driver, like `yy` or `tr`. The second argument is a `{type,method}` pair and the third the filename.

7
```
\protect
```

\dodoinsertfile ● \setPDFdestination ●

\ifusepagedestinations ●

8.3 Generic TEX Solutions

1
```
\unprotect
```

\dostartobject
\dostopobject
\doinsertobject

Reuse of object is not supported by the DVI format. We therefore just duplicate them using boxes.

```
\startspecials[tex]
```

3
```
\definespecial\dostartobject#1#2#3#4%
  {\setbox\nextbox=\vbox\bgroup
     \def\dodostopobject%
       {\egroup
        \ifx#1\undefined
          \newbox#1\relax
        \fi
        \global\setbox#1=\box\nextbox}}
```

4
```
\definespecial\dostopobject%
  {\dodostopobject}
```

5
```
\definespecial\doinsertobject#1%
  {\copy#1\relax}
```

6
```
\stopspecials
```

7
```
\protect
```

spec-ini
spec-mis
spec-dvi
spec-ps
spec-yy
spec-tr
spec-tpd
spec-pdf
spec-1p0
spec-1p1
spec-win
spec-etc
spec-mne
spec-htm

contents register context syst mult supp lang font colo spec core cont m s exit go back

\doinsertobject ●
\dostartobject ●
\dostopobject ●

8.4 Adobe PostScript

\dostartgraymode
\dostopgraymode
\dostartrgbcolormode
\dostartcmykcolormode
\dostartgraycolormode
\dostopcolormode
\dostartrotation
\dostoprotation
\doselectfirstpaperbin
\doselectsecondpaperbin

This implementation is straightforward and can be used as a default with postscript–drivers. We use ps: as opening, because most drivers support this.

```
\startspecials[postscript]

\def\@@insertpostscriptliteral {ps: }
\def\@@insertpostscriptretain  {" }

\definespecial\dostartgraymode#1%
  {\special
     {\@@insertpostscriptliteral
        #1\space setgray}}

\definespecial\dostopgraymode%
  {\special
     {\@@insertpostscriptliteral
        0 setgray}}
```

5
```
\definespecial\dostartrgbcolormode#1#2#3%
  {\special
     {\@@insertpostscriptliteral
        #1\space #2\space #3\space setrgbcolor}}
```

6
```
\definespecial\dostartcmykcolormode#1#2#3#4%
  {\special
     {\@@insertpostscriptliteral
        #1\space #2\space #3\space #4\space setcmykcolor}}
```

7
```
\definespecial\dostartgraycolormode#1%
  {\special
```

spec-ini
spec-mis
spec-dvi
spec-ps
spec-yy
spec-tr
spec-tpd
spec-pdf
spec-1p0
spec-1p1
spec-win
spec-etc
spec-mne
spec-htm

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
      {\@@insertpostscriptliteral
         #1\space setgray}}
```

8
```
\definespecial\dostopcolormode
  {\special
      {\@@insertpostscriptliteral
         0 setgray}}
```

9
```
\definespecial\doselectfirstpaperbin%
  {\special
      {\@@insertpostscriptliteral
         statusdict begin 1 setpapertray end}}  % checken
```

10
```
\definespecial\doselectsecondpaperbin%
  {\special
      {\@@insertpostscriptliteral
         statusdict begin 0 setpapertray end}}  % checken
```

11
```
\definespecial\dostartrotation#1%
  {\special
      {\@@insertpostscriptliteral
         gsave #1\space rotate}}
```

12
```
\definespecial\dostoprotation%
  {\special
      {\@@insertpostscriptliteral
         currentfont grestore setfont}}
```

spec-ini
spec-mis
spec-dvi
spec-ps
spec-yy
spec-tr
spec-tpd
spec-pdf
spec-1p0
spec-1p1
spec-win
spec-etc
spec-mne
spec-htm

contents register context syst mult supp lang font colo spec core cont m s exit go back

\doovalbox This implementation is a mixture of several possible implementations. We use some constants that may not be changed. It took some time to find them, but these values offer quite accurate results. The macro calls for `\forgetall`, which resets indentation, skips and `\everypar`.

Because a stroke follows the line, we correct for half of the linewidth. Furthermore we use scaling to overcome some limitations in the precision (< 1 sp) and to prevent rounding errors. We also do some correction for large values. We let PostScript compare some arguments with `a b eq {action} fi`.

The path is based on a macro of J. Hefferon cs. We also tried the D. Salomon implementation, but this did not work well, just like some other alternatives.

13
```
\def\@@insertpostscriptliteral {ps: }
\def\@@insertpostscriptretain  {postscript } % unknown
```

14
```
\def\dosomeovalcalc#1#2#3%
  {\dimen2=#1sp%
   \advance\dimen2 by #2%
   \ScaledPointsToBigPoints{\number\dimen2}#3}
```

15
```
\definespecial\doovalbox#1#2#3#4#5#6#7%
  {\bgroup
   \dimen0=#4sp\divide\dimen0 by 2
   \dosomeovalcalc{0}{+\dimen0}\xmin
   \dosomeovalcalc{#1}{-\dimen0}\xmax
   \dosomeovalcalc{#2}{-\dimen0}\ymax
   \dosomeovalcalc{#3}{+\dimen0}\ymin
   \dosomeovalcalc{#4}{0pt}\stroke
   \dosomeovalcalc{#5}{0pt}\radius
   \edef\dostroke{#6}%
   \edef\dofill{#7}%
   \vbox
```

```
     \bgroup
     \offinterlineskip
     \forgetall
     \hsize\!!zeropoint
     \vrule\!!width\!!zeropoint\!!height#2sp\!!depth#3sp\relax
     \special
       {\@@insertpostscriptretain
          gsave
            newpath
            \xmin\space \radius\space add \ymin\space moveto
            \xmax\space \ymin\space \xmax\space \ymax\space \radius\space arcto
            \xmax\space \ymax\space \xmin\space \ymax\space \radius\space arcto
            \xmin\space \ymax\space \xmin\space \ymin\space \radius\space arcto
            \xmin\space \ymin\space \xmax\space \ymin\space \radius\space arcto
            \xmin\space \radius\space add \ymin\space moveto
            16 {pop} repeat
            closepath
            (\dostroke) (1) eq
              {\stroke\space 0 ne
                {gsave
                   \stroke\space setlinewidth
                    stroke
                 grestore} if} if
            (\dofill) (1) eq
              {fill} if
          grestore}%
     \egroup
   \egroup}

16 \stopspecials
```

spec-ini
spec-mis
spec-dvi
spec-ps
spec-yy
spec-tr
spec-tpd
spec-pdf
spec-1p0
spec-1p1
spec-win
spec-etc
spec-mne
spec-htm

contents register context syst mult supp lang font colo spec core cont m s exit go back

8.5 Y&Y's DVIPSONE and DVIWINDO

\dostartgraymode
\dostopgraymode
\dostartrgbcolormode
\dostartcmykcolormode
\dostartgraycolormode
\dostopcolormode
\doinsertfile

We implement a nice and simple figure–insertion special and make use of Y&Y's color specials. Otherwise DVIWINDO would not show colors.

```
\startspecials[yandy]   % [postscript]

\def\@@insertpostscriptliteral {ps: }
\def\@@insertpostscriptretain  {" }

\definespecial\dostartgraymode#1%
  {\special{color gray #1}}

\definespecial\dostopgraymode%
  {\special{color gray 0}}

\definespecial\dostartrgbcolormode#1#2#3%
  {\special{color rgb #1 #2 #3}}

\definespecial\dostartcmykcolormode#1#2#3#4%
  {\special{color cmyk #1 #2 #3 #4}}

\definespecial\dostartgraycolormode#1%
  {\special{color gray #1}}

\definespecial\dostopcolormode%
  {\special{color gray 0}}

\def\doyyinserteps#1#2#3#4#5#6#7#8%  equals rockiky
  {\ScaledPointsToBigPoints{#4}\width
   \ScaledPointsToBigPoints{#5}\height
   \special
```

```
     {psfile=#1
        hscale=#2\space
        vscale=#3\space
        hoffset=\width \space
        voffset=\height}}
```

10
```
\def\doyyinserttif#1#2#3#4#5#6#7#8%
  {\special{insertimage: #1 #6 #7}}
```

11
```
\definespecial\doinsertfile#1#2#3#4#5#6#7#8#9%
  {\bgroup
   \dodoinsertfile{yy}{#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}%
   \egroup}
```

12
```
\stopspecials
```

8.6 Thomas Rokicky's DVIPS

\doinsertfile

We overrule the figure–insertion special. Things should be more accurate, but maybe someday . . .

1
```
\startspecials[rokicky]  % [postscript]
```

2
```
\def\@@insertpostscriptliteral {ps: }
\def\@@insertpostscriptretain  {" }
```

3
```
\def\dotrinserteps#1#2#3#4#5#6#7#8%
  {\ScaledPointsToBigPoints{#4}\width
   \ScaledPointsToBigPoints{#5}\height
   \special
     {psfile=#1
        hscale=#2\space
        vscale=#3\space
        hoffset=\width \space
        voffset=\height}}
```

4
```
\definespecial\doinsertfile#1#2#3#4#5#6#7#8#9%
  {\bgroup
   \dodoinsertfile{tr}{#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}%
   \egroup}
```

5
```
\stopspecials
```

`\doinsertfile` ●

8.7 PDFTEX

Being one of the first typographical systems able to support advances PDF support, TEX is also one of the first systems to produce high quality PDF code directly. Thanks to Han The Thanh c.s. the TEX community can leap forward once again.

One important characteristic of PDFTEX is that is can produce standard DVI code as well as PDF code. This enables us to use one format file to support both output formats.

All modules in this group use specials to tell drivers what non–TEX actions to take. Because from the TEX point of view, there is no difference between DVI and PDF, we therefore only have to bend the DVI driver support into PDF support. Technically spoken, specials no longer serve a purpose, except from ending up as comment in the PDF file. The core primitive in this module therefore is the PDFTEX primitive `\pdfliteral`.

Before we continue we need to make sure if indeed those PDFTEX primitives are permitted. If no primitives are available, we just stop reading any further.

1
```
\ifx\pdfoutput\undefined \endinput \else \unprotect \fi
```

Once we are sure that we're indeed supporting PDFTEX, we force PDF output at the highest compression. For debugging purposes one can set the compresslevel to 0.

2
```
\pdfoutput       =1
\pdfcompresslevel=9
```

Now we have to make sure no other specials are supported, else PDFTEX will keep on telling us that we're wrong.

3
```
\usespecials[reset]
```

Just in case we mimmick specials, we have to make sure no default specials end up in the process.

spec-ini
spec-mis
spec-dvi
spec-ps
spec-yy
spec-tr
spec-tpd
spec-pdf
spec-1p0
spec-1p1
spec-win
spec-etc
spec-mne
spec-htm

4
```
\let\defaultspecial=\gobbleoneargument
```

Having reset all the special support, we have to define all needed and possible support in this module.

5
```
\startspecials[tpd]
```

This means that by saying

```
\usespecials[tpd]
```

we get ourselves PDF output.

\dosetuppaper If we don't set the paper size, PDFTEX will certainly do not the way we want, therefore we need:

6
```
\definespecial\dosetuppaper#1#2#3%
  {\global\pdfpagewidth =#2\relax
   \global\pdfpageheight=#3\relax}
```

\doinsertfile Graphics are not part of TEX and therefore not part of the DVI standard. PDF on the other hand has several graphic primitives. During the multi–step process TEX $\longrightarrow$ DVI $\longrightarrow$ POSTSCRIPT $\longrightarrow$ PDF one can insert graphics using specials. In PDFTEX however there is only one step! This means that PDFTEX itself has to do the inclusion.

At the moment PDFTEX supports inclusion of bitmap PNG graphics as well as not too complicated PDF code. Using this last option, we are able to include both METAPOST and PDF output produced by GHOSTVIEW.

We fall back on the generic CONTEXT module supp-pdf to accomplish PDF inclusion. The methods implemented there are hooked into both the figure placement mechanisms of CONTEXT and the specials inclusion mechanism.

7
```
\definespecial\doinsertfile#1#2#3#4#5#6#7#8#9%
  {\bgroup
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
   \dodoinsertfile{tpd}{#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}%
   \egroup}
```

The three methods supported for the moment are `mps` for METAPOST graphics, `pdf` for GHOSTVIEW PDF code, and `png` for bitmap graphics.

8
```
\def\dotpdinsertmps#1#2#3#4#5#6#7#8%
  {\scratchdimen=#2pt \PointsToReal{.01\scratchdimen}\xscale
   \scratchdimen=#3pt \PointsToReal{.01\scratchdimen}\yscale
   \convertMPtoPDF{#1}\xscale\yscale}
```

9
```
\def\dotpdinsertpdf#1#2#3#4#5#6#7#8%
  {\beforesplitstring#1\at.\to\filename
   \scratchdimen=#2pt \PointsToReal{.01\scratchdimen}\xscale
   \scratchdimen=#3pt \PointsToReal{.01\scratchdimen}\yscale
   \convertPDFtoPDF{\filename.pdf}\xscale\yscale{#4sp}{#5sp}{#6sp}{#7sp}}
```

10
```
\def\dotpdinsertpng#1#2#3#4#5#6#7#8%
  {\pdfimage width #6sp height #7sp #1\relax}
```

PDF supports the inclusion of video movies. In CONTEXT we support these in a way similar to figure inclusion.

11
```
\def\dotpdinsertmov#1#2#3#4#5#6#7#8%
  {\ScaledPointsToBigPoints{#6}\width
   \ScaledPointsToBigPoints{#7}\height
   \edef\pdf@@posterize{\ifcase#8 \or/Poster true\fi}%
   \pdfannotlink
     width #6sp
     height #7sp
     attr {/Border [0 0 0]}
     user {/Subtype /Movie
           /Movie <</F (#1) /Aspect [\width\space \height\space] \pdf@@posterize>>
           /A <</ShowControls false>>}
   \pdfendlink}
```

\doovalbox

For drawing ovals we use quite raw PDF code. The next implementation does not differ that much from the one implemented in the POSTSCRIPT driver.

12
```
\def\dosomeovalcalc#1#2#3%
  {\dimen2=#1sp
   \advance\dimen2 by #2\relax
   \ScaledPointsToBigPoints{\number\dimen2}#3}
```

13
```
\definespecial\doovalbox#1#2#3#4#5#6#7%
  {\bgroup
   \dimen0=#4sp\divide\dimen0 by 2
   \dosomeovalcalc{0} {+\dimen0}\xmin
   \dosomeovalcalc{#1}{-\dimen0}\xmax
   \dosomeovalcalc{#2}{-\dimen0}\ymax
   \dosomeovalcalc{#3}{+\dimen0}\ymin
   \advance\dimen0 by #5sp
   \dosomeovalcalc{0} {+\dimen0}\xxmin
   \dosomeovalcalc{#1}{-\dimen0}\xxmax
   \dosomeovalcalc{#2}{-\dimen0}\yymax
   \dosomeovalcalc{#3}{+\dimen0}\yymin
   \dosomeovalcalc{#4}{0pt}\stroke
   \dosomeovalcalc{#5}{0pt}\radius
   \edef\dostroke{#6}%
   \edef\dofill{#7}%
   \vbox
   \bgroup
   \offinterlineskip
   \forgetall
   \hsize\!!zeropoint
   \vrule\!!width\!!zeropoint\!!height#2sp\!!depth#3sp\relax
   \pdfliteral
     {q
      \stroke\space w
      \xxmin\space \ymin\space  m
      \xxmax\space \ymin\space  l
      \xmax\space  \ymin\space  \xmax\space  \yymin\space y
      \xmax\space  \yymax\space l
      \xmax\space  \ymax\space  \xxmax\space \ymax\space  y
      \xxmin\space \ymax\space  l
      \xmin\space  \ymax\space  \xmin\space  \yymax\space y
      \xmin\space  \yymin\space l
      \xmin\space  \ymin\space  \xxmin\space \ymin\space  y
      \ifnum\dostroke=1 S \fi
      \ifnum\dofill  =1 f \fi
      Q}%
   \egroup
 \egroup}
```

\dostartgraymode
\dostopgraymode
\dostartrgbcolormode
\dostartcmykcolormode
\dostartgraycolormode
\dostopcolormode

In PDF there are two color states, one for strokes and one for fills. This means that we have to set the color in a rather redundant looking way. Unfortunately this makes the PDF file much larger than needed.

```
\definespecial\dostartgraymode#1%
  {\pdfliteral{#1 g #1 G}}

\definespecial\dostopgraymode%
  {\pdfliteral{0 g 0 G}}

\definespecial\dostartrgbcolormode#1#2#3%
  {\pdfliteral{#1 #2 #3 rg #1 #2 #3 RG}}

\definespecial\dostartcmykcolormode#1#2#3#4%
  {\pdfliteral{#1 #2 #3 #4 k #1 #2 #3 #4 K}}
```

17

```
\definespecial\dostartgraycolormode#1%
  {\pdfliteral{#1 g #1 G}}
```

18

```
\definespecial\dostopcolormode%
  {\pdfliteral{0 g 0 G}}
```

19

\dostartrotation
\dostoprotation

Rotating some text can be accomplished by setting the first four elements of the transform matrix. We only support some fixed angles. The q's take care of grouping.

20

```
\definespecial\dostartrotation#1%
  {\processaction
     [#1]
     [ 90=>\pdfliteral{q  0  1 -1  0 0 0 cm},
      180=>\pdfliteral{q -1  0  0 -1 0 0 cm},
      270=>\pdfliteral{q  0 -1  1  0 0 0 cm},
      360=>\pdfliteral{q  1  0  0  1 0 0 cm}]}
```

21
```
\definespecial\dostoprotation%
  {\pdfliteral{Q}}
```

\dosetupinteraction

Nothing special is needed to enable PDF commands and interaction. We stick with a message.

22
```
\definespecial\dosetupinteraction%
  {\showmessage{\m!interactions}{21}{pdftex}}
```

\dostartthisisrealpage
\dostartthisislocation..
\dostartgotolocation

The interactions macros are the core of this module. We support both page destinations and named ones.

For the moment we use object number (that is, behind the screens, the user uses his own numbers) destinations instead of page ones. The latter works, but not 100%.

23
```
\definespecial\dostartthisisrealpage#1%
  {\pdfdest num #1 fit}  % will be {} when page is ok
```

24
```
\definespecial\dostartthisislocation#1%
  {\ifusepagedestinations \else
     \setPDFdestination{#1}%
     \doifsomething{\PDFdestination}
       {\pdfdest name {\PDFdestination} fit}%
   \fi}
```

When going to a location, we obey the time and space saving boolean`\ifusepagedestination`. Names destinations are stripped and made robust.

25
```
\definespecial\dostartgotolocation#1#2#3#4#5%
  {\bgroup
   \doifelsenothing{#3}
     {\!!doneafalse}
     {\doifparentfileelse{#3}
```

spec-ini
spec-mis
spec-dvi
spec-ps
spec-yy
spec-tr
spec-tpd
spec-pdf
spec-1p0
spec-1p1
spec-win
spec-etc
spec-mne
spec-htm

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
      {\!!doneafalse}
      {\!!doneatrue}}%
  \ifusepagedestinations
    \if!!donea \else
      \scratchcounter=0#5\relax
      \edef\PDFdestination{\the\scratchcounter}%
      \pdfannotlink
        width #1sp
        height #2sp
        depth 0pt
        attr{/Border [0 0 0]}
      % goto \if!!donea file {#3.pdf} \fi page \PDFdestination\space {/Fit}
        goto \if!!donea file {#3.pdf} \fi num \PDFdestination\space
      \pdfendlink
    \fi
  \else
    \setPDFdestination{#4}%
    \doifsomething{\PDFdestination}
      {\pdfannotlink
         width #1sp
         height #2sp
         depth 0pt
         attr{/Border [0 0 0]}
         goto \if!!donea file {#3.pdf} \fi name {\PDFdestination}%
       \pdfendlink}%
  \fi
  \egroup}
```

26
```
\definespecial\dostartgotorealpage#1#2#3#4%
  {\bgroup
```

```
    \doifelsenothing{#3}
      {\!!doneafalse}
      {\doifparentfileelse{#3}
         {\!!doneafalse}
         {\!!doneatrue}}%
    \if!!donea \else
       \scratchcounter=0#4\relax
       \edef\PDFdestination{\the\scratchcounter}%
       \pdfannotlink
         width #1sp
         height #2sp
         depth 0pt
         attr{/Border [0 0 0]}
       % goto \if!!donea file {#3.pdf} \fi page \PDFdestination\space {/Fit}
         goto \if!!donea file {#3.pdf} \fi num \PDFdestination\space
         \pdfendlink
    \fi
    \egroup}
```

\dostarthide
\dostophide

Hiding parts of the document for printing is not yet supported by PDF and therefore PDFTEX.

27
```
\definespecial\dostarthide%
  {}
```

28
```
\definespecial\dostophide%
  {}
```

\dosetupscreen

Setting of the screen boundingbox involves some calculations. Here we also take care of (non) full screen startup. The dimensions are rounded.

29

```
\definespecial\dosetupscreen#1#2#3#4#5%
  {\bgroup
   \!!widtha=#3sp
   \advance\!!widtha by #1sp
   \!!heighta=-#4sp
   \!!heightb=\pdfpageheight
   \advance\!!heightb by -#2sp
   \advance\!!heighta by \!!heightb
   \ScaledPointsToWholeBigPoints{#1}\left
   \ScaledPointsToWholeBigPoints{\number\!!heighta}\bottom
   \ScaledPointsToWholeBigPoints{\number\!!widtha}\width
   \ScaledPointsToWholeBigPoints{\number\!!heightb}\height
   \expanded{\global\noexpand\pdfpagesattr=
     {/CropBox [\left\space\bottom\space\width\space\height]}}%
   \ifcase#5\else
     \pdfcatalog pagemode {/FullScreen}\relax
   \fi
   \egroup}
```

\dostartexecutecommand

PDF viewers enable us to navigate using menus and shortcut keys. These navigational tools can also be accessed by using annotations. The next special takes care of inserting them.

30

```
\definespecial\dostartexecutecommand#1#2#3#4%
  {\bgroup
   \processaction
     [#3]
     [      first=>\def\command{First},
```

spec-ini
spec-mis
spec-dvi
spec-ps
spec-yy
spec-tr
spec-tpd
spec-pdf
spec-1p0
spec-1p1
spec-win
spec-etc
spec-mne
spec-htm

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
          previous=>\def\command{Prev},
              next=>\def\command{Next},
              last=>\def\command{Last},
          backward=>\def\command{GoBack},
           forward=>\def\command{GoForward},
             print=>\def\command{Print},
              exit=>\def\command{Quit},
             close=>\def\command{Close},
              help=>\def\command{HelpUserGuide},
              swap=>\def\command{FullScreen},
        \s!unknown=>\let\command=\s!unknown]%
     \pdfannotlink
       width #1sp
       height #2sp
       depth 0pt
       attr {/Border [0 0 0]}
       user {/S /Named /N /\command}
     \pdfendlink
     \egroup}
```

\dosetupidentity Documents can be tagged with an application accessible title and subtitle, the authorname, a date, the creator, keywords etc. For the moment PDFTEX only supports the first three of these.

31
```
\definespecial\dosetupidentity#1#2#3#4#5%
  {\pdfinfo title {#1} subject {#2} author {#3}\relax} % creator {#4}
```

\dostartrunprogam

Although possible, running applications is not yet implemented here.

32
```
\definespecial\dostartrunprogram#1#2#3%
  {}
```

\dostartgotoprofile
\dostopgotoprofile
\dobeginofprofile
\doendofprofile

CONTEXT user profiles and version control fall back on PDF article threads. Unfortunately one cannot influence the view yet in an (for me) acceptable way.

33
```
\definespecial\dostartgotoprofile#1#2#3%
  {\pdfannotlink
     width #1sp
     height #2sp
     depth 0pt
     attr{/Border [0 0 0]}
     thread name {#3}%
   \pdfendlink}
```

34
```
\definespecial\dobeginofprofile#1#2#3%
  {\doifsomething{#1}
     {\pdfthread name {#1}}}
```

35
```
\definespecial\doendofprofile#1#2#3%
  {\pdfendthread}
```

\dostartobject
\dostopobject
\doinsertobject

Due to PDF's object oriented character, we can include and reuse objects. These can be compared with TEX's boxes. The TEX counterpart is defined in the module `spec-dvi`. We don't use the dimensions here.

36
```
\definespecial\dostartobject#1#2#3#4%
  {\setbox\nextbox=\vbox\bgroup
     \def\dodostopobject%
```

```
          {\egroup
           \pdfsetform\nextbox
           \scratchcounter=\pdflastform
           \xdef#1{\the\scratchcounter}}}
```

37
```
\definespecial\dostopobject%
  {\dodostopobject}
```

38
```
\definespecial\doinsertobject#1%
  {\pdfrefform#1\relax}
```

39
```
\stopspecials
```

40
```
\protect \endinput
```

spec-ini
spec-mis
spec-dvi
spec-ps
spec-yy
spec-tr
spec-tpd
spec-pdf
spec-1p0
spec-1p1
spec-win
spec-etc
spec-mne
spec-htm

contents register context syst mult supp lang font colo spec core cont m s exit go back

\dobeginofprofile ●
\doendofprofile ●
\doinsertfile ●
\doinsertobject ●
\doovalbox ●
\dosetupidentity ●
\dosetupinteraction ●
\dosetuppaper ●
\dosetupscreen ●
\dostartcmykcolormode ●
\dostartexecutecommand ●
\dostartgotolocation ●
\dostartgotoprofile ●
\dostartgraycolormode ●
\dostartgraymode ●
\dostarthide ●
\dostartobject ●
\dostartrgbcolormode ●
\dostartrotation ●
\dostartrunprogam ●
\dostartthisislocation
dostartgotorealpage ●
\dostartthisisrealpage ●
\dostopcolormode ●
\dostopgotoprofile ●
\dostopgraymode ●
\dostophide ●
\dostopobject ●
\dostoprotation ●

spec-ini
spec-mis
spec-dvi
spec-ps
spec-yy
spec-tr
spec-tpd
spec-pdf
spec-1p0
spec-1p1
spec-win
spec-etc
spec-mne
spec-htm

8.8 Adobe PDF version 1.2

1

```
\unprotect
```

These specials are not as beautiful as they should be. The main reason for this is that we started with DVIWINDO, which lacks support of EPS–insertions, but offered a powerfull linking mechanism. The first version of PDF did not support labels but only pagenumbers. This dreadfull omission was corrected in version 2.0, but we continue to support both alternatives. One never knows.

Although the concepts behind the **pdfmark**'s are still far from perfect, version 2.1 brought another change. This time the format was changed. So much for upward compatibility.

2

```
\startspecials[pdf]
```

Instead of a prolog, we can put the code in the file ourselve.

```
   \definespecial\dosetupinteraction%
     {\special
        {\@@insertpostscriptliteral
           /pdfmark where
             {pop}
             {userdict /pdfmark /cleartomark load put}
           ifelse}}
```

We decided to use a prolog file. The following code has to be put somewhere. To overcome problems, we always embed the fonts, but copyrights force us always to make subsets.

```
   /currentdistillerparams where
     { pop } { userdict /currentdistillerparams { 1 dict } put } ifelse

   /setdistillerparams     where
```

```
    { pop } { userdict /setdistillerparams { pop } put } ifelse

  << /AntiAliasColorImages   true
     /AntiAliasGrayImages    true
     /AntiAliasMonoImages    true
     /ConvertCMYKImagesToRGB true
     /MaxSubsetPct           99
     /EmbedAllFonts          true
     /SubSetFonts            true >> setdistillerparams
```

Beware, this is the PostScript Level 2 way of doing things.

3
```
\definespecial\dosetupinteraction%
  {\showmessage{\m!interactions}{21}{acrobat}}
```

4
```
\definespecial\dostartthisislocation#1%
  {\ifusepagedestinations \else
     \setPDFdestination{#1}%
     \doifsomething{\PDFdestination}
       {\special
          {\@@insertpostscriptretain
             [/Dest /\PDFdestination\space % (\PDFdestination)
              /View [/Fit]
              /DEST
             pdfmark}}%
   \fi}
```

5
```
\definespecial\dostartgotolocation#1#2#3#4#5%
  {\bgroup
   \ScaledPointsToBigPoints{#1}\width
   \ScaledPointsToBigPoints{#2}\height
```

```
\doifelsenothing{#3}
  {\!!doneafalse}
  {\doifparentfileelse{#3}
     {\!!doneafalse}
     {\!!doneatrue}}%
\ifusepagedestinations
  \doifnot{0#5}{0}
    {\special
       {\@@insertpostscriptretain
          [\if!!donea
             /Action /GoToR
             /File (#3.pdf)
           \else
             /Action /GoTo
           \fi
           /Rect [0 0 \width\space \height]
           /Border [0 0 0]
           /Page #5
           /View [/Fit]
           /Subtype /Link
           /ANN
          pdfmark}}%
\else
  \setPDFdestination{#4}%
  \doifsomething{\PDFdestination}
    {\special
       {\@@insertpostscriptretain
          [\if!!donea
             /Action /GoToR
```

spec-ini
spec-mis
spec-dvi
spec-ps
spec-yy
spec-tr
spec-tpd
spec-pdf
spec-1p0
spec-1p1
spec-win
spec-etc
spec-mne
spec-htm

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
                    /File (#3.pdf)
                  \else
                    /Action /GoTo
                  \fi
                  /Rect [0 0 \width\space \height]
                  /Border [0 0 0]
                  /Dest /\PDFdestination\space % (\PDFdestination)
                  /Subtype /Link
                  /ANN
                pdfmark}}%
    \fi
    \egroup}
```

6
```
\definespecial\dostartgotorealpage#1#2#3#4%
  {\bgroup
   \ScaledPointsToBigPoints{#1}\width
   \ScaledPointsToBigPoints{#2}\height
   \doifelsenothing{#3}
     {\!!doneafalse}
     {\doifparentfileelse{#3}
        {\!!doneafalse}
        {\!!doneatrue}}%
   \doifnot{0#4}{0}
     {\special
        {\@@insertpostscriptretain
           [\if!!donea
              /Action /GoToR
              /File (#3.pdf)
            \else
              /Action /GoTo
            \fi
            /Rect [0 0 \width\space \height]
            /Border [0 0 0]
            /View [/Fit]
            /Page #4
            /Subtype /Link
            /ANN
           pdfmark}}%
   \egroup}
```

7
```
\definespecial\dostartexecutecommand#1#2#3#4%
  {\bgroup
   \ScaledPointsToBigPoints{#1}\width
   \ScaledPointsToBigPoints{#2}\height
   \processaction
     [#3]
     [     first=>\def\command{First},
        previous=>\def\command{Prev},
            next=>\def\command{Next},
            last=>\def\command{Last},
        backward=>\def\command{GoBack},
         forward=>\def\command{GoForward},
           print=>\def\command{Print},
            exit=>\def\command{Quit},
           close=>\def\command{Close},
            help=>\def\command{HelpUserGuide},
            swap=>\def\command{FullScreen},
      \s!unknown=>\let\command=\s!unknown]%
    \special
      {\@@insertpostscriptretain
```

```
          [/Action <</Subtype /Named /N /\command>>
           /Rect [0 0 \width\space \height]
           /Border [0 0 0]
           /Subtype /Link
           /ANN
          pdfmark}%
    \egroup}
```

8
```
\definespecial\dostopexecutecommand%
  {}
```

9
```
\edef\@@psbgroup{\string{}
\edef\@@psegroup{\string}}
```

10
```
\definespecial\dostarthide%
  {\special
     {\@@insertpostscriptretain
          [/DataSource (false \@@psbgroup)
           /PS
          pdfmark}}
```

11
```
\definespecial\dostophide%
  {\special
     {\@@insertpostscriptretain
         [/DataSource (\@@psegroup if)
          /PS
         pdfmark}}
```

12
```
\definespecial\dosetupscreen#1#2#3#4#5%
  {\bgroup
   \!!widtha=#3sp
```

```
    \advance\!!widtha by #1sp
    \!!heighta=-#4sp
    \!!heightb=\printpapierhoogte
    \advance\!!heightb by -#2sp
    \advance\!!heighta by \!!heightb
    \ScaledPointsToBigPoints{#1}\left
    \ScaledPointsToBigPoints{\number\!!heighta}\bottom
    \ScaledPointsToBigPoints{\number\!!widtha}\width
    \ScaledPointsToBigPoints{\number\!!heightb}\height
\ScaledPointsToWholeBigPoints{#1}\left
\ScaledPointsToWholeBigPoints{\number\!!heighta}\bottom
\ScaledPointsToWholeBigPoints{\number\!!widtha}\width
\ScaledPointsToWholeBigPoints{\number\!!heightb}\height
    \edef\pdf@@screenmode{\ifcase#5/UseNone\else/FullScreen\fi}%
    \special
      {\@@insertpostscriptretain
          [/CropBox [\left\space\bottom\space\width\space\height]
           /PAGES
          pdfmark}%
    \special
      {\@@insertpostscriptretain
          [/PageMode \pdf@@screenmode\space
           /Page 1
           /View [/Fit]
           /ViewerPreferences
              << /PageLayout /SinglePage
                 /NonFullScreenPageMode /UseNone >>
           /DOCVIEW
          pdfmark}%
```

```
   \egroup}
```

13
```
\definespecial\dosetupidentity#1#2#3#4#5%
  {\special
     {\@@insertpostscriptretain
        [/Title (#1)
        /Subject (#2)
        /Author (#3)
        /Creator (#4)
        /ModificationDate (#5)
        /DOCINFO
       pdfmark}}
```

14
```
\definespecial\dostartrunprogram#1#2#3%
  {\bgroup
   \ScaledPointsToBigPoints{#1}\width
   \ScaledPointsToBigPoints{#2}\height
   \special
     {\@@insertpostscriptretain
        [/Action /Launch
         /File (#3)
         /Rect [0 0 \width\space \height]
         /Border [0 0 0]
         /Subtype /Link
         /ANN
        pdfmark}%
   \egroup}
```

15
```
\definespecial\dostartgotoprofile#1#2#3%
  {\bgroup
```

```
    \ScaledPointsToBigPoints{#1}\width
    \ScaledPointsToBigPoints{#2}\height
    \doifsomething{#3}
      {\special
         {\@@insertpostscriptretain
            [/Action /Article
             /Dest (#3)
             /Rect [0 0 \width\space \height]
             /Border [0 0 0]
             /View [/Fit]
             /Subtype /Link
             /ANN
            pdfmark}}%
    \egroup}
```

16
```
\definespecial\dobeginofprofile#1#2#3% label width page
  {\bgroup
    \doifelsenothing{#1}
      {\!!doneatrue}
      {\!!doneafalse}%
    \doifnot{0#3}{0}
      {\special
         {\@@insertpostscriptretain
            [/Title (#1)
             /Rect [0 0 0 0]
             \if!!donea /Page #3 \fi
             /ARTICLE
            pdfmark}}%
    \egroup}
```

17
```
\definespecial\doendofprofile#1#2#3%
  {}
```

18
```
\def\docommoninsertmov#1#2#3#4#5#6#7#8%
  {\bgroup
   \ScaledPointsToBigPoints{#6}\width
   \ScaledPointsToBigPoints{#7}\height
   \edef\pdf@@posterize{\ifcase#8 \or/Poster true\fi}%
   \special
     {\@@insertpostscriptretain
        [/Type /Annot
         /Subtype /Movie
         /Rect [0 0 \width\space \height]
         /Movie <</F (#1) /Aspect [\width\space \height] \pdf@@posterize>>
         /A <</ShowControls false>>
         /ANN
        pdfmark}%
   \egroup}
```

19
```
\let\doyandyinsertmov = \docommoninsertmov
\let\dotrinsertmov    = \docommoninsertmov
```

20
```
\newbox\pdfobjects
```

21
```
\definespecial\dostartobject#1#2#3#4%
  {\setbox\nextbox=\hbox\bgroup
     \bgroup
     \ScaledPointsToBigPoints{#2}\width
     \ScaledPointsToBigPoints{#3}\height
     \ScaledPointsToBigPoints{#4}\depth
     \escapechar=-1
```

```
    \special
      {\@@insertpostscriptretain
         [/BBox [0 -\depth\space \width\space \height]
          /_objdef {object:\string#1}
          /BP pdfmark}%
    \egroup}
```

22
```
\definespecial\dostopobject%
  {\special
     {\@@insertpostscriptretain
        [/EP pdfmark}%
   \egroup
   \smashbox\nextbox
   \global\setbox\pdfobjects=\hbox{\box\pdfobjects\box\nextbox}}
```

23
```
\definespecial\doinsertobject#1%
  {\hbox\bgroup
    \box\pdfobjects
    \escapechar=-1
    \special
      {\@@insertpostscriptretain
         [{object:\string#1} /SP pdfmark}%
   \egroup}
```

24
```
\stopspecials
```

25
```
\protect
```

```
\dobeginofprofile        ●
\doendofprofile          ●
\doinsertobject          ●
\dosetupidentity         ●
\dosetupinteraction      ●
\dosetupscreen           ●
\dostartcommand          ●
\dostartgotolocation     ●
\dostartgotoprofile      ●
\dostartgotorealpage     ●
\dostartobject           ●
\dostartrunprogram       ●
\dostartthisislocation   ●
\dostopobject            ●

\usepagedestinations     ●
```

8.9 Adobe PDF version 1.0

\dostartgotolocation
\dostartgotorealpage
\dostartthisislocation

Due to an incompatible update of the PDF–format and the lack of version 2 viewers for MS–DOS, we still can support the version 1 specials. Because we use the same library–name, the definitions below overrule previous defined ones.

1
```
\startspecials[pdf]
```

2
```
\definespecial\dostartthisislocation#1%
  {}
```

3
```
\def\pdfgoto#1#2#3%
  {\doifsomething{#3}
     {\bgroup
      \ScaledPointsToBigPoints{#1}\width
      \ScaledPointsToBigPoints{#2}\height
      \special
        {\@@insertpostscriptretain
           [/Action /GoTo
            /Rect [0 0 \width\space \height]
            /Border [0 0 0]
            /Page #3
            /View [/Fit]
            /LNK
           pdfmark}%
      \egroup}}
```

4
```
\definespecial\dostartgotolocation#1#2#3#4#5%
  {\pdfgoto{#1}{#2}{#5}}
```

5
```
\definespecial\dostartgotorealpage#1#2#3#4%
  {\pdfgoto{#1}{#2}{#4}}
```

6
```
\stopspecials
```

\dostartgotolocation ●	\dostartthisislocation ●
\dostartgotorealpage ●	

8.10 Adobe PDF version 1.1

\dostartthisislocation
\dostartgotolocation
\usepagedestination

Staring with version 1.2 of the PDF–format destinations can be specified using `(strings)` instead of `/names`. Using strings is faster and consumes less memory. This module installe version 1.1 specials.

1
```
\unprotect
```

2
```
\startspecials[pdf]
```

3
```
\definespecial\dostartthisislocation#1%
  {\ifusepagedestinations \else
     \setPDFdestination{#1}%
     \doifsomething{\PDFdestination}
       {\special
          {\@@insertpostscriptretain
             [/Dest /\PDFdestination\space
              /View [/Fit]
              /DEST
             pdfmark}}%
   \fi}
```

4
```
\definespecial\dostartgotolocation#1#2#3#4#5%
  {\bgroup
   \ScaledPointsToBigPoints{#1}\width
   \ScaledPointsToBigPoints{#2}\height
   \doifelsenothing{#3}
     {\!!doneafalse}
     {\doifparentfileelse{#3}
        {\!!doneafalse}
        {\!!doneatrue}}%
```

```
\ifusepagedestinations
  \doifnot{0#5}{0}
    {\special
       {\@@insertpostscriptretain
          [\if!!donea
             /Action /GoToR
             /File (#3.pdf)
           \else
             /Action /GoTo
           \fi
           /Rect [0 0 \width\space \height]
           /Border [0 0 0]
           /View [/Fit]
           /Page \PDFdestination\space
           /Subtype /Link
           /ANN
          pdfmark}}%
\else
  \setPDFdestination{#4}%
  \doifsomething{\PDFdestination}
    {\special
       {\@@insertpostscriptretain
          [\if!!donea
             /Action /GoToR
             /File (#3.pdf)
           \else
             /Action /GoTo
           \fi
           /Rect [0 0 \width\space \height]
```

```
                 /Border [0 0 0]
                 /Dest /\PDFdestination\space
                 /Subtype /Link
                 /ANN
               pdfmark}}%
   \fi
   \egroup}
```

5
```
\stopspecials
```

6
```
\protect
```

\dostartgotolocation ●
\dostartthisislocation ●
\usepagedestination ●

8.11 Y&Y's DVIWINDO

1

```
\unprotect
```

As told before, these were the first interactive specials. In those days, these kind of specials were still elegant and straightforward.

\dosetupinteraction
\dostartthisislocation
\dostartthisisrealpage
\dostartgotolocation
\dostartgotorealpage
\dostartrunprogram
\usepagedestinations

```
\startspecials[dviwindo]

\definespecial\dosetupinteraction%
  {\showmessage{\m!interactions}{21}{dviwindo}}

\definespecial\dostartgotolocation#1#2#3#4#5%
  {\bgroup
   \doifelsenothing{#3}
     {\!!doneafalse}
     {\doifparentfileelse{#3}
        {\!!doneafalse}
        {\!!doneatrue}}%
   \special
     {button:
        #1 #2
        \if!!donea
          file: #3,
        \fi
        \ifusepagedestinations
          page:#5
        \else
          "#4"
        \fi}%
```

4

spec-ini
spec-mis
spec-dvi
spec-ps
spec-yy
spec-tr
spec-tpd
spec-pdf
spec-1p0
spec-1p1
spec-win
spec-etc
spec-mne
spec-htm

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
   \egroup}
```

5
```
\definespecial\dostartgotorealpage#1#2#3#4%
  {\bgroup
   \doifelsenothing{#3}
     {\!!doneafalse}
     {\doifparentfileelse{#3}
        {\!!doneafalse}
        {\!!doneatrue}}%
   \special
     {button:
        #1 #2
        \if!!donea
          file: #3,
        \fi
        page:#4}%
   \egroup}
```

6
```
\definespecial\dostartthisislocation#1%
  {\ifusepagedestinations
     \special{mark: "#1"}%
   \fi}
```

7
```
\definespecial\dostartthisisrealpage#1%
  {}
```

8
```
\definespecial\dostartrunprogram#1#2#3%
  {\special{button: #1 #2 launch: #3}}
```

9
```
\let\doyandyinsertmov = \docommoninsertmov
\let\dotrinsertmov    = \docommoninsertmov
```

spec-ini
spec-mis
spec-dvi
spec-ps
spec-yy
spec-tr
spec-tpd
spec-pdf
spec-1p0
spec-1p1
spec-win
spec-etc
spec-mne
spec-htm

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
\stopspecials

\protect
```

\dosetupinteraction ●	\dostartthisisrealpage ●
\dostartgotolocation ●	
\dostartgotorealpage ●	\usepagedestinations ●
\dostartrunprogram ●	
\dostartthisislocation ●	

8.12 General Ones

TEX produces files in the DVI format. This format is well defined and stable. In this format one-byte commands are used which can optionally be followed by length specifiers and arguments. The DVI–format incorporates a channel to the outside world. This channel is activated by the TEX primitive `\special`. The sequence

```
\special{Hello here I am.}
```

results in DVI–codes:

```
xxx1 16 Hello here I am.
```

The `xxx1` is represented in byte code 239 and the number of following bytes in a 1, 2, 3 or 4 byte number. So here we get $1 + 1 + 16$ bytes of code.

Translating these codes is upto the DVI driver. It's common use to ignore specials that cannot be interpreted, so the example string should have no consequences for the output.

In 1996 I became involved in a discussion on the distribution of electronic math journals, of which many are produced with TEX. In this EMJ list a few targets were formulated. One of these was the development of a utility that would enable whatever driver to correctly interpret whatever special format. This dvi-etc idea originates at Larry Siebenmann, who also published and talked on this subject on several meetings of TEX user groups. The second target is more ambitious and concerns communication between DVI previewers and other programs.

Being responsible for the development of part of the utility, I decided to define this `spec-etc` special module. It's main purpose was providing a starting point for and conversions. As a result, some original specials were redefined (to start–stop alternatives) to permit use of these modules in other packages.

1
```
\writestatus{loading}{Context Special Macros / dvi-etc}
```

I decided to save as much info in the specials as available, just be able to support as many drivers as possible.

2
```
\startspecials[etc]
```

\dostartgraymode
\dostopgraymode
\dostartrgbcolormode
\dostartcmykcolormode
\dostartgraycolormode
\dostopcolormode

Switching to and from color can be done in two ways:

1. insert driver specific commands
2. pass instructions to the output device

The first approach is more general and lays the responsibility at the driver side. Probably due to the fact that TEX does not directly support color, we have been confronted for the last few years with changing special definitions. The need for support depends on how a macro package handles colored text that crosses the page boundary. Again, there are two approaches.

1. let TEX do the job
2. let the driver handle things

The first approach is as driver independant as possible and can easily be accomplished by using TEX's mark mechanism. In CONTEXT we follow this approach. More and more, drivers are starting to support color, including stacking them.

Colors as well as grayscales can be represented in scales from 0 to 1. When drivers use values in the range 0..255, this value has to be adapted in the translation process. Technically it's possible to get a grayscale from combining colors. In the RGB color system, a color with Red, Green and Blue components of 0.80 show the same gray as a Gray Scale specified 0.80. The CMYK color system supports a Black component apart from Cyan, Magenta and Yellow.

Depending on the target format, color support differs from gray support. PostScript for example offers different operators for setting gray and color. This is because printing something using three colors is someting else than printing with just black.

In CONTEXT we have implemented a color subsystem that supports the use of well defined colors that, when printed in black and white, still can be distinguished. This approach enables us to serve both printed and electronic versions, using colored text and illustrations. More on the fundamentals of this topic can be found in the MAPS of the Dutch User Group, 14 (95.1).

To satsfy all those needs, we define four specials which supply enough information for drivers to act upon. We could have used more general commands with the keywords 'rgb' and 'gray', but because these specials are used often, we prefer the more direct and shorter alternative.

```
\definespecial\dostartgraymode#1%
  {\special{:etc:startgray g #1}}
```

```
\definespecial\dostopgraymode%
  {\special{:etc:stopgray}}
```

```
\definespecial\dostartrgbcolormode#1#2#3%
  {\special{:etc:startrgbcolor r #1 g #2 b #3}}
```

```
\definespecial\dostartcmykcolormode#1#2#3#4%
  {\special{:etc:startcmykcolor c #1 m #2 y #3 k #4}}
```

```
\definespecial\dostartgraycolormode#1%
  {\special{:etc:startgraycolor g #1}}
```

```
\definespecial\dostopcolormode%
  {\special{:etc:stopcolor}}
```

For some drivers, the stop special is of no use and can simply call the start one with zero arguments.

\doinsertfile

The support of inserting files (like illustrations) comes in many flavors. Some drivers use scales, some take dimensions. Some need offsets and others act on stored characteristics. They need one thing in common: a filename. Although separate specials for different formats sometimes are more clear, we decided to combine them all in one:

```
\definespecial\doinsertfile#1#2#3#4#5#6#7#8#9%
  {\special{:etc:insertfile
     type    #1
     file    #2
     xscale  #3
     yscale  #4
     xoffset #5
     yoffset #6
     width   #7
     height  #8
     options #9}}
```

No start–stop construction is needed here, because there in no further interference of TEX. All dimensions are output as scaled points and scales as a number, where 100 equal 100%.

\dostartgotolocation
\dostartgotorealpage

When we want to support hypertext buttons, again we have to deal with two concepts.

1. let TEX highlight the text
2. let the driver show us where to click

The first approach is the most secure one. It gives us complete control over the visual appearance of hyper buttons. The second alternative lets the driver guess what part of the text needs highlighting. As long as we deal with not too complicated textual buttons, this is no problem. It's even a bit more efficient when we take long mid paragraph active regions into account. When we let TEX handle active sentences *for instance marked like this one*, we have to take care of line- and pagebreaks ourselve. However, it's no trivial matter to let a driver find out where things begin and end. Because most hyperlinks can be found in tables of contents and registers, the saving in terms of bytes can be neglected and the first approach is a clear winner.

The most convenient way of cross–referencing is using named destinations. A more simple scheme is using page numbers as destinations. Because the latter alternative can often be implemented more efficient, and because we cannot be sure what scheme a driver supports, we always have to supply a pagenumber, even when we use named destinations.

To enable a driver to find out what to make active, we have to provide begin and endpoints, so like with color, we use pairs of specials. The first scheme can be satisfied with proper dimensions of the areas to be made active.

10
```
\definespecial\dostartgotolocation#1#2#3#4#5%
  {\special{:etc:startgotolocation
     width  #1
     height #2
     file   #3
     label  #4
     page   #5}}
```

11
```
\definespecial\dostopgotolocation%
  {\special{:etc:stopgotolocation}}
```

12
```
\definespecial\dostartgotorealpage#1#2#3#4%
  {\special{:etc:startgotopage
     width  #1
     height #2
     file   #3
     page   #4}}
```

spec-ini
spec-mis
spec-dvi
spec-ps
spec-yy
spec-tr
spec-tpd
spec-pdf
spec-1p0
spec-1p1
spec-win
spec-etc
spec-mne
spec-htm

13
```
\definespecial\dostopgotorealpage%
  {\special{:etc:stopgotopage}}
```

One may wonder why jumps to page and location are not combined. By splitting them, we enable macro–packages to force the prefered alternative, while on the other hand drivers can pick up the alternative desired most.

\dogotolocation
\dogotorealpage

Launching a program is done in a similar way. Here too we provide the dimensions of the active area, and here too we support start–stop handling.

14
```
\definespecial\dostartrunprogram#1#2#3%
  {\special{:etc:startrunprogram
     width  #1
     height #2
     file   #3}}
```

15
```
\definespecial\dostoprunprogram%
  {\special{:etc:stoprunprogram}}
```

\dostartthisislocation
\dostartthisisrealpage

Before we can goto some location or page, we have to tell the system where it can be found. Because some drivers follow the SGML approach of begin–end tags, we have to support pairs. A possible extension to this scheme is supplying coordinates for viewing the text.

16
```
\definespecial\dostartthisislocation#1%
  {\special{:etc:startmarklocation label #1}}
```

17
```
\definespecial\dostopthisislocation%
  {\special{:etc:stopmarklocation}}
```

18
```
\definespecial\dostartthisisrealpage#1%
  {\special{:etc:startmarkpage page #1}}
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

spec-ini
spec-mis
spec-dvi
spec-ps
spec-yy
spec-tr
spec-tpd
spec-pdf
spec-1p0
spec-1p1
spec-win
spec-etc
spec-mne
spec-htm

19
```
\definespecial\dostopthisisrealpage%
  {\special{:etc:stopmarkpage}}
```

In CONTEXT we don't use the four `\stopsomething` macros because we let TEX do the typography.

After these core specials, we come to some more specialized macros concerning rotation, background and borders, hyper links and printing.

`\dosetupidentity` We can identify the document with some strings. It's completely upto the driver to do something with this information.

20
```
\definespecial\dosetupidentity#1#2#3#4#5%
  {\special{:etc:setupidentity
      title   "#1"
      subject "#2"
      author  "#3"
      creator "#4"
      date    "#5"}}
```

`\dosetuppage` We can identify the document with some strings. It's completely upto the driver to do something with this information.

21
```
\definespecial\dosetuppaper#1#2#3%
  {\special{:etc:setuppaper
      type   #1
      width  #2
      height #3}}
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

\dosetupprinter
\dosetupscreen

Sometimes we want to tell the printer or the device that displays the text a bit more about the dimensions used. CONTEXT for instance uses the screen setup for starting up full screen.

The format is one of `letter`, `legal`, `a4`, `a5` etc. The dimensions are again scaled points. The full screen option is signaled by `0` or `1`.

22
```
\definespecial\dosetupprinter#1#2#3#4#5%
  {\special{:etc:setupprinter
     format  #1
     hoffset #2
     voffset #3
     width   #2
     height  #3}}
```

23
```
\definespecial\dosetupscreen#1#2#3#4#5%
  {\special{:etc:setupscreen
     hoffset    #1
     voffset    #2
     width      #3
     height     #4
     fullscreen #5}}
```

24
```
\stopspecials
```

\dogotolocation ●
\dogotorealpage ●
\doinsertfile ●
\dosetupidentity ●
\dosetuppage ●
\dosetupprinter ●
\dosetupscreen ●
\dostartcmykcolormode ●
\dostartgotolocation ●
\dostartgotorealpage ●
\dostartgraycolormode ●
\dostartgraymode ●
\dostartrgbcolormode ●
\dostartthisislocation ●
\dostartthisisrealpage ●
\dostopcolormode ●
\dostopgraymode ●

8.13 General Mnemonic Ones

```
\definespecial\dostartgraymode#1%
  {\special{:etc:bgm #1}}

\definespecial\dostopgraymode%
  {\special{:etc:egm}}

\definespecial\dostartrgbcolormode#1#2#3%
  {\special{:etc:bcr #1 #2 #3}}

\definespecial\dostartcmykcolormode#1#2#3#4%
  {\special{:etc:bcc #1 #2 #3 #4}}

\definespecial\dostartgraycolormode#1%
  {\special{:etc:bcg #1}}

\definespecial\dostopcolormode%
  {\special{:etc:ecm}}

\definespecial\doinsertfile#1#2#3#4#5#6#7#8#9%
  {\special{:etc:ins #1 #2 #3 #4 #5 #6 #7 #8 #9}}

\definespecial\dostartgotolocation#1#2#3#4#5%
  {\special{:etc:bgl #1 #2 #3 #4 #5}}

\definespecial\dostopgotolocation%
  {\special{:etc:egl}}

\definespecial\dostartgotorealpage#1#2#3#4%
  {\special{:etc:bgp #1 #2 #3 #4}}
```

```
\definespecial\dostopgotorealpage%
  {\special{:etc:egp}}

\definespecial\dostartrunprogram#1#2#3%
  {\special{:etc:brp #1 #2 #3}}

\definespecial\dostoprunprogram%
  {\special{:etc:erp}}

\definespecial\dostartthisislocation#1%
  {\special{:etc:bml #1}}

\definespecial\dostopthisislocation%
  {\special{:etc:eml}}

\definespecial\dostartthisisrealpage#1%
  {\special{:etc:bmp #1}}

\definespecial\dostopthisisrealpage%
  {\special{:etc:emp}}

\definespecial\dosetupidentity#1#2#3#4#5%
  {\special{:etc:sid "#1" "#2" "#3" "#4" "#5"}}

\definespecial\dosetuppaper#1#2#3%
  {\special{:etc:pap #1 #2 #3}}

\definespecial\dosetupprinter#1#2#3#4#5%
  {\special{:etc:spr #1 #2 #3 #4 #5}}

\definespecial\dosetupscreen#1#2#3#4#5%
  {\special{:etc:ssc #1 #2 #3 #4 #5}}
```

8.14 HTML

1
```
\unprotect
```

\dosetupinteraction
\dostartthisislocation
\dostopthisislocation
\dostartthisisrealpage
\dostopthisisrealpage
\dostartgotolocation
\dostopgotolocation
\dostartgotorealpage
\dostopgotorealpage

The HTML way of specifying linked locations is adapted by some drivers. This kind of hypertext support originated in the LaTeX world. Because we let TeX take care of all typography, we fake some content with a \hbox.

```
\startspecials[html]

\definespecial\dosetupinteraction%
  {\showmessage{\m!interactions}{21}{html}}

\def\htmlstartgoto#1#2#3#4%
  {\special
     {html: <a href="#3\string###4">}}

\def\htmlstartthisis#1%
  {\special
     {html: <a name="#4">}}

\def\htmlstop%
  {\special
     {html: </a>}}
```

6

7
```
\definespecial\dostartgotolocation#1#2#3#4#5%
  {\htmlstartgoto{#1}{#2}{#3}{#4}}
```

8
```
\definespecial\dostopgotolocation%
  {\htmlstop}
```

```
\definespecial\dostartgotorealpage#1#2#3#4%
  {\htmlstartgoto{#1}{#2}{#3}{page:#4}}

\definespecial\dostopgotorealpage%
  {\htmlstop}

\definespecial\dostartthisislocation#1%
  {\htmlstartthisis{#1}}

\definespecial\dostopthisislocation%
  {\htmlstop}

\definespecial\dostartthisisrealpage#1%
  {\htmlstartthisis{page:#1}}

\definespecial\dostopthisisrealpage%
  {\htmlstop}

\stopspecials

\protect
```

9 Core Commands

9.1 [to be documented: core-gen]

This module is not yet fully documented.

9.2 Verbatim

1
```
\writestatus{loading}{Context Core Macros / Verbatim}
```

2
```
\unprotect
```

We are going to embed the general verbatim support macros in a proper environment. First we show the common setup macro, so we know what features are supported. The options are hooked into the support macros via the `\obey` macros.

3
```
\def\setupcommonverbatim#1%
  {\def\verbatimfont{\tttf}%
   \doifvalue{#1\c!spatie}{\v!aan}
     {\def\obeyspaces{\setcontrolspaces}}%
   \doifvalue{#1\c!tab}{\v!aan}
     {\def\obeytabs{\settabskips}}%
   \doifvalue{#1\c!pagina}{\v!nee}
     {\def\obeypages{\ignorepages}}%
   \ExpandFirstAfter\processaction
     [\getvalue{#1\c!optie}]
     [\v!commandos=>\def\obeycharacters{\setupcommandsintype{#1}},
         \v!schuin=>\let\obeycharacters=\setupslantedtype
                    \let\obeytabs=\ignoretabs,
        \v!normaal=>\let\obeycharacters=\setupgroupedtype,
          \v!geen=>\let\obeycharacters=\relax,
         \v!kleur=>\let\obeytabs=\ignoretabs
                    \let\obeycharacters=\setupprettytextype]}
```

The verbatim commands have a rather long and turbulent history. Most users of CONTEXT probably will never use some of the features, but I've kept in mind that when one is writing a users manual, about everything can and undoubtly will be subject to a verbatim treatment.

Verbatim command are very sensitive to argument processing, which is a direct result of the ⟨*catcodes*⟩ being fixed at reading time. With our growing understanding of TEX, especially of the mechanism that can be used for looking ahead and manipulating ⟨*catcodes*⟩, the verbatim support became more and more advanced and natural.

Typesetting inline verbatim can be accomplished by `\type`, which in this sentence was typeset by saying just `\type{\type}`, which in turn was typeset by .... Using the normal grouping characters `{}` is the most natural way of using this command.

A second, more or less redundant, alternative is delimiting the argument with an own character. This method was implemented in the context of a publication in the MAPS, where this way of delimiting is recognized by LATEX users.

The third, more original alternative, is the one using `<<` and `>>` as delimiters. This alternative can be used in situations where slanted typeseting is needed.

4
```
\def\lesscharacter     {<}
\def\morecharacter     {>}
```

5
```
\chardef\texescape      = `\\
\chardef\leftargument   = `\{
\chardef\rightargument  = `\}
```

\type

We define `\type` as a protected command. First we set the catcodes of `<` and `>` and then we start looking ahead.

6
```
\unexpanded\def\type%
  {\bgroup
   \catcode`\<=\@@other
   \catcode`\>=\@@other
   \futurelet\next\dotype}
```

Next we distinguish between the three alternatives and call for the appropriate macros.

7
```
\def\dotype%
  {\ifx\next\bgroup
     \initializetype
     \initializetypegrouping
     \def\next%
       {\afterassignment\protectfirsttype\let\next=}%
   \else\if\next<%
     \doifelse{\@@tyoptie}{\v!geen}
       {\initializetype
        \setupnotypegrouping
        \def\next%
          {\let\next=}}
       {\def\next<##1%
          {\initializetype
           \if##1<%
           \else
             \setupalternativetypegrouping
             ##1%
           \fi}}%
   \else
     \def\next##1%
       {\initializetype
        \catcode`##1=\@@endgroup}%
   \fi\fi
   \next}
```

8
```
\bgroup
\catcode`\[=\@@begingroup
\catcode`\]=\@@endgroup
\catcode`\{=\@@active
\catcode`\}=\@@active
\gdef\initializetypegrouping%
  [\catcode`\{=\@@active
   \catcode`\}=\@@endgroup         % otherwise things go wrong ...
   \def\activerightargument%
     [\rightargument
      \egroup]%
   \def\activeleftargument%
     [\bgroup
      \leftargument               %% this way TeXEdit can check: {
      \catcode`\}=\@@active        % ... in alignments (tables)
      \let}=\activerightargument]%
   \let{=\activeleftargument]%    %% this way TeXEdit can check: }
\egroup
```

9
```
\bgroup
\catcode`\<=\@@active
\catcode`\>=\@@active
\gdef\setupalternativetypegrouping%
  {\catcode`\<=\@@active
   \catcode`\>=\@@active
   \def<%
     {\bgroup
      \switchslantedtype}%
   \def>%
     {\egroup}}
\egroup
```

10
```
\def\setupnotypegrouping%
  {\catcode`\<=\@@begingroup
   \catcode`\>=\@@endgroup}
```

When writing the manual to CONTEXT and documenting this source we needed to typeset `<<` and `>>`. Because we wanted to do this in the natural way, we've adapted the original definition a bit. We still show teh original because we think it's shows a bit better what we are doing.

```
   \bgroup
   \catcode`\<=\@@active
   \catcode`\>=\@@active
   \gdef\setupgroupedtype%
     {\catcode`\<=\@@active
      \catcode`\>=\@@active
      \def<%
        {\def\do%
           {\ifx\next<%
              \def\next{\bgroup\switchslantedtype\let\next=}%
            \else
              \let\next=\lesscharacter
            \fi
            \next}%
         \futurelet\next\do}%
      \def>%
        {\def\do%
           {\ifx\next>%
              \def\next{\egroup\let\next=}%
            \else
              \let\next=\morecharacter
            \fi
```

```
              \next}%
           \futurelet\next\do}}
     \egroup
```

The final implementation looks a bit further and treats the lone << and >> a bit different.

11
```
\def\doenterdoublelesstype%
  {\ifx\next\egroup
      \lesscharacter\lesscharacter
   \else
     \bgroup\switchslantedtype
     \let\doenterdoublemoretype=\egroup
   \fi}
```

12
```
\def\doenterdoublemoretype%
  {\def\doenterdoubletype
     {\ifx\next\egroup
        \morecharacter\morecharacter
      \fi}}
```

13
```
\bgroup
\catcode`\<=\@@active
\catcode`\>=\@@active
\gdef\setupgroupedtype%
  {\catcode`\<=\@@active
   \catcode`\>=\@@active
   \def<%
     {\def\do%
        {\ifx\next<%
           \def\next%
             {\def\enterdoubletype%
```

```
              {\futurelet\next\doenterdoublelesstype}%
            \afterassignment\enterdoubletype
            \let\next=}%
       \else
         \let\next=\lesscharacter
       \fi
       \next}%
    \futurelet\next\do}%
  \def>%
    {\def\do%
       {\ifx\next>%
          \def\next%
            {\def\enterdoubletype%
               {\futurelet\next\doenterdoublemoretype}%
             \afterassignment\enterdoubletype
             \let\next=}%
        \else
          \let\next=\morecharacter
        \fi
        \next}%
     \futurelet\next\do}}
\egroup
```

14
```
\newif\ifslantedtypeactivated
\newif\ifslantedtypepermitted
```

15
```
\def\switchslantedtype%
  {\ifslantedtypepermitted
     \ifslantedtypeactivated
       \slantedtypeactivatedfalse\tttf
```

```
    \else
      \slantedtypeactivatedtrue\ttsl
    \fi
  \fi}
```

16
```
\def\setupcommandsintype#1%
  {\setupgroupedtype
   \edef\!!stringa{\getvalue{#1\c!escape}}%
   \@EA\catcode\@EA`\!!stringa=\@@escape}
```

17
```
\def\setupslantedtype%
  {\setupgroupedtype
   \slantedtypepermittedtrue}
```

18
```
\bgroup
\catcode`\<=\active
\catcode`\>=\active
\gdef\doprotectfirsttype%
  {\ifx\next<%
     \let\next=\relax
   \else\ifx\next\bgroup
     \let\next=\relax
   \else\ifx\next\activeleftargument
     \let\next=\relax
   \else
     \let\next=\string
   \fi\fi\fi
   \next}
\egroup
```

19
```
\def\protectfirsttype%
  {\futurelet\next\doprotectfirsttype}
```

The neccessary initializations are done by calling `\initializetype` which in return calls for the support macro `\setupinlineverbatim`.

20
```
\def\initializetype%
  {\let\obeylines=\ignorelines
   \setupcommonverbatim{\??ty}%
   \setupinlineverbatim}
```

\setuptype Some characteristics of `\type` can be set up by:

21
```
\def\setuptype%
  {\dodoubleargument\getparameters[\??ty]}
```

\typ Although it's not clear from the macros, one character trait of this macros, which are build on top of the support module, is that they don't hyphenate. We therefore offer the alternative `\typ`. The current implementation works all right, but a decent hyphenation support of `\tt` text will be implemented soon.

22
```
\unexpanded\def\typ%
  {\bgroup
   \def\obeyedspace{ }%
   \tttf\hyphenchar\font=45
   \ttsl\hyphenchar\font=45
   \spaceskip.5em\!!plus.25em\!!minus.25em\relax
   \futurelet\next\dotype}
```

\tex
\arg
\mat
\dis

Sometimes, for instance when we pass verbatim text as an argument, the fixed ⟨*catcodes*⟩ interfere with our wishes. An experimental implementation of character by character processing of verbatim text did overcome this limitation, but we've decided not to use that slow and sometimes troublesome solution. Instead we stick to some 'old' CONTEXT macros for typesetting typical TEX characters.

The next implementation is more clear but less versatile, so we treated it for a beter one.

```
\def\dospecialtype#1#2%
  {\bgroup
   \initializetype
   \catcode`\{=\@@begingroup
   \catcode`\}=\@@endgroup
   \def\dospecialtype%
     {\def\dospecialtype{#2\egroup}%
      \bgroup
      \aftergroup\dospecialtype
      #1}%
   \afterassignment\dospecialtype
   \let\next=}

\unexpanded\def\tex{\dospecialtype\texescape\relax}
\unexpanded\def\arg{\dospecialtype\leftargument\rightargument}
\unexpanded\def\mat{\dospecialtype\$\$}
\unexpanded\def\dis{\dospecialtype{\$\$}{\$\$}}
```

Better but less readable is:

23
```
\def\doprocessgroup#1#2#3%
  {\bgroup
   #1%
   \def\doprocessgroup%
```

core-gen core-ver core-vis core-mul core-fnt core-nav core-sec core-out core-col core-fil core-blk core-rul core-ref core-mis core-enu core-lay core-int core-flo core-fig core-tab core-for core-lst core-01a core-01b core-01c core-01d core-01e core-02a core-02b core-02c core-02d

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
      {\def\doprocessgroup{#3\egroup}%
       \bgroup
       \aftergroup\doprocessgroup
       #2}%
   \afterassignment\doprocessgroup
   \let\next=}
```

24
```
\def\setgroupedtype%
  {\initializetype
   \catcode`\{=\@@begingroup
   \catcode`\}=\@@endgroup}
```

25
```
\unexpanded\def\tex{\doprocessgroup\setgroupedtype\texescape\relax}
\unexpanded\def\arg{\doprocessgroup\setgroupedtype\leftargument\rightargument}
\unexpanded\def\mat{\doprocessgroup\setgroupedtype\$\$}
\unexpanded\def\dis{\doprocessgroup\setgroupedtype{\$\$}{\$\$}}
```

\starttyping Display verbatim is realized far more easy, which is mostly due to the fact that we use \stop... as delimiter. The implementation inherits some features, for instance the support of linenumbering, which can best be studied in the documented support module.

26
```
\def\initializetyping#1%
  {\doifelsevalue{\??tp#1\c!marge}{\v!standaard}
     {\advance\leftskip by \@@sllinks}
     {\advance\leftskip by \getvalue{\??tp#1\c!marge}}%
   \setupcommonverbatim{\??tp#1}}
```

The basic display verbatim commands are defined in an indirect way. As we will see, they are a specific case of a more general mechanism.

27
```
\def\dostarttyping#1%
  {\getvalue{\??tp#1\c!voor}%
   \startopelkaar % includes \bgroup
   \initializetyping{#1}%
   \expandafter\processdisplayverbatim\expandafter{\s!stop#1}}
```

28
```
\def\dostoptyping#1%
  {\stopopelkaar  % includes \egroup
   \getvalue{\??tp#1\c!na}}
```

\definetyping

For most users the standard **\start–\stop**–pair will suffice, but for documentation purposes the next definition command can be of use:

```
   \definetyping[extratyping][margin=3em]

   \startextratyping
   these extra ones are indented by 1 em
   \stopextratyping
```

The definitions default to the standard typing values.

29
```
\def\presettyping[#1][#2]%
  {\getparameters
     [\??tp#1]
     [\c!voor=\@@tpvoor,
      \c!na=\@@tpna,
      \c!spatie=\@@tpspatie,
      \c!pagina=\@@tppagina,
      \c!tab=\@@tptab,
      \c!optie=\@@tpoptie,
      \c!marge=\@@tpmarge,
      \c!escape=\@@tpescape,
      #2]}
```

30
```
\def\dodefinetyping[#1][#2]%
  {\setvalue{\e!start#1}{\dostarttyping{#1}}%
   \setvalue{\e!stop#1}{\dostoptyping{#1}}%
   \presettyping[#1][#2]}
```

31
```
\def\definetyping%
  {\dodoubleempty\dodefinetyping}
```

32
```
\definetyping[\v!typen]
```

\setuptyping The setup of typing accepts two arguments. The optional first one identifies the user defined ones. If only one argument is given, the values apply to both the standard command `\starttyping` and `\typefile`.

33
```
\def\dosetuptyping[#1][#2]%
  {\ifsecondargument
     \getparameters[\??tp#1][#2]%
   \else
     \getparameters[\??tp][#1]%
   \fi}
```

34
```
\def\setuptyping%
  {\dodoubleempty\dosetuptyping}
```

We use the CONTEXT color system for switching to and from color mode. We can always redefine these colors afterwards.

35
```
\definecolor [texprettyone]   [r=.9, g=.0, b=.0] % red
\definecolor [texprettytwo]   [r=.0, g=.8, b=.0] % green
```

```
\definecolor [texprettythree] [r=.0, g=.0, b=.9] % blue
\definecolor [texprettyfour]  [r=.8, g=.8, b=.6] % yellow
```

We can use some core color commands. These are faster than the standard color swithing ones and work ok on a line by line basis.

36
```
\def\texbeginofpretty[#1]{\startcolormode{#1}}
\def\texendofpretty      {\stopcolormode}
```

\EveryPar \EveryLine \iflinepar One of the features of these commands is the support of `\EveryPar`, `\EveryLine` and `\iflinepar`. In the documentation of the verbatim support module we give some examples of line- and paragraph numerbering using these macros.

\typefile Typesetting files verbatim (for the moment) only supports colorization of TEX sources as valid option. The other setup values are inherited from display verbatim. The implementation of `\typefile` is straightforward:

37
```
\presettyping[\v!file][]
```

38
```
\def\typefile#1%
  {\getvalue{\??tp\v!file\c!voor}%
   \startopelkaar % includes \bgroup
   \doifnotvalue{\??tp\v!file\c!optie}{\v!kleur}
     {\setuptyping[\v!file][\c!optie=\v!geen]}%
   \initializetyping\v!file
   \processfileverbatim{#1}%
   \stopopelkaar  % includes \egroup
   \getvalue{\??tp\v!file\c!na}}
```

The setups for inline verbatim default to:

39
```
\setuptype
  [\c!spatie=\v!uit,
   \c!pagina=\v!nee,
      \c!tab=\v!nee,
    \c!optie=\v!normaal]
```

The setups for display verbatim and file verbatim are shared. One can adapt the extra defined typing environments, but they also default to the values below. Watch the alternative escape character.

40
```
\setuptyping
  [  \c!voor=\blanko,
       \c!na=\blanko,
   \c!spatie=\v!uit,
   \c!pagina=\v!nee,
      \c!tab=\v!aan,
    \c!optie=\v!geen,
    \c!marge=\!!zeropoint,
   \c!escape=/]
```

41
```
\permitshiftedendofverbatim
```

42
```
\protect
```

9.3 Visualization

This module adds some more visualization cues to the ones supplied in the support module.

1
```
\writestatus{loading}{Context Support Macros / Visualization}
```

2
```
\unprotect
```

\indent \noindent \leavevmode \par

TEX acts upon paragraphs. In mosts documents paragraphs are separated by empty lines, which internally are handled as `\par`. Paragraphs can be indented or not, depending on the setting of `\parindent`, the first token of a paragraph and/or user suppressed or forced indentation.

Because the actual typesetting is based on both explicit user and implicit system actions, visualization is only possible for the user supplied `\indent`, `\noindent`, `\leavevmode` and `\par`. Other 'clever' tricks will quite certainly lead to more failures than successes, so we only support these three explicit primitives and one macro:

3
```
\let\normalnoindent   = \noindent
\let\normalindent     = \indent
\let\normalpar        = \par
```

4
```
\let\normalleavevmode = \leavevmode
```

5
```
\def\showparagraphcue#1#2#3#4#5%
  {\bgroup
   \scratchdimen#1\relax
   \dontinterfere
   \dontcomplain
   \boxrulewidth=5\testrulewidth
   #3#4\relax
   \setbox0=\normalhbox to \scratchdimen
```

```
     {#2{\ruledhbox to \scratchdimen
           {\vrule
              #5 20\testrulewidth
              \!!width \!!zeropoint
            \normalhss}}}%
   \smashbox0
   \normalpenalty\!!tenthousand
   \box0
   \egroup}
```

6
```
\def\ruledhanging%
  {\ifdim\hangindent>\!!zeropoint\relax
     \ifnum\hangafter<0
       \normalhbox
         {\boxrulewidth=5\testrulewidth
          \setbox0=\ruledhbox to \hangindent
            {\scratchdimen=\ht\strutbox
             \advance\scratchdimen by \dp\strutbox
             \vrule
               \!!width\!!zeropoint
               \!!height\!!zeropoint
               \!!depth-\hangafter\scratchdimen}%
          \normalhskip-\hangindent
          \smashbox0
          \raise\ht\strutbox\box0}%
     \fi
   \fi}
```

7
```
\def\ruledparagraphcues%
  {\bgroup
```

```
    \dontcomplain
    \normalhbox to \!!zeropoint
      {\ifdim\leftskip>\!!zeropoint\relax
         \showparagraphcue\leftskip\llap\relax\relax\!!depth
         \normalhskip-\leftskip
       \fi
       \ruledhanging
       \normalhskip\hsize
       \ifdim\rightskip>\!!zeropoint\relax
         \normalhskip-\rightskip
         \showparagraphcue\rightskip\relax\relax\relax\!!depth
       \fi}%
    \egroup}
```

8
```
\def\ruledpar%
  {\relax
   \ifhmode
     \showparagraphcue{40\testrulewidth}\relax\rightrulefalse\relax\!!height
   \fi
   \normalpar}
```

9
```
\def\rulednoindent%
  {\relax
   \normalnoindent
   \ruledparagraphcues
   \showparagraphcue{40\testrulewidth}\llap\leftrulefalse\relax\!!height}
```

10
```
\def\ruledindent%
  {\relax
   \normalnoindent
  \ruledparagraphcues
  \ifdim\parindent>\!!zeropoint\relax
    \showparagraphcue\parindent\relax\relax\relax\!!height
  \else
    \showparagraphcue{40\testrulewidth}\llap\relax\relax\!!height
  \fi
  \normalhskip\parindent}
```

11
```
\def\ruledleavevmode%
  {\relax
   \normalleavevmode
   \ifdim\parindent>\!!zeropoint\relax
     \normalhskip-\parindent
     \ruledparagraphcues
     \showparagraphcue\parindent\relax\leftrulefalse\rightrulefalse\!!height
     \normalhskip\parindent
   \else
     \ruledparagraphcues
     \showparagraphcue{40\testrulewidth}\llap\leftrulefalse\rightrulefalse\!!height
   \fi}
```

12
```
\def\dontshowimplicits%
  {\let\noindent    = \normalnoindent
   \let\indent      = \normalindent
   \let\leavevmode = \normalleavevmode
   \let\par         = \normalpar}
```

13
```
\def\showimplicits%
  {\testrulewidth  = \defaulttestrulewidth
   \let\noindent    = \rulednoindent
\let\indent      = \ruledindent
\let\leavevmode = \ruledleavevmode
\let\par         = \ruledpar}
```

The next few–line examples show the four cues. Keep in mind that we only see them when we explicitly open or close a paragraph.

Visualizing some TEX primitives and Plain TEX macros can be very instructive, at least it is to me. Here we see `\indent` and `\ruledpar` in action, while `\parindent` equals `0.0pt`.

Visualizing some TEX primitives and Plain TEX macros can be very instructive, at least it is to me. Here we see `\noindent` and `\ruledpar` in action, while `\parindent` equals `0.0pt`.

Visualizing some TEX primitives and Plain TEX macros can be very instructive, at least it is to me. Here we see `\leavevmode` and `\ruledpar` in action, while `\parindent` equals `0.0pt`.

Visualizing some TEX primitives and Plain TEX macros can be very instructive, at least it is to me. Here we see `\indent` and `\ruledpar` in action, while `\parindent` equals `60.0pt`.

Visualizing some TEX primitives and Plain TEX macros can be very instructive, at least it is to me. Here we see `\noindent` and `\ruledpar` in action, while `\parindent` equals `60.0pt`.

Visualizing some TEX primitives and Plain TEX macros can be very instructive, at least it is to me. Here we see `\leavevmode` and `\ruledpar` in action, while `\parindent` equals `60.0pt`.

Visualizing some TEX primitives and Plain TEX macros can be very instructive, at least it is to me. Here we see `\indent` and `\ruledpar` in action, while `\parindent` equals `60.0pt`.

Visualizing some TEX primitives and Plain TEX macros can be very instructive, at least it is to me. Here we see `\noindent` and `\ruledpar` in action, while `\parindent` equals `60.0pt`.

Visualizing some TEX primitives and Plain TEX macros can be very instructive, at least it is to me. Here we see \leavevmode and \ruledpar in action, while \parindent equals 60.0pt.

These examples also demonstrate the visualization of \leftskip and \rightskip.

14
```
\newcounter\ruledbaselines
```

15
```
\def\debuggertext#1%
  {\ifx\ttxx\undefined
     $\scriptscriptstyle#1$%
   \else
     {\ttxx#1}%
   \fi}
```

16
```
\def\ruledbaseline%
  {\vrule \!!width \!!zeropoint
   \bgroup
   \dontinterfere
   \doglobal\increment\ruledbaselines
   \scratchdimen=\ht\strutbox
   \advance\scratchdimen by \dp\strutbox
   \multiply\scratchdimen by 3 % 5
   \setbox\scratchbox=\normalvbox to 2\scratchdimen
     {\leaders
      \normalhbox
        {\strut
         \vrule
           \!!width 120pt
           \!!height .5\testrulewidth
           \!!depth  .5\testrulewidth}
      \normalvfill}%
    \smashbox\scratchbox
    \advance\scratchdimen by \ht\strutbox
    \setbox\scratchbox=\normalhbox
      {\normalhskip -48pt
       \normalhbox to 24pt
         {\normalhss\debuggertext\ruledbaselines\normalhskip6pt}%
       \raise\scratchdimen\box\scratchbox}%
    \smashbox\scratchbox
    \box\scratchbox
    \egroup}
```

17

```
\def\showbaselines%
  {\testrulewidth=\defaulttestrulewidth
   \EveryPar{\ruledbaseline}}
```

18

```
\def\makecutbox#1%
  {\edef\ruledheight{\the\ht#1}%
   \edef\ruleddepth {\the\dp#1}%
   \edef\ruledwidth {\the\wd#1}%
   \setbox\scratchbox=\normalvbox
     {\dontcomplain
      \offinterlineskip
      \scratchdimen=12pt\relax
      \def\verrule##1##2%
        {\vrule
           \!!width\boxrulewidth
           \!!height##1\scratchdimen
           \!!depth##2\scratchdimen}%
      \def\horrule##1##2##3%
```

```
        {\normalhskip##1\scratchdimen
         \vrule
           \!!height\boxrulewidth
           \!!width##2\scratchdimen
         \normalhskip##3\scratchdimen}%
      \normalvskip-3\scratchdimen
      \normalhbox to \ruledwidth
        {\verrule{3}{-1}\normalhss\verrule{3}{-1}}%
      \normalhbox to \ruledwidth
        {\horrule{-3}{2}{1}\normalhss\horrule{1}{2}{-3}}%
      \normalvskip-\boxrulewidth
      \vskip\ruledheight
      \ifdim\ruleddepth>\!!zeropoint\relax
        \normalvskip-.5\boxrulewidth
        \normalhbox to \ruledwidth
          {\horrule{-2}{1}{1}\normalhss\horrule{1}{1}{-2}}%
        \normalvskip-.5\boxrulewidth
        \vskip\ruleddepth
      \fi
      \normalvskip-\boxrulewidth
      \normalhbox to \ruledwidth
        {\horrule{-3}{2}{1}\normalhss\horrule{1}{2}{-3}}%
      \normalhbox to \ruledwidth
        {\verrule{-1}{3}\normalhss\verrule{-1}{3}}}%
   \dp\scratchbox=\ruleddepth      % This re-bounding is needed and
   \ht\scratchbox=\ruledheight     % surfaced while typesetting continuous
   \setbox\scratchbox=\normalhbox  % double culumns with pagecutmark
     {\lower\ruleddepth\box\scratchbox}%
   \setbox#1=\ifhbox#1\normalhbox\else\normalvbox\fi
```

core-gen core-ver core-vis core-mul core-fnt core-nav core-sec core-out core-col core-fil core-blk core-rul core-ref core-mis core-enu core-lay core-int core-flo core-fig core-tab core-for core-lst core-01a core-01b core-01c core-01d core-01e core-02a core-02b core-02c core-02d

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
      {\normalhbox
         {\wd#1=\!!zeropoint
          \box#1\relax
          \box\scratchbox}}%
    \wd#1=\ruledwidth
    \ht#1=\ruledheight
    \dp#1=\ruleddepth}
```

19
```
\def\cuthbox%
  {\normalhbox\bgroup
   \dowithnextbox{\makecutbox\nextbox\box\nextbox\egroup}%
   \normalhbox}
```

20
```
\def\cutvbox%
  {\normalvbox\bgroup
   \dowithnextbox{\makecutbox\nextbox\box\nextbox\egroup}%
   \normalvbox}
```

21
```
\def\cutvtop%
  {\normalvtop\bgroup
   \dowithnextbox{\makecutbox\nextbox\box\nextbox\egroup}%
   \normalvtop}
```

22
```
\protect
```

core-gen
core-ver
core-vis
core-mul
core-fnt
core-nav
core-sec
core-out
core-col
core-fil
core-blk
core-rul
core-ref
core-mis
core-enu
core-lay
core-int
core-flo
core-fig
core-tab
core-for
core-lst
core-01a
core-01b
core-01c
core-01d
core-01e
core-02a
core-02b
core-02c
core-02d

contents register context syst mult supp lang font colo spec core cont m s exit go back

\indent ●	\noindent ●
\leavevmode ●	\par ●

core-gen
core-ver
core-vis
core-mul
core-fnt
core-nav
core-sec
core-out
core-col
core-fil
core-blk
core-rul
core-ref
core-mis
core-enu
core-lay
core-int
core-flo
core-fig
core-tab
core-for
core-lst
core-01a
core-01b
core-01c
core-01d
core-01e
core-02a
core-02b
core-02c
core-02d

contents register context syst mult supp lang font colo spec core cont m s exit go back

9.4 Multi Column Ouput

```
\writestatus{loading}{Context Support Macros / Multi Column Output}

\unprotect

\protect
```

9.5 Font Support

1
```
\writestatus{loading}{Context Core Macros / Font Support}
```

2
```
\unprotect
```

\kap
\KAP
\Kap
\Kaps
\nokap

We already introduced `\kap` as way to capitalize words. This command comes in several versions:

```
\kap {let's put on a \kap{cap}}
\kap {let's put on a \nokap{cap}}
\KAP {let's put on a \\{cap}}
\Kap {let's put on a \\{cap}}
\Kaps{let's put on a cap}
```

Note the use of `\nokap`, `\\` and the nested `\kap`.

LET'S PUT ON A CAP LET'S PUT ON A cap let's put on a CAP Let's put on a CAP Let's Put On A Cap

These macros show te main reason why we introduced the smaller `\tx` and `\txx`.

```
\kap\romeins{1995}
```

This at first sight unusual capitilization is completely legal.

```
\kap{...}

...      tekst
```

core-gen core-ver core-vis core-mul core-fnt core-nav core-sec core-out core-col core-fil core-blk core-rul core-ref core-mis core-enu core-lay core-int core-flo core-fig core-tab core-for core-lst core-01a core-01b core-01c core-01d core-01e core-02a core-02b core-02c core-02d

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
\Kap{...}

...      tekst
```

```
\KAP{...}

...      tekst
```

```
\Kaps{..  ... ..}

...      tekst
```

```
\nokap{...}

...      tekst
```

3
```
\unexpanded\def\kap%
  {\futurelet\next\dokap}
```

4
```
\def\dokap%
  {\ifx\next\bgroup
     \def\next{\dodokap\relax}%
   \else
     \def\next{\dodokap}%
   \fi
   \next}
```

5
```
\def\dodokap#1#2%
  {\ifmmode\hbox\fi{\tx\ignorespaces\uppercase{#1{#2}}}}
```

6
```
\unexpanded\def\KAP#1%
  {{\def\\##1{\kap{##1}}#1}}
```

7
```
\unexpanded\def\Kap#1%
  {\KAP{\\#1}}
```

8
```
\def\nokap#1%
  {\lowercase{#1}}
```

9
```
\def\Kaps%
  {\let\processword=\Kap\processwords}
```

\Word
\Words
\WORD
\WORDS

This is probably not the right place to present the next set of macros.

```
   \Word {far too many words}
   \Words{far too many words}
   \WORD {far too many words}
   \WORDS{far too many words}

   \kap {let's put on a \kap{cap}}
   \kap {let's put on a \nokap{cap}}
   \KAP {let's put on a \\{cap}}
   \Kap {let's put on a \\{cap}}
   \Kaps{let's put on a cap}
```

This calls result in:

LET'S PUT ON A CAP LET'S PUT ON A cap let's put on a CAP Let's put on a CAP Let's Put On A Cap

```
\Woord{...}

...      tekst
```

```
\Woorden{..  ... ..}

...      tekst
```

```
\WOORD{...}

...      tekst
```

```
\WOORDEN{..  ... ..}

...      tekst
```

10
```
\def\doWord#1%
  {\uppercase{#1}}
```

11
```
\def\Word#1%
  {\doWord#1}
```

12
```
\def\doprocesswords#1 #2\od%
  {\ConvertToConstant\doifnot{#1}{}
     {\processword{#1} %
      \doprocesswords#2 \od}}
```

13
```
\def\processwords#1%
  {\doprocesswords#1 \od\unskip}
```

14
```
\def\Words%
  {\let\processwords=\Word\processwords}
```

15
```
\def\WORD#1%
  {\def\kap#1{#1}%
   \edef\next{#1}%
   \uppercase\expandafter{\next}}
```

16
```
\def\WORDS#1%
  {\WORD{#1}}
```

\stretched Stretching characters in a word is a sort of typographical murder. Nevertheless we support this manipulation for use in for instance titles.

```
\hbox to 5cm{\stretched{murder}}

\kap {let's put on a \kap{cap}}
\kap {let's put on a \nokap{cap}}
\KAP {let's put on a \\{cap}}
\Kap {let's put on a \\{cap}}
\Kaps{let's put on a cap}
```

or

LET'S PUT ON A CAP LET'S PUT ON A cap let's put on a CAP Let's put on a CAP Let's Put On A Cap

```
\opgerekt{..  ... ..}

...      tekst
```

17
```
\def\stretched%
  {\processtokens\relax\hss\relax\normalspace}
```

\underbar
\underbars
\overstrike
\overstrikes

In the rare case that we need undelined words, for instance because all font alternatives are already in use, one can use `\underbar` and `\overstrike` and their plural forms.

```
\underbars{drawing \underbar{bars} under words is a typewriter leftover}
\overstrikes{striking words makes them \overstrike{unreadable}}
```

drawing bars under words is a typewriter leftover ~~striking words makes them unreadable~~

The next macros are derived from the PLAIN TeX one, but also supports nesting. The `$` keeps us in horizontal mode and at the same time applies grouping.

```
\onderstreep{...}

...      tekst
```

```
\onderstrepen{..  ... ..}

...      tekst
```

```
\doorstreep{...}
...      tekst
```

```
\doorstrepen{..  ... ..}
...      tekst
```

18
```
\newcounter\underbarlevel
```

19
```
\def\dounderbar#1#2#3%
  {\bmath
   \setbox0=\hbox{#3}%
   \setbox2=\hbox{\vrule\!!width\wd0\!!height#1\!!depth#2}%
   \wd0=\!!zeropoint
   \box0\box2
   \emath}
```

20
```
\def\underbar#1%
  {\bgroup
   \increment\underbarlevel
   \dimen0=1.5\normallineskip   % was \dimen0=1.5\lineskip
   \dimen0=\underbarlevel\dimen0
   \dimen2=\dimen0
   \advance\dimen2 by .4pt
   \dounderbar{-\dimen0}{\dimen2}{#1}%
   \egroup}
```

```
21 \def\underbars%
     {\let\processword=\underbar\processwords}

22 \def\overstrike#1%
     {\bgroup
      \dimen0=2.5\lineskip
      \dimen2=\dimen0
      \advance\dimen2 by .4pt
      \dounderbar{\dimen2}{-\dimen0}{#1}%
      \egroup}

23 \def\overstrikes%
     {\let\processword=\overstrike\processwords}

24 \protect \endinput
```

9.6 [to be documented: core-nav]

This module is not yet fully documented.

9.7 [to be extracted: core-sec]

This module is not yet split off.

core-gen
core-ver
core-vis
core-mul
core-fnt
core-nav
core-sec
core-out
core-col
core-fil
core-blk
core-rul
core-ref
core-mis
core-enu
core-lay
core-int
core-flo
core-fig
core-tab
core-for
core-lst
core-01a
core-01b
core-01c
core-01d
core-01e
core-02a
core-02b
core-02c
core-02d

contents register context syst mult supp lang font colo spec core cont m s exit go back

9.8 [to be extracted: core-out]

This module is not yet split off.

core-gen
core-ver
core-vis
core-mul
core-fnt
core-nav
core-sec
core-out
core-col
core-fil
core-blk
core-rul
core-ref
core-mis
core-enu
core-lay
core-int
core-flo
core-fig
core-tab
core-for
core-lst
core-01a
core-01b
core-01c
core-01d
core-01e
core-02a
core-02b
core-02c
core-02d

contents register context syst mult supp lang font colo spec core cont m s exit go back

9.9 [to be extracted: core-col]

This module is not yet split off.

9.10 [to be extracted: core-fil]

This module is not yet split off.

9.11 [to be extracted: core-blk]

This module is not yet split off.

core-gen
core-ver
core-vis
core-mul
core-fnt
core-nav
core-sec
core-out
core-col
core-fil
core-blk
core-rul
core-ref
core-mis
core-enu
core-lay
core-int
core-flo
core-fig
core-tab
core-for
core-lst
core-01a
core-01b
core-01c
core-01d
core-01e
core-02a
core-02b
core-02c
core-02d

contents register context syst mult supp lang font colo spec core cont m s exit go back

9.12 [to be extracted: core-rul]

This module is not yet split off.

core-gen
core-ver
core-vis
core-mul
core-fnt
core-nav
core-sec
core-out
core-col
core-fil
core-blk
core-rul
core-ref
core-mis
core-enu
core-lay
core-int
core-flo
core-fig
core-tab
core-for
core-lst
core-01a
core-01b
core-01c
core-01d
core-01e
core-02a
core-02b
core-02c
core-02d

contents register context syst mult supp lang font colo spec core cont m s exit go back

9.13 [to be extracted: core-ref]

This module is not yet split off.

9.14 [to be extracted: core-mis]

This module is not yet split off.

9.15 [to be extracted: core-enu]

This module is not yet split off.

9.16 [to be extracted: core-lay]

This module is not yet split off.

9.17 [to be extracted: core-int]

This module is not yet split off.

core-gen
core-ver
core-vis
core-mul
core-fnt
core-nav
core-sec
core-out
core-col
core-fil
core-blk
core-rul
core-ref
core-mis
core-enu
core-lay
core-int
core-flo
core-fig
core-tab
core-for
core-lst
core-01a
core-01b
core-01c
core-01d
core-01e
core-02a
core-02b
core-02c
core-02d

contents register context syst mult supp lang font colo spec core cont m s exit go back

9.18 [to be extracted: core-flo]

This module is not yet split off.

9.19 [to be extracted: core-fig]

This module is not yet split off.

9.20 [to be extracted: core-tab]

This module is not yet split off.

9.21 [to be extracted: core-for]

This module is not yet split off.

9.22 [to be extracted: core-lst]

This module is not yet split off.

core-gen
core-ver
core-vis
core-mul
core-fnt
core-nav
core-sec
core-out
core-col
core-fil
core-blk
core-rul
core-ref
core-mis
core-enu
core-lay
core-int
core-flo
core-fig
core-tab
core-for
core-lst
core-01a
core-01b
core-01c
core-01d
core-01e
core-02a
core-02b
core-02c
core-02d

contents register context syst mult supp lang font colo spec core cont m s exit go back

9.23 [to be documented: core-01a]

This module is not yet fully documented.

9.24 [to be documented: core-01b]

This module is not yet fully documented.

9.25 [to be documented: core-01c]

This module is not yet fully documented.

9.26 [to be documented: core-01d]

This module is not yet fully documented.

9.27 [to be documented: core-01e]

This module is not yet fully documented.

core-gen
core-ver
core-vis
core-mul
core-fnt
core-nav
core-sec
core-out
core-col
core-fil
core-blk
core-rul
core-ref
core-mis
core-enu
core-lay
core-int
core-flo
core-fig
core-tab
core-for
core-lst
core-01a
core-01b
core-01c
core-01d
core-01e
core-02a
core-02b
core-02c
core-02d

contents register context syst mult supp lang font colo spec core cont m s exit go back

9.28 [to be documented: core-02a]

This module is not yet fully documented.

9.29 [to be documented: core-02b]

This module is not yet fully documented.

9.30 [to be documented: core-02c]

This module is not yet fully documented.

9.31 [to be documented: core-02d]

This module is not yet fully documented.

10 Context User Modules

10.1 [to be documented: cont-log]

This module is not yet fully documented.

cont-log
cont-new
cont-old
cont-sys

contents register context syst mult supp lang font colo spec core cont m s exit go back

10.2 [to be documented: cont-new]

This module is not yet fully documented.

cont-log
cont-new
cont-old
cont-sys

contents register context syst mult supp lang font colo spec core cont m s exit go back

10.3 [to be documented: cont-old]

This module is not yet fully documented.

cont-log
cont-new
cont-old
cont-sys

contents register context syst mult supp lang font colo spec core cont m s exit go back

10.4 [to be documented: cont-sys]

This module is not yet fully documented.

11 Extra Modules

11.1 METAPOST Support

METAPOST is both an extension and a dialect of METAFONT and is written by John Hobby. This language combines a lot of the the strength of METAFONT, TEX and POSTSCRIPT in one package: a sophisticated graphics language, high quality typography and portability based on outlines. First we need some auxiliary macro's.

```
\ifx \undefined \writestatus \input supp-mis.tex \relax \fi
```

For some reason, METAPOST needs the public domain DVI to POSTSCRIPT converter DVIPS. This symbiosis originates in the need to include the fonts (glyphs) that METAPOST uses in the POSTSCRIPT file. Driver independancy was one of my prerequisites for using METAPOST, so I decided to build this kind of support myself. Personally I consider driver dependancy a drawback for the dissemination of such a package. This module more or less decouples METAPOST and DVIPS.

The next three commands do the job. They may be called more than once:

```
\UseMetaPostFile      {filenaam}
\UseMetaPostProofFont {fontname}
\UseMetaPostGraphic   {fiilename}
```

For testing purposes there is:

```
\ShowMetaPostData
```

And for troublesome situations, like independant page processing we've got:

```
\ReUseMetaPostData
```

This module is independant of CONTEXT, therefore we start with:

```
\ifx \undefined \writestatus \input supp-mis.tex \fi
```

```
\unprotect
```

The process for generating a METAPOST illustration looks more or less like:

```
mptotex filename.mp filename.tex
tex filename.tex
dvitomp filename.dvi filename.mpx
mp filename.mp
```

The file `filename.tex` contains the TEX specific commands and ships out a page for every piece of text in the metaposting. These files look like:

```
\shipout\hbox{\smash{\hbox{\hbox{% line 10 examples.mp
$a$}\vrule width1sp}}}
\shipout\hbox{\smash{\hbox{\hbox{% line 11 examples.mp
$b$}\vrule width1sp}}}
\end{document}
```

Where the last line takes care of both PLAIN TEX and LATEX, and kindly ignores all other formats that redefined `\end`, which makes it a bit more implementation dependent. By the way, there also seems to be a dependency on `\voffset` and `\hoffset`, which should be 0pt while shiping out.

An alternative to the next solution could be a utility that generates a decent prologue file based on the `filename.mpx`. For the moment we stick to the TEX based solution. There are two methods, the first one uses the intermediate TEX file, created by METAPOST, the second one uses the graphic files themselve. The latter method is the most secure.

m-metapo
m-pictex
m-pstric
m-sgml
m-chemie
m-eenhei
m-cweb
ppchtex

contents register context syst mult supp lang font colo spec core cont m s exit go back

\UseMetaPostFile

The TEX file can be used to determine what fonts and glyphs are needed. We only have to take care of the `\shipout` and `\end`. The next command stores the glyphs in a box:

```
\UseMetaPostFile{examples}
```

We reserve a box for this manipulations and append successive calls to `\UseMetaPostFile` to this box. The local redefinition of `\shipout` takes care of the METAPOST ones, the global redefinition is needed later on. We need to reset `\everypar`, otherwise unwanted side effects can occur.

4
```
\newbox\MetaPostData
```

5
```
\def\UseMetaPostData#1%
  {\global\setbox\MetaPostData=\vbox
     {\everypar{}
      \unvbox\MetaPostData
      \hbox{#1}}%
   \global\let\shipout=\MetaPostShipOut}
```

6
```
\def\UseMetaPostFile#1%
  {\UseMetaPostData
     {\let\shipout=\relax
      \def\end##1{}
      \input #1\relax}}
```

One may wonder why we don't say `\let\end=\endinput`. It turns out that when we give an `\endinput` in the middle of a sentence, the rest of the line is still processed. This is very handy when writing macros, because these are considered one line. That way `\endinput` can be buried very deep in `\if`'s.

The box is to be shipped out on the first occasion, if possible before the first graphic inclusion, otherwise some drivers still will not be able to produce the right files; one never knows in advance

m-metapo
m-pictex
m-pstric
m-sgml
m-chemie
m-eenhei
m-cweb
ppchtex

how a driver collects and writes down its fonts. The most secure way of doing this is putting the box somewhere on the page in a white color or scaled to zero. Both mechanism can fail, for instance when we use a background, or when scaling to zero is not supported. By including the box contents, the driver will embed the right glyphs, even when they are out of sight.

We use a brute mechanism and make use of the fact that most viewers and drivers clip the page, due to physical constraints. By putting the glyphs meters above the pagebody, we can be quite sure that they never show up, even on A0 paper format.

We can put the box contents somewhere by hand, but an automatic mechanism is more safe, because that way we can take care of unwanted interference. Putting the glyphs in a box at the top of a page (raised `\maxdimen`) undoubtedly interferes with `\topskip`, so I soon decided to manipulate one of the TEX primitives that is always used and that could be overloaded without problems. The primitive best suited for this purpose was (of course) `\shipout`. As always, we save the original meaning:

7
```
\let\normalshipout=\shipout
```

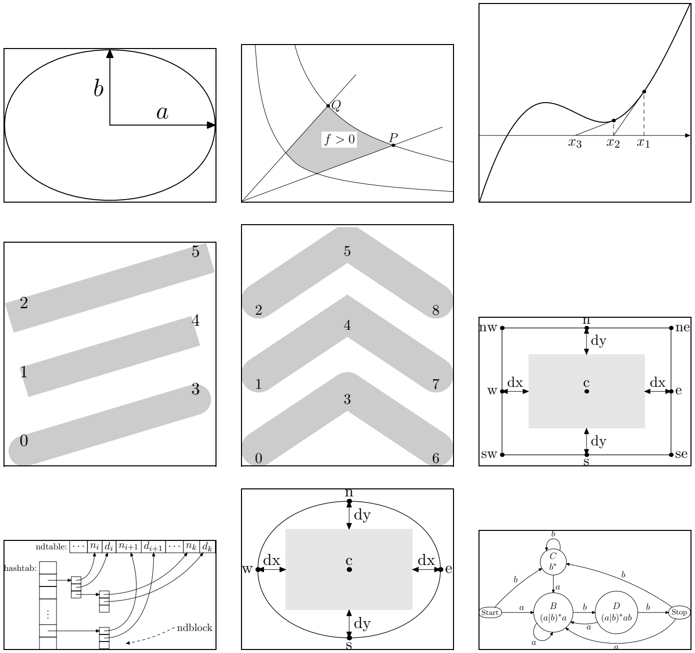
We cannot shipout the box separated from the page, because every `\shipout` generates a page. The next macros do the job.

8
```
\def\DumpMetaPostGlyphs
  {\vbox
     {\wd\MetaPostData=\!!zeropoint
      \ht\MetaPostData=\!!zeropoint
      \dp\MetaPostData=\!!zeropoint
      \kern\maxdimen
      \copy\MetaPostData
      \kern-\maxdimen}}
```

9
```
\def\RestoreShipOut%
  {\global\let\shipout=\normalshipout}
```

10
```
\def\MetaPostShipOut%
  {\dowithnextbox
     {\normalshipout\vbox
        {\DumpMetaPostGlyphs
         \nointerlineskip
         \box\nextbox}
      \RestoreShipOut}}
```

Now let's prove that things work all right and show the example files that are part of the METAPOST distribution:



In this file we preloaded the right fonts by something like:

```
\bgroup
\switchtocorps          [12pt,cmr,rm]   % the 'original' fonts
\UseMetaPostFile        {examples}      % the mp generated file
\UseMetaPostProofFont {cmr10}           % the default prooffont
```

```
\egroup
```

The first line switches to the default CONTEXT fonts and is package dependant. In real PLAIN TEX one can omit this line.

\ReUseMetaPostData

When possible, the METAPOST files are to be produced with prologues, which can be accomplished by including the next command in the METAPOST source (the `mp` file):

```
prologues := 1 ;  % this should be default
```

If after all these precautions things still go wrong, for instance because the driver produced POSTSCRIPT files on a page by page base, one can use:

```
\ReUseMetaPostData
```

After which the 'invisible' box is output at every page. The extra overhead is not that large.

11

```
\def\ReUseMetaPostData%
  {\let\RestoreShipOut=\relax}
```

In most cases the amount of extra overhead is small compared to the rest of the data.

\UseMetaPostProofFont

METAPOST does not use TEX for typesetting the proofings. This means that we have to load the used proof font, which is cmr10 by default, explicitly. The easiest way of doing this is calling an extra file, in which this font is called:

```
\UseMetaPostFile{mp-proof}
```

Such a file looks like:

```
\font\MetaPostProofFont=cmr10

\dostepwiserecurse{48}{127}{1}
```

m-metapo
m-pictex
m-pstric
m-sgml
m-chemie
m-eenhei
m-cweb
ppchtex

contents register context syst mult supp lang font colo spec core cont m s exit go back

```
     {{\MetaPostProofFont\char\recurselevel\ }}
```

or:

```
   \font\MetaPostProofFont=cmr10 0 1 2 3 4 5 6 7 8 9
```

We provide an extra routine for the prooffonts:

```
   \UseMetaPostProofFont{cmr10}
```

Because we want this module to be independant of CONTEXT, we use the more plain alternative instead of the more byte saving alternative `\dostepwiserecurse`.

12
```
\def\UseMetaPostProofFont#1%
  {\UseMetaPostData
     {\font\MetaPostProofFont=#1\relax
      \MetaPostProofFont
      \scratchcounter=32
      \loop
        \char\scratchcounter
        \hskip .5em plus .1em
        \ifnum\scratchcounter<\ifeightbitcharacters255\else126\fi
          \advance\scratchcounter by 1
      \repeat}}
```

Another pitfall lays in the format one uses. One must be sure that both the METAPOST run and the one that generates the document call the same fonts. It's best to use the same format and the same environment (e.g. corps size). Using scalable POSTSCRIPT fonts is less critical.

But, there is another way of doing things! The next solution is derived from the method we use to convert METAPOST code to PDF code.

m-metapo
m-pictex
m-pstric
m-sgml
m-chemie
m-eenhei
m-cweb
ppchtex

contents register context syst mult supp lang font colo spec core cont m s exit go back

\UseMetaPostGraphic

After writing module **supp-pdf** I decided to use a bit more advanced way, using the METAPOST created graphic files themselves.

13
```
\def\UseMetaPostGraphic#1%
  {\bgroup
   \message{[MP fonts #1]}%
   %\uncatcodespecials
   \endlinechar=-1
   \setMPspecials
   \obeyMPspecials
   \doprocessfile\scratchread{#1}\handleMPSline
   \egroup}
```

This macro scans the graphics file for the `fshow` operator, that is, lines that start with `(`. If found it interprets the line, which looks like:

```
   (string ... string) font size fshow
```

Font definitions specified in the preamble are simply ignored. Only lines starting with `(` are interpreted.

14
```
\def\dohandleMPSline#1#2\relax%
  {\if#1(%
     \expandafter\includeMPcharacters\fileline\relax
   \fi}
```

15
```
\def\handleMPSline%
  {\expandafter\dohandleMPSline\fileline\relax}
```

Before we start scanning for data, we first change some ⟨*catcodes*⟩. The first set of macro's is copied from module **supp-pdf**. This scheme is a bit overdone for this module, but using the same macros saves us some memory.

16
```
\def\octalMPcharacter#1#2#3%
  {\char'#1#2#3\relax}
```

17
```
\bgroup
\catcode`\|=\@@comment
\catcode`\%=\@@active
\catcode`\[=\@@active
\catcode`\]=\@@active
\catcode`\{=\@@active
\catcode`\}=\@@active
\catcode`B=\@@begingroup
\catcode`E=\@@endgroup
\gdef\ignoreMPspecials|
  B\def%BE|
   \def[BE|
   \def]BE|
   \def{BE|
   \def}BEE
\gdef\obeyMPspecials|
  B\def%B\char 37\relax E|
   \def[B\char 91\relax E|
   \def]B\char 93\relax E|
   \def{B\char123\relax E|
   \def}B\char125\relax EE
\gdef\setMPspecials|
  B\catcode`\%=\@@active
   \catcode`\[=\@@active
   \catcode`\]=\@@active
   \catcode`\{=\@@active
   \catcode`\}=\@@active
```

m-metapo
m-pictex
m-pstric
m-sgml
m-chemie
m-eenhei
m-cweb
ppchtex

contents register context syst mult supp lang font colo spec core cont m s exit go back

m-metapo
m-pictex
m-pstric
m-sgml
m-chemie
m-eenhei
m-cweb
ppchtex

```
   \catcode`\$=\@@letter
   \catcode`\_=\@@letter
   \catcode`\#=\@@letter
   \catcode`\^=\@@letter
   \catcode`\&=\@@letter
   \catcode`\|=\@@letter
   \catcode`\~=\@@letter
   \def\(B\char40\relax     E|
   \def\)B\char41\relax     E|
   \def\\B\char92\relax     E|
   \def\0B\octalMPcharacter0E|
   \def\1B\octalMPcharacter1E|
   \def\2B\octalMPcharacter2E|
   \def\3B\octalMPcharacter3E|
   \def\4B\octalMPcharacter4E|
   \def\5B\octalMPcharacter5E|
   \def\6B\octalMPcharacter6E|
   \def\7B\octalMPcharacter7E|
   \def\8B\octalMPcharacter8E|
   \def\9B\octalMPcharacter9EE
\egroup
```

The lines starting with (are interpreted and handled by

18
```
\def\includeMPcharacters(#1) #2 #3 #4\relax%
  {\UseMetaPostData{\font\temp=#2 at #3bp\temp#1}}
```

This method is both robust and reasonable fast. The only disadvantage is that one has to load all graphics. This method is completely macro package independant.

\ShowMetaPostData

One may wonder what happens behind the screens. If wanted and needed one can show the texts METAPOST uses by calling:

```
\ShowMetaPostData
```

Because the labels have no height and depth, we use a bit different definition of `\UseMetaPostData`. This time we force a decent linedistance. Because we also want to typeset this data in this text, we also enable linebreaks and correct some spacing. This is how it looks:

˗ ! ” # $ % & ’ ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; ¡ = ¿ ? @ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [“] ˆ ˙ ‘ a b c d e f g h i j k l m n o p q r s t u v w x y z – — ˝ ˜

a b C b^* f $>$ D 0 $(a|b)^*ab$ P Q x_1 x_2 x_3 $\cdots$ n_i d_i n_{i+1} d_{i+1} n_k d_k $\vdots$ ndblock B $(a|b)^*a$

b b a a a b b a a b

The size of the characters corresponds to the size used during the TEX run needed for the METAPOST job. The vertical spacing is not optimal, but suits its purpose.

The less instructive definitions of both macros complete this module.

19
```
\def\UseMetaPostData#1%
  {\global\setbox\MetaPostData=\vbox
     {\everypar{}
      \unvbox\MetaPostData
      \prevdepth\!!zeropoint
      %\baselineskip30pt
      \ignorespaces#1}%
   \global\let\shipout=\MetaPostShipOut}
```

m-metapo
m-pictex
m-pstric
m-sgml
m-chemie
m-eenhei
m-cweb
ppchtex

contents register context syst mult supp lang font colo spec core cont m s exit go back

20
```
\def\UseMetaPostFile#1%
  {\UseMetaPostData
     {\def\shipout{ \discretionary{}{}{}}
      \def\end##1{}
      \input #1\relax}}
```

21
```
\def\ShowMetaPostData%
  {\unvbox\MetaPostData
   \vskip15pt}
```

This time I can't prove that things work ok, simply because the right glyphs are already in the file.

```
\UseMetaPostGraphic{mp-exa-1}
\UseMetaPostGraphic{mp-exa-2}
\UseMetaPostGraphic{mp-exa-3}
\UseMetaPostGraphic{mp-exa-4}
\UseMetaPostGraphic{mp-exa-5}
\UseMetaPostGraphic{mp-exa-6}
\UseMetaPostGraphic{mp-exa-7}
\UseMetaPostGraphic{mp-exa-8}
\UseMetaPostGraphic{mp-exa-9}
```

Would have done the job.

So in order to get the right glyphs in the POSTSCRIPT file we can use `\UseMetaPostFile` for loading the TEX file and `\UseMetaPostProofFont` for loading additional fonts:

1. say `prologues:=1` in the METAPOST file
2. (temporary) activate the fonts used in the graphs
3. reuse METAPOST data when needed
4. load the proofing font when used

or if we want the graphics do the job (the prefered way) use instead `\UseMetaPostGraphic`:

1. preload all graphic files
2. reuse METAPOST data when needed

This module will probably be enhanced and/or improved, when I'm past the first experiences with METAPOST. I did consider scanning the POSTSCRIPT file that METAPOST produces. A little bit of scanning and interpreting could do the job quite well, but I wonder if it could be done robust.

```
22  \protect
```

`\ReUseMetaPostData` ●

`\ShowMetaPostData` ●

`\UseMetaPostFile` ●
`\UseMetaPostGraphic` ●
`\UseMetaPostProofFont` ●

m-metapo
m-pictex
m-pstric
m-sgml
m-chemie
m-eenhei
m-cweb
ppchtex

contents register context syst mult supp lang font colo spec core cont m s exit go back

11.2 PICTEX Loading Macros

m-metapo
m-pictex
m-pstric
m-sgml
m-chemie
m-eenhei
m-cweb
ppchtex

TEX provides 256 ⟨*dimensions*⟩ and 256 ⟨*skips*⟩. In CONTEXT this is no problem, but in packages that have many authors, one can be quite sure that a lot of ⟨*dimensions*⟩ are allocated. Packages that use PICTEX can therefore run out of ⟨*dimensions*⟩ quite fast. This module was written as a reaction to persistent problems with loading PPCHTEX in LATEX and PICTEX deserves a solution. I therefore dedicate this module to Tobias Burnus and Dirk Kuypers, who use PPCHTEX in a LATEX environment and suggested a lot of extensions to the repertoire of PPCHTEX commands.

This module presents a solution that is quite effective: all ⟨*dimensions*⟩ are drawn from the pool of ⟨*dimensions*⟩ and ⟨*skips*⟩, depending on the availability. This is possible because ⟨*dimensions*⟩ are ⟨*skips*⟩ without a glue component. Therefore we can use ⟨*skips*⟩ as ⟨*dimensions*⟩. However, some incompatibility can result from assignments that look like:

```
\somedimen=\someskip
```

In such cases the ⟨*dimension*⟩ equals the fixed part of the ⟨*skip*⟩ or in other words: this assignment strips off the glue. Because PICTEX uses no glue components, I thought I could interchange both register types without problems, but alas, this didn't hold for all ⟨*dimensions*⟩.

In PLAIN TEX the allocation macros are defined with (as) `\outer`. This means that they cannot appear inside macros, not even in an indirect way. We therefore have to redefine both `\newdimen` and `\newskip` to non–`\outer` alternatives. In most macro packages this redefinition already took place. We save the original meanings, so we can restores them afterwards.

1
```
\let\normalnewdimen = \newdimen
\let\normalnewskip  = \newskip
```

2
```
\catcode`@=11 % I'd rather used \unprotect
\def\temporarynewdimen {\alloc@1\dimen\dimendef\insc@unt}
\def\temporarynewskip  {\alloc@2\skip \skipdef \insc@unt}
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

`\catcode`@=12 % and \protect.`

Here comes the trick. Depending on how many ⟨*dimensions*⟩ and ⟨*skips*⟩ are allocated, the `\newdimen` assigns a ⟨*dimensions*⟩ or ⟨*skip*⟩. PLAIN TEX allocates 15 ⟨*dimensions*⟩ and 17 ⟨*skips*⟩. After loading PICTEX, 71 ⟨*dimensions*⟩ and and 71 ⟨*skips*⟩ are allocated. Indeed, PICTEX needs 110 ⟨*dimensions*⟩!

```
\def\newdimen%
  {\ifnum\count11>\count12
      \let\next=\temporarynewskip
   \else
      \let\next=\temporarynewdimen
   \fi
   \next}
```

When I was testing a new version of PPCHTEX in PLAIN TEX I had to find out that this exchange of registers sometimes leads to unwanted results. It took me some hours to find out that the source of errors originated in constructions like:

```
\ifdim\DimenOne<\DimenTwo whatever you want \else or not \fi
```

When `\DimenOne` is a ⟨*skip*⟩ and `\DimenTwo` is a ⟨*dimension*⟩, TEX scans for some optional glue component, like in:

```
\skip0=\dimen0 plus 10pt minus 5pt
```

The most robust solution to this problem is:

```
\ifdim\DimenOne<\DimenTwo\relax right \else wrong \fi
```

Some close reading of the PICTEX source however learned me that this problem could be solved best by just honoring the allocation of ⟨*dimensions*⟩ when the name of the macro explictly stated the character sequence `dimen`. A next implementation therefore automatically declared all ⟨*dimensions*⟩

with this sequence in their names with `\dimen`. Again I was too optimistic, so now we do it this way (the comments are from PICTEX, which like TABLE, is an example of a well documented package):

3
```
\catcode`!=11
\temporarynewdimen\!dimenA        %.AW.X.DVEUL..OYQRST
\temporarynewdimen\!dimenB        %....X.DVEU...O.QRS.
\temporarynewdimen\!dimenC        %..W.X.DVEU......RS.
\temporarynewdimen\!dimenD        %..W.X.DVEU....Y.RS.
\temporarynewdimen\!dimenE        %..W........G..YQ.S.
\temporarynewdimen\!dimenF        %...........G..YQ.S.
\temporarynewdimen\!dimenG        %...........G..YQ.S.
\temporarynewdimen\!dimenH        %...........G..Y..S.
\temporarynewdimen\!dimenI        %...BX.........Y....
\temporarynewdimen\!dxpos         %..W......U..P....S.
\temporarynewdimen\!dypos         %..WB.....U..P......
\temporarynewdimen\!xloc          %..WB.....U.......S.
\temporarynewdimen\!xpos          %..........L.P..Q.ST
\temporarynewdimen\!yloc          %..WB.....U.......S.
\temporarynewdimen\!ypos          %..........L.P..Q.ST
\temporarynewdimen\!zpt           %.AWBX.DVEULGP.YQ.ST
```

Tobias tested this module in all kind of LATEX dialects so we were able to find out that we also needed to declare:

4
```
\temporarynewdimen\linethickness
\catcode`!=12
```

After all, the new definition of `\newdimen` became:

5
```
\def\newdimen#1%
  {\ifx#1\undefined
```

```
      \ifnum\count11>\count12\relax
        \temporarynewskip#1\relax
      \else
        \temporarynewdimen#1\relax
      \fi
      %\edef\ascii{\meaning#1}%
      %\immediate\write20{\string#1 becomes \ascii}%
    \else
      %\edef\ascii{\meaning#1}%
      %\immediate\write20{\string#1 already is \ascii}%
    \fi}
```

Curious readers can still find the previous solution in the source. The next macro is used instead of `\input`. This macro also reports some statistics.

6
```
\def\dimeninput#1 %
  {\message{[before: d=\the\count11,s=\the\count12]}%
   \input #1 \relax
   \message{[after: d=\the\count11,s=\the\count12]}}%
```

Not every package defines `\fiverm`, PICTEX's pixel, so let's take care of that omision now:

7
```
\ifx\undefined\fiverm
  \font\fiverm=cmr5
\fi
```

The actual loading of PICTEX depends on the package. For LATEX users we take care of loading the auxiliary ones too.

8
```
\ifx\beginpicture\undefined
  \ifx\newenvironment\undefined
```

```
    \dimeninput pictex.tex    \relax
  \else
    \dimeninput prepictex.tex \relax
    \dimeninput pictex.tex    \relax
    \dimeninput postpictex.tex \relax
  \fi
\fi
```

Finally we restore the old definitions of `\newdimen` and `\newskip`:

9
```
\let\newdimen = \normalnewdimen
\let\newskip  = \normalnewskip
```

and just hope for the best.

10
```
\endinput
```

m-metapo
m-pictex
m-pstric
m-sgml
m-chemie
m-eenhei
m-cweb
ppchtex

contents register context syst mult supp lang font colo spec core cont m s exit go back

11.3 [to be documented: m-pstric]

This module is not yet fully documented.

m-metapo
m-pictex
m-pstric
m-sgml
m-chemie
m-eenhei
m-cweb
ppchtex

contents register context syst mult supp lang font colo spec core cont m s exit go back

11.4 [to be documented: m-sgml]

This module is not yet fully documented.

m-metapo
m-pictex
m-pstric
m-sgml
m-chemie
m-eenhei
m-cweb
ppchtex

contents register context syst mult supp lang font colo spec core cont m s exit go back

11.5 [to be documented: m-chemie]

This module is not yet fully documented.

m-metapo
m-pictex
m-pstric
m-sgml
m-chemie
m-eenhei
m-cweb
ppchtex

contents register context syst mult supp lang font colo spec core cont m s exit go back

11.6 [to be documented: m-eenhei]

This module is not yet fully documented.

11.7 [to be documented: m-cweb]

This module is not yet fully documented.

m-metapo
m-pictex
m-pstric
m-sgml
m-chemie
m-eenhei
m-cweb
ppchtex

contents register context syst mult supp lang font colo spec core cont m s exit go back

11.8 [to be documented: ppchtex]

This module is not yet fully documented.

12 Dedicated Setup Examples

s-pre-01
s-pro-01

contents register context syst mult supp lang font colo spec core cont m s exit go back

12.1 Presentation Environment 1

s-pre-01
s-pro-01

1
```
\startenvironment s-she-01
```

This environment can be used to typeset interactive presentations. This module was first used at the 1997 TUG meeting.

\language

Because thismodule is defined in english, we default to the english hyphenation patterns and labels too.

2
```
\language
  [en]
```

\setupcorps
\switchtocorps

For screen reading, a Lucida Bright font looks nice. We use the default 12 point size for ornaments, but switch the main text size to 14.4 point.

3
```
\setupcorps
  [lbr]
```

4
```
\switchtocorps
  [14.4pt]
```

\setupcolors
\definecolor

Screen presentations without color just look dull, so we enable color support. We define ourselves a yellowish backgroundcolor and a not too dark blue interactioncolor.

5
```
\setupcolors
  [status=start]
```

6
```
\definecolor [backgroundcolor]  [r=1,  g=1,  b=.7]
\definecolor [interactioncolor] [r=.1, g=.5, b=.8]
```

\setuppapersize
\setuplayout
\setupinteractionscreen

We use a nice large screen, and dedicate the right edge and bottom part to navigational tools. We automatically set the width and height of the page and start up full screen.

s-pre-01
s-pro-01

7
```
\setuppapersize
  [S6]
```

8
```
\setuplayout
  [topspace=12pt,
   header=0pt,
   height=fit,
   footer=0pt,
   bottomdistance=8pt,
   bottom=10pt,
   backspace=12pt,
   margin=0pt,
   width=fit,
   edgedistance=12pt,
   rightedge=96pt]
```

9
```
\setupinteractionscreen
  [option=max,
   width=fit,
   height=fit]
```

\setupbackgrounds

We set the pagecolor to yellow except the part of the screen that is used to display the running text. By seting the offset to 3pt the text will not touch the yellow parts. We do not set the depth.

10
```
\setupbackgrounds
  [page]
  [background=color,
   backgroundcolor=backgroundcolor,
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

s-pre-01
s-pro-01

```
   offset=3pt]
```

11
```
\setupbackgrounds
  [text] [text]
  [background=color,
   backgroundcolor=white]
```

I considered the next setup too, but finaly decided to comment it out.

```
\setupbackgrounds
  [bottom] [text]
  [frame=on,
   framecolor=white]
```

\setupinteractions

We did not enable interactive text support yet, so let's do that now. We force page reference to circumvent problems with named destinations. (At the moment this module was written, both Adobe Acrobat Reader and PDFTEX gave troubles.)

12
```
\setupinteraction
  [page=yes,
   status=start]
```

\setupbottomtexts

At the bottom of the screen we show two navigational bars. At the left we show the subpage bar, at the right we use a non default backward/forward bar.

13
```
\setupbottomtexts
  [\InteractionBar]
  [\InteractionButtons]
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

\interactionbar

The left bar gets a white border (on the yellow backrgound). Because we don't want to typeset an empty frame when no subpage bar is shown, we check for the number of subpages.

s-pre-01
s-pro-01

14

```
\def\InteractionBar%
  {\ifnum\nofsubpages>1
     \framed
       [framecolor=white,height=\bottomheight,strut=no]
       {\interactionbar[alternative=e,height=1.25ex]}
   \fi}
```

\setupinteractionbar
\interactionbuttons

The right hand buttons enable us to jump back and forward, as well as to the previous and next jump. We also enable to close the presentation.

15

```
\setupinteractionbar
  [framecolor=white,
   height=\bottomheight,
   strut=no]
```

16

```
\def\InteractionButtons%
  {\interactionbuttons
     [width=15em]
     [PreviousJump,NextJump,
      firstpage,
      firstsubpage,previouspage,nextpage,lastsubpage,
      lastpage,
      CloseDocument]}
```

\topic
\subject

A presentation after loading this module looks like:

s-pre-01
s-pro-01

```
\startstandardmakeup
  \bfd \setupalign[middle]
  \vfil About Whatever
  \vfil Topics
  \vfil\vfil\vfil
\stopstandardmakeup

\topic {Some topic}

\subject {Alfa}

.....

\subject {Beta}

.....
```

\definehead

The commands `\topic` and `\sheet` are defined as copies of head.

17
```
\definehead [topic]   [paragraph]
\definehead [subject] [subparagraph]
```

\setuphead

We use our own command for typesetting the titles. We hide sectionnumbers from viewing. Each topic is followed by a list of subjects that belong to the topic.

18
```
\setuphead
  [topic,subject]
  [command=\HeadLine,
   page=yes,
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

s-pre-01
s-pro-01

```
    style=\tfb,
    sectionnumber=no]
```

19
```
\setuphead
  [topic]
  [after=\PlaceSubjectList]
```

20
```
\setuphead
  [subject]
  [continue=no]
```

\framed
\middlelined

The command used to typeset the head lines is rather simple. We just center the framed title. The frame macro optimizes the alignment and at the same time enables us to typset a nice colored rule.

21
```
\def\HeadLine#1#2%
  {\middlelined
     {\framed
        [bottomframe=on,
         framecolor=backgroundcolor,
         width=.8\hsize,
         align=middle]
        {#2}}}
```

\setuplist

The subject list is automatically placed. We center each subject line by using oen of the default alternatives (g). We could have said:

```
\setuplist
  [subject]
  [alternative=none,
   command=\SubjectListLine,
   interaction=all]
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

s-pre-01
s-pro-01

```
  \def\SubjectListLine#1#2#3%
    {\middlelined{#2}}
```

But why should we complicate things when we can say:

22

```
\def\PlaceSubjectList%
  {\blank
   \placelist[subject]}
```

23

```
\setuplist
  [subject]
  [alternative=g,
   interaction=all]
```

\setuptexttexts

The topics will be listed in the right edge, using:

24

```
\setuptexttexts
  [edge]
  [][\TopicList]
```

\setuplist
\placelist

The actual topic list is typeset using a `\vbox`. We have to specify `criterium` because otherwise no list will be typeset. (By default lists are typeset locally.)

25

```
\def\TopicList%
  {\vbox to \vsize
     {\placelist[topic][criterium=all]
      \vss}}
```

26

```
\setuplist
  [topic]
  [alternative=none,
   command=\TopicLine]
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

\limitatetext

Because topic lines can be rathe rather long, we limit their length to the width of the right edge. This macro is part of the support macros: it's there but only in english.

s-pre-01
s-pro-01

27

```
\def\TopicLine#1#2#3%
  {\limitatetext{#2}{\rightedgewidth}{...}
   \par}
```

\setuptexttexts
\button

During a presentation, we want to use the cursor to point to parts of the text. Furthermore we want to be able to jump to the next page, without too much positioning on buttons. Therefore we make the text part of the screen into an invisible button.

28

```
\setuptexttexts
  [\GotoNextPage][]
```

29

```
\def\GotoNextPage
  {\button[width=\hsize,height=\vsize,frame=off]{}[nextpage]}
```

\setupsubpagenumber

The left bottom navigation bar shows the subpages, which will be counted by topic.

30

```
\setupsubpagenumber
  [way=bytopic,
   status=start]
```

31

```
\stopenvironment
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

s-pre-01
s-pro-01

contents register context syst mult supp lang font colo spec core cont m s exit go back

s-pre-01
s-pro-01

12.2 Produkt Environment 1

1
```
\startomgeving s-pro-01
```

Deze omgeving kan worden gebruikt om een produktcatalogus te zetten. Er kan gebruik worden gemaakt van de gebruikelijke structurerende elementen en een aantal specifieke elementen.

\stellayoutin
\stelinteractieschermin

De onderstaande indeling toont ons dat het instellen nogal nauw kijkt. Dit is een gevolg van de aspect–ratio van het beeldscherm. Bovendien moeten we rekening houden met de offset van een achtergrond. We starten full–screen up.

2
```
\stellayoutin
  [rugwit=12pt,
   marge=0pt,
   linkerrand=0pt,
   breedte=454pt,
   randafstand=14pt,
   rechterrand=110pt,
   kopwit=12pt,
   boven=0pt,
   hoofd=0pt,
   hoogte=426pt,
   voetafstand=14pt,
   voet=12pt,
   onderafstand=0pt,
   onder=0pt]
```

3
```
\stelinteractieschermin
  [breedte=600pt,
   hoogte=450pt,
   optie=max]
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

\stelkleurenin

Standaard worden kleuren niet weergegeven. We kunnen dus wel overal kleuren instellen, maar ze worden pas zichtbaar wanneer het onderstaande commando wordt aangeroepen.

s-pre-01
s-pro-01

4
```
\stelkleurenin
  [status=start]
```

\stelwitruimtein

We houden van een ruim opgezette tekst, ondanks het gebrek aan ruimte op het scherm.

5
```
\stelwitruimtein
  [groot]
```

\stelkorpsin
\stelkopin

We maken gebruik van de $\mathcal{AMS}$-TEX fonts. Verder gebruiken we de 12 punts Computer Modern Roman (de standaard instelling).

6
```
\stelkorpsin
  [ams]
```

\stelachtergrondenin

Omdat we op het scherm zowel een tekst- als een navigatiedeel onderscheiden, gebruiken we een achtergrond.

7
```
\stelachtergrondenin
  [pagina]
  [offset=4pt]
```

8
```
\stelachtergrondenin
  [tekst,voet]
  [tekst]
  [achtergrond=raster]
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

s-pre-01
s-pro-01

\definieersorteren
\merk

In produktinformatie komt nogal vaak een merknaam voor. Daarom definiëren we een sortering:

9
```
\definieersorteren
  [merk]
  [merken]
```

Normaal roepen we een sortering of synoniem op met een commando. Omdat we hier spaties (mogen) gebruiken, moeten we gebruik maken van `\naam`. Stel dat we bijvoorbeeld de volgende merken hebben gedefineerd:

```
    \merk  [type 45/60]          {Type 45/60}
    \merk  [eshalite labiel c]   {Eshalite Labiel C}
```

dan roepen we zo'n merk op met:

```
    ..... ..... ..... \naam{type 45/60} ..... ..... ..... .....
```

Overigens gebruiken we bewust niet `produkt` in plaats van `merk`, omdat het commando `\produkt` al bestaat!

\stelinteractiein

Standaard is een tekst *niet* interactief. Ook worden menu's standaard niet getoond. De hier ingestelde kleur en het gekozen lettertype worden door de andere commando's overgenomen, tenzij daar anders ingesteld. Bij de keuze van de kleur moeten we rekening houden met de kwaliteit van het de weergave en kleurenblindheid.

10
```
\stelinteractiein
  [status=start,
   menu=aan,
   letter=normaal,
   kleur=donkergroen]
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

\definieerregister
\stelregisterin
\stelkopteksthin
\soortprodukt
\toepassing
\index

We maken twee (extra) registers aan. Een voor de soort produkten (merken) en een voor de toepassingen van deze produkten. We kiezen vervolgens voor een afwijkende koptekst. Standaard wordt bij het plaats van een index een verwijzing aangemaakt; we zullen dit straks in het nog te definiëren menu zien.

s-pre-01
s-pro-01

11
```
\definieerregister [soortprodukt] [soortprodukten]
\definieerregister [toepassing]   [toepassingen]
```

12
```
\stelregisterin    [soortprodukt] [symbool=2,aanduiding=nee]
\stelregisterin    [toepassing]   [symbool=2,aanduiding=nee]
\stelregisterin    [index]        [symbool=2,aanduiding=nee]
```

13
```
\stelkoptekstin
  [soortprodukt=Register produkten,
   toepassing=Register toepassingen]
```

\definieerkop
\produktgroep
\produktsubgroep
\produktsubsubgroep
\produkthoofdstuk

In de tekst kunnen we gebruik maken van de gebruikelijke structurende elementen, zoals `\hoofdstuk`. Omwille van de leesbaarheid definiëren we echter ook wat meer gerichte elementen.

```
\definieerkop [produkthoofdstuk]    [hoofdstuk]
```

15
```
\definieerkop [produktgroep]        [paragraaf]
\definieerkop [produktsubgroep]     [subparagraaf]
\definieerkop [produktsubsubgroep]  [subsubparagraaf]
```

Ook stellen we de (standaard) koppen wat anders in:

16
```
\stelkopin
  [hoofdstuk]
  [letter=\bfd]
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

s-pre-01
s-pro-01

17
```
\stelkopin
  [titel]
  [letter=\bfd]
```

18
```
\stelkopin
  [paragraaf]
  [letter=\bfc]
```

19
```
\stelkopin
  [subparagraaf]
  [letter=\bfb]
```

20
```
\stelkopin
  [subsubparagraaf]
  [letter=\bfa]
```

21
```
\stelkopin
  [subsubsubparagraaf]
  [letter=\bf]
```

\definieersamengesteld..
\stelsamengesteldelijs..
\vollediginhoud

Standaard wordt per kop een lijst gedefinieerd. We kunnen zo'n lijst oproepen met het commando `\plaatslijst`. We hebben echter een voorkeur voor een samengestelde lijst, omdat dergelijke lijsten standaard worden doorgelust. Omdat we de bestaande structurerende elementen en de nieuwe door elkaar gebruiken, passen we de definitie van de inhoudsgave wat aan:

22
```
\definieersamengesteldelijst
  [inhoud]
  [hoofdstuk,produkthoofdstuk,
   paragraaf,produktgroep,
   subparagraaf,produktsubgroep,
   subsubparagraaf,produktsubsubgroep,
   subsubsubparagraaf]
```

23
```
\stelsamengesteldelijstin
  [inhoud]
  [symbool=2,
   paginanummer=nee,
   voor=,
   na=,
   breedte=2em]
```

\stelkopin

Deze instellingen lijken op het eerste gezicht wat ingewikkeld. Het mechanisme om koppen te plaatsen is namelijk vrij geavanceerd, omdat rekening moet worden gehouden met nummers, markeringen, afbreken, verwijzingen enz.

We plaatsen elke kop op een nieuwe bladzijde, gevolgd door een hoofdstuk. Op deze bladzijde plaatsen we ook een inhoudsopgave. Op het laagste niveau gebruiken we een (automatische) prefix voor de verwijzingen, omdat we in het menu naar de verschillende onderdelen willen kunnen springen. Bij de andere koppen moeten we de prefix natuurlijk uitzetten, omdat anders de (inhoudsopgaven in de) menu's doorlopen.

24
```
\def\hoofdstukinhoud%
  {\blanko
   \plaatsinhoud[niveau=paragraaf]%
   \iflijstgeplaatst
     \pagina
   \else
     \blanko
   \fi}
```

25
```
\stelkopin
  [hoofdstuk]
```

s-pre-01
s-pro-01

```
     [na=\hoofdstukinhoud]

26 \stelkopin
     [titel]
     [na=\blanko]

27 \stelkopin
     [paragraaf]
     [voor=\blanko,
      na=\blanko]

28 \stelkopin
     [subparagraaf]
     [voor=\blanko,
      na=\blanko]

29 \stelkopin
     [subsubparagraaf]
     [voor=\blanko,
      na=\blanko]

30 \stelkopin
     [subsubsubparagraaf]
     [voor=\blanko,
      na=\blanko]

31 \stelkopin
     [produkthoofdstuk]
     [na={\blanko\plaatsinhoud[niveau=produktgroep]}]

32 \stelkopin
     [produktgroep]
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

s-pre-01
s-pro-01

```
  [prefix=-,
   pagina=ja,
   doorgaan=nee,
   na={\blanko\plaatsinhoud[niveau=produktsubgroep]}]
```

33
```
\stelkopin
  [produktsubgroep]
  [prefix=-,
   pagina=ja,
   doorgaan=nee,
   na={\blanko\plaatsinhoud[niveau=produktsubsubgroep]}]
```

34
```
\stelkopin
  [produktsubsubgroep]
  [prefix=+,
   pagina=ja,
   doorgaan=nee,
   na=\blanko]
```

\geenproduktsubgroepen

Wanneer geen produktsubgroepen aanwezig zijn, moeten we de teller handmatig ophogen en zelf een inhoud plaatsen.

35
```
\def\geenproduktsubgroepen%
  {\stelkopnummerin[produktsubgroep][+1]%
   \plaatsinhoud}
```

\produktinformatie

Bij de produktinformatie gebruiken we de kopcommando's indirekt. Dit maakt het mogelijk de merken op naam op te roepen, wat twee voordelen heeft:

1. we gebruiken de merknaam als verwijzing en kunnen dus naar het betreffende onderdeel springen
2. we roepen de merknaam op en garanderen zo een consiste vormgeving

contents register context syst mult supp lang font colo spec core cont m s exit go back

s-pre-01
s-pro-01

We kunnen dus volstaan met de volgende kopdefinitie, voor de rest wordt automatisch gezorgd:

```
    \produktinformatie[eshalite labiel c]
```

We genereren bovendien een verwijzing naar het register met produkten.

36
```
\def\produktinformatie[#1]%
  {\produktsubsubgroep[#1]{\naam{#1}}
   \soortprodukt{\naam{#1}}}
```

\definieerkop \rubriek \mogelijkheden \gegevens \verwerking \leveringswijze \eigenschappen

Bij de uiteindelijke omschrijving van een produkt gebruiken we de onderstaande indeling. We maken gebruik van een nummerloze kop `\rubriek`. Let op: deze kop moeten we op een lager niveau dan `\produktsubsubgroep` definiëren!

37
```
\definieerkop
  [rubriek]
  [subsubsubonderwerp]
```

38
```
\def\mogelijkheden  {\rubriek [mogelijkheden]  {Mogelijkheden}}
\def\gegevens       {\rubriek [gegevens]       {Technische gegevens}}
\def\verwerking     {\rubriek [verwerking]     {Verwerking}}
\def\leveringswijze {\rubriek [leveringswijze] {Leveringswijze}}
\def\eigenschappen  {\rubriek [eigenschappen]  {Eigenschappen}}
```

\stelkoppenin

We willen geen kopnummers zien, hoewel voor testdoeleinden zichtbare nummers zondermeer handig zijn.

39
```
\stelkoppenin
  [sectienummer=nee]
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

s-pre-01
s-pro-01

\stelvoetin
\stelvoettekstenin

Omdat we willen weten waar we zitten, genereren we een voettekst, die we in een vette letter tonen.

40
```
\stelvoetin
  [letter=vet]
```

41
```
\stelvoettekstenin
  [tekst]
  [hoofdstuk]
  [produktsubsubgroep]
```

\definieermarkering
\koppelmarkering

Het volgende commando verwachten we hier eigenlijk niet maar is toch nodig. Standaard is namelijk `produktsubsubgroep` gekoppeld aan `subsubparagraaf`. Dit geldt ook voor de markeringen, die in beide gevallen weer gekoppeld zijn aan `sectie-5`, het niveau van subsubparagrafen. De volgende commando's herdefinieren de markering en koppel deze zelfstandig aan het niveau van de subparagrafen (oftewel `sectie-5`).

42
```
\definieermarkering
  [produktsubsubgroep]
```

43
```
\koppelmarkering
  [produktsubsubgroep]
  [subsubparagraaf]
```

Dergelijke manipulaties veronderstellen nogal wat inzicht in de werking van CONTEXT, vandaar dat we hier wat meer uitleg geven. Er zijn (standaard) 7 niveaus van koppen. Bij elke overgang naar een nieuwe kop gebeurt, afhankelijk van de instellingen, vrij veel:

1. overgaan op een nieuwe bladzijde
2. de lokale prefix uitschakelen
3. tellers ophogen en resetten
4. onderliggende markeringen resetten
5. de kop plaatsen
6. verwijzingen toekennen
7. naar een lijst schrijven
8. de eigen markering instellen

contents register context syst mult supp lang font colo spec core cont m s exit go back

s-pre-01
s-pro-01

In geval van een interactieve tekst gebeurt ook het volgende:

9. synchroniseren
10. de plaats markeren

Het tijdstip van het plaatsen van de kop is zo uitgekiend dat verwijzingen, markeringen en het schrijven naar de lijst in de pas loopt. Een gebruiker kan eigen commando's aan het mechanisme koppelen, maar zal gebruik moeten blijven maken van het ingebakken pagina–overgang–mechanisme.

Elke genummerde kop heeft een eigen markering. Wanneer een kop echter de kenmerken erft van een andere kop, dan erft ze ook de markering. We mogen bij het instellen van bijvoorbeeld de voetregels beide kop–namen gebruiken. Vaak is dit ook wat we wensen, omdat een andere kop meestal niet meer is dan een variatie in vormgeving. In het geval van de produktgroepen willen we deze koppeling echter voor wat betreft de markeringen loslaten, omdat niet elk hoofdstuk vergelijkbare subsubparagrafen heeft. Door de betreffende markering te herdefiniëren en herkoppelen vervalt de directe relatie met de subsubparagraaf. Alleen het niveau blijft overeenkomen.

\stelinteractiebalkin
\stelvoettekstenin

In de rand plaatsen we een eenvoudig doch doeltreffend navigatie–instrument.

44
```
\stelinteractiebalkin
  [achtergrond=raster,
   kader=uit,
   kleur=rood]
```

45
```
\stelvoettekstenin
  [rand]
  []
  [\geentest{\interactiebalk[variant=b]}]
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

\blokkopjes
\figuren

Het is niet zo zinvol figuren (plaatsblokken) te nummeren, vandaar dat we de nummers uitzetten.

s-pre-01
s-pro-01

46
```
\stelblokkopjesin
  [nummer=nee]
```

\stelinteractiemenuin

We kiezen voor menu's (rechts) met een grijze achtergrond en rode letters. Als een menu–item niet beschikbaar is, dan wordt de achtergrond wel getoond, maar de tekst niet. Om te voorkomen dat ten onrechte naar een volgende pagina wordt gesprongen — dit is standaard gedrag van Acrobat in full–screen mode — zijn de grijze vlakken wel actief, dat wil zeggen: we springen naar de huidige bladzijde.

We zetten het kader natuurlijk uit. Als we geen achtergrond kiezen maar kaders, dan worden alleen de beschikbare items getoond. De gebruiker kan het verschil tussen beide alternatieven zien door het kader aan te laten staan.

47
```
\stelinteractiemenuin
  [rechts]
  [achtergrond=raster,
   hoogte=18pt,
   kader=uit,
   kleur=rood]
```

We definiëren het rechter menu als volgt. Let op het gebruik van `\vfilll`. Door in plaats van `inhoud` de verwijzing `vorigeinhoud` te gebruiken, kunnen we door de inhoudsopgaven terug springen.

48
```
\stelinteractiemenuin
  [rechts]
  [{inhoud[vorigeinhoud]},
   {produkten[soortprodukt]},
   {toepassingen[toepassing]},
   {\vfilll},
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

s-pre-01
s-pro-01

```
   {mogelijkheden[mogelijkheden]},
   {gegevens[gegevens]},
   {verwerking[verwerking]},
   {leveringswijze[leveringswijze]},
   {eigenschappen[eigenschappen]},]
```

\geentekstbeschikbaar Tot slot een commando dat handig is bij het opzetten van (grotendeels lege) structuren:

49
```
\def\geentekstbeschikbaar%
  {{\em Er is nog geen tekst beschikbaar cq.\ ingevoerd}}
```

\omgeving We laden achtereenvolgens de merken, afkortingen, logos, figuren en eenheden. Bovendien laden we standaard een omgeving `o-layout` waarin variaties op de hier gedefinieerde layout kunnen worden aangebracht.

50
```
\omgeving o-merken
\omgeving o-afkortingen
\omgeving o-logos
\omgeving o-figuren
\omgeving o-eenheden
\omgeving o-layout
```

51
```
Het is geen bezwaar als een of meer van deze omgevingen er
niet zijn.
```

52
```
\stopomgeving
```

contents register context syst mult supp lang font colo spec core cont m s exit go back

s-pre-01
s-pro-01

s-pre-01
s-pro-01

contents register context syst mult supp lang font colo spec core cont m s exit go back

Register

s-pre-01
s-pro-01

s-pre-01
s-pro-01

contents register context syst mult supp lang font colo spec core cont m s exit go back

s-pre-01
s-pro-01

contents register context syst mult supp lang font colo spec core cont m s exit go back

s-pre-01
s-pro-01

s-pre-01
s-pro-01

contents register context syst mult supp lang font colo spec core cont m s exit go back

s-pre-01
s-pro-01

s-pre-01
s-pro-01

contents register context syst mult supp lang font colo spec core cont m s exit go back

s-pre-01
s-pro-01

contents register context syst mult supp lang font colo spec core cont m s exit go back

s-pre-01
s-pro-01

contents register context syst mult supp lang font colo spec core cont m s exit go back

s-pre-01
s-pro-01

contents register context syst mult supp lang font colo spec core cont m s exit go back

s-pre-01
s-pro-01

s-pre-01
s-pro-01

contents register context syst mult supp lang font colo spec core cont m s exit go back

PRAGMA

bureau voor
onderwijskundig
advies- en
ontwikkelwerk

s-pre-01
s-pro-01

Postbus 125 8000AC Zwolle NL | Ridderstraat 27 8061GH Hasselt NL | kvk: 050 48142
tel: +31 (0)38 477 53 69 | fax: 477 53 74 | e-mail: pragma@pi.net | postbank: 4604213 | abn-amro: 59.22.03.700