CONTEXT

Language Options

group: CONTEXT Support Macros

version: 1997.03.20

date: 1997 July 25

author: Hans Hagen

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.

```
\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.[1] 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

[1] In 1996 the spelling of the dutch language has been slightly reformed which made this topic actual again.

\installdisc.. 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

\compoundhyp.. \beginofsubs.. \endofsubsen.. 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:

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 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.

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
    \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
     \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.

\beginofsubs..
\endofsubsen..

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.

\enableactiv..

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
   \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.[2] 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.

`\installcomp..` 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}
```

[2] In the CONTEXT manual the `<` and `>` are made active and used for some cross–reference trickery.

```
\installcompoundcharacter "e {\"e}
\installcompoundcharacter "i {\"i}
\installcompoundcharacter "u {\"u}
\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~}}%
   \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
             \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{\dododohandlecompoundcharacter#1#2\blankspace\ignorespaces}%
       \else
         \def\next{\dododohandlecompoundcharacter#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
```