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

\CONTEXT

\CONTEXT

\CONTEXT

\unprotect

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT

\CONTEXT
The mechanism described here is one of the older inner parts of \texttt{CONTEXT}. The most recent extensions concern some special cases as well as the possibility to install other characters as delimiters. The preferred way of specifying compound words is using \texttt{||}, which is installed by:

\begin{verbatim}
\installdiscretionaries || -
\end{verbatim}

Some alternative definitions are:

\begin{verbatim}
\installdiscretionaries ** -
\installdiscretionaries ++ -
\installdiscretionaries // -
\installdiscretionaries ~~ -
\end{verbatim}

after which we can say:

\begin{verbatim}
\texttt{test**test**test}
\texttt{test++test++test}
\texttt{test//test//test}
\texttt{test--test--test}
\end{verbatim}

Now let's go to the macros. First we define some variables. In the main \texttt{CONTEXT} modules these can be tuned by a setup command. Watch the (maybe) better looking compound hyphen.

\begin{verbatim}
\def\compoundhyphen {{-}\kern-.25ex{-}}
\def\beginofsubsentence {---}
\def\endofsubsentence {---}
\end{verbatim}

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:

\begin{verbatim}
\def\prewordbreak {\penalty10000\hskip0pt\relax}
\def\postwordbreak {\penalty0\prewordbreak}
\end{verbatim}

We first show the original implementation, which only supports | as command and delimiter. Before activating | we save it's value:

\begin{verbatim}
\edef\domainmodediscretionary{|}
\end{verbatim}

We need a two stage \texttt{\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 \texttt{\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 \texttt{\nextnext}.  

\begin{verbatim}
\edef\domainmodediscretionary{|}
\end{verbatim}

after which we're ready to define it's meaning to:

\begin{verbatim}
\catcode'|=\active
\end{verbatim}

\begin{verbatim}
\edef\domainmodediscretionary{|}
\end{verbatim}

after which we're ready to define it's meaning to:

\begin{verbatim}
\catcode'|=\active
\end{verbatim}

\begin{verbatim}
\edef\domainmodediscretionary{|}
\end{verbatim}

after which we're ready to define it's meaning to:

\begin{verbatim}
\catcode'|=\active
\end{verbatim}

\begin{verbatim}
\edef\domainmodediscretionary{|}
\end{verbatim}

We need a two stage \texttt{\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 \texttt{\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 \texttt{\nextnext}. 

\begin{verbatim}
\edef\domainmodediscretionary{|}
\end{verbatim}

after which we're ready to define it's meaning to:

\begin{verbatim}
\catcode'|=\active
\end{verbatim}

\begin{verbatim}
\edef\domainmodediscretionary{|}
\end{verbatim}

after which we're ready to define it's meaning to:

\begin{verbatim}
\catcode'|=\active
\end{verbatim}

\begin{verbatim}
\edef\domainmodediscretionary{|}
\end{verbatim}

We need a two stage \texttt{\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 \texttt{\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 \texttt{\nextnext}. 

\begin{verbatim}
\edef\domainmodediscretionary{|}
\end{verbatim}

after which we're ready to define it's meaning to:

\begin{verbatim}
\catcode'|=\active
\end{verbatim}

\begin{verbatim}
\edef\domainmodediscretionary{|}
\end{verbatim}

after which we're ready to define it's meaning to:

\begin{verbatim}
\catcode'|=\active
\end{verbatim}

\begin{verbatim}
\edef\domainmodediscretionary{|}
\end{verbatim}

We need a two stage \texttt{\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 \texttt{\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 \texttt{\nextnext}. 

\begin{verbatim}
\edef\domainmodediscretionary{|}
\end{verbatim}

after which we're ready to define it's meaning to:
The main macro consists of quite some \ifx tests while \checkafterdiscretionary handles the commas. We show the simplified version here:

\def\dododotextmodediscretionary{% 
\let\next\next\next=\egroup \ifx \checkafterdiscretionary \prewordbreak\hbox{\compoundhyphen\next\next}\postwordbreak \else\ifx=\next \prewordbreak\compoundhyphen \else\ifx\next \discretionary{-}{}{(}\prewordbreak \else\ifx~\next \discretionary{-}{ }{} \postwordbreak \else \ifx(\next \prewordbreak\discretionary{}{(-}{(}\prewordbreak \else\ifx)\next \prewordbreak\discretionary{-)}{}{)}\prewordbreak \else\ifx\next \discretionary{-}{ }{'}\postwordbreak \else \checkafterdiscretionary \prewordbreak\hbox{\betweendiscretionaries\next\next}\postwordbreak \fi\fi\fi\fi\fi\fi \next\next\next}

\def\checkafterdiscretionary{% \ifx,\next\next \def\next\next\next{\afterassignment\egroup\let\next=}\% \else \let\next\next=\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}
A bit more \texttt{\char'\char\char} and character trickery enables us to discard the two intermediate steps. This trick originates on page 394 of the \TeX\ book, 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.

\def\installdiscretionaries#1#2#3\%{
  \setvalue{mathmodediscretionary\string#1}{#1}\%
  \catcode'#1='\@active
  \installdiscretionaries#1#2\%
}

\def\dodoinstalldiscretionaries\%{
  \setvalue{textmodediscretionary\string#1}{#1}\%
  \unexpanded\def#1{\discretionarycommand#1}
}

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 \texttt{\futurelet} because we have to take following characters into account.
The macro \checkbeforediscretionary takes care of loners like \textmodediscretionary||word, while its counterpart \checkafterdiscretionary is responsible for handling the comma.
In the previous macros we provided two hooks which can be used to support nested sub-sentences. In \texttt{CONTEXT} these hooks are used to insert a small space when needed.

\beginofsubs...
\endofsubsens...

\let\beginofsubs白沙ting=elax
\let\endofsubs白沙ting =elax

Before we show some more tricky alternative, we first install the mechanism:

\\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 \texttt{uccode}'s instead. \LaTeX{} initializes these codes of the alphabetic glyphs to their uppercase counterparts. Normally the other characters remain zero. If so, we can use the \texttt{uccode} as a signal.

The more advanced mechanism is activated by calling:

\\enableactive...
\enableactivediscretionaries

which is defined as:

\\def\enableactivediscretionaries\%
\{\uccode'='\relax \uccode'~='~\relax \uccode'_='\relax \uccode'(='\relax \uccode')=')\relax \uccode'='='\relax \uccode'<='<\relax \uccode'>='>\relax
\let\dotextmodediscretionary = \activedotextmodediscretionary
\let\dododotextmodediscretionary = \activeddodotextmodediscretionary\}

We only have to redefine two macros. While saving the \texttt{uccode} in a macro we have to take care of empty arguments, like in 11.

\\def\activedotextmodediscretionary#1\%
{\bgroup
\def\dodotextmodediscretionary##1#1\%
{\def\betweendiscretionary{\checkafterdiscretionary}
\def\nextuccode####1####2\relax\%
{\ifcat\noexpand####1\noexpand\relax
\edef\nextuccode{0}\%
\else
\edef\nextuccode{\the\uccode'####1}\%
\fi}
\nextuccode##1@elax
\futurelet\nextnext\dodotextmodediscretionary\%
\let\discretionarycommand=#1\%
\def\textmodediscretionary{\getvalue{\textmodediscretionary\string#1}}\%
\futurelet\next\dodotextmodediscretionary\%
}

This time we use \texttt{ifnum}:

\\def\activeddodotextmodediscretionary\%
{\let\nextnextnext=\egroup
\ifx\discretionarycommand\next
\checkafterdiscretionary
\bgroup
\checkbeforediscretionary
\prewordbreak\hbox{\textmodediscretionary\nextnext}\postwordbreak
}
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. 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 \< or \> statements.

When Tobias Burnus started translating the Dutch manual of PPCHTeX into German, he suggested to let \context support the \texttt{german.sty} method of handling compound characters, especially the umlaut.

This package is meant for use with \texttt{Plain TeX} as well as \texttt{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"

\footnote{In the \context manual the < and > are made active and used for some cross-reference trickery.}
Language Options

\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.

\def\@nc@{@nc@} % normal character
\def\@cc@{@cc@} % compound character
\def\@cs@{@cs@} % compound characters

16 \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.

17 \def\dohandlecompoundcharacter% {
\ifx\next\bgroup
\let\next=\relax
\else\ifx\next\egroup
\let\next=\relax
\else
\let\next=\dohandlecompoundcharacter%
\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 if 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.

\begin{verbatim}
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
    \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#1#2\%
    {\dodohandlecompoundcharacter#1#2}\%
    \futurelet\next\dohandlecompoundcharacter}
\end{verbatim}

In later modules we will see how these commands are used.
Language Options

\beginofsubsentence 2 \endofsubsentence 2
\beginofsubsentencespacing 6 \endofsubsentencespacing 6
\compoundhyphen 2 \installcompoundcharacter 7
\enableactivediscretionaries 6 \installdiscretionaries 2