The `soul-ori` package

Melchior FRANZ

November 17, 2003

Abstract

This article describes the `soul-ori` package[1], which provides h y p h e n a t a b l e l e t t e r s p a c i n g (s p a c i n g o u t), underlining and some derivatives such as ~~overstriking~~ and highlighting. Although the package is optimized for LaTeX 2ε, it also works with Plain TeX and with other flavors of TeX like, for instance, ConTeXt. By the way, the package name `soul` is only a combination of the two macro names `\so` (space out) and `\ul` (underline)—nothing poetic at all.

Contents

[1] This file has version number 3.0, last revised 2023-18-02.

1 Typesetting rules

There are several possibilities to emphasize parts of a paragraph, not all of which are considered good style. While underlining is commonly rejected, experts dispute about whether letterspacing should be used or not, and in which cases. If you are not interested in such debates, you may well skip to the next section.

Theory ...

To understand the experts' arguments we have to know about the conception of *page grayness*. The sum of all characters on a page represents a certain amount of grayness, provided that the letters are printed black onto white paper.

Jan Tschichold [10], a well known and recognized typographer, accepts only forms of emphasizing, which do not disturb this grayness. This is only true of italic shape, caps, and caps-and-small-caps fonts, but not of ordinary letterspacing, underlining, bold face type and so on, all of which appear as either dark or light spots in the text area. In his opinion emphasized text shall not catch the eye when running over the text, but rather when actually reading the respective words.

Other, less restrictive typographers [11] call this kind of emphasizing 'integrated' or 'aesthetic', while they describe 'active' emphasizing apart from it, which actually *has* to catch the reader's eye. To the latter group belong commonly despised things like letterspacing, demibold face type and even underlined and colored text.

On the other hand, Tschichold suggests to space out caps and caps-and-small-caps fonts on title pages, headings and running headers from 1 pt up to 2 pt. Even in running text legibility of uppercase letters should be improved with slight letterspacing, since (the Roman) majuscules don't look right, if they are spaced like (the Carolingian) minuscules.[2]

... and Practice

However, in the last centuries letterspacing was excessively used, underlining at least sometimes, because capitals and italic shape could not be used together with the *Fraktur* font and other black-letter fonts, which are sometimes also called "old German" fonts. This tradition is widely continued until today. The same limitations apply still today to many languages with non-latin glyphs, which is why letterspacing has a strong tradition in eastern countries where Cyrillic fonts are used.

The Duden [4], a well known German dictionary, explains how to space out properly: *Punctuation marks are spaced out like letters, except quotation marks and periods. Numbers are never spaced out. The German syllable* -sche *is not spaced out in cases like* "der V i r c h o w sche Versuch"[3]*. In the old German* Fraktur *fonts the ligatures* `ch`, `ck`, `sz` (ß) *and* `tz` *are not broken within spaced out text.*

While some books follow all these rules [6], others don't [7]. In fact, most books in my personal library do *not* space out commas.

[2] This suggestion is followed throughout this article, although Prof. Knuth already considered slight letterspacing with his `cmcsc` fonts.

[3] the Virchow experiment

2 Short introduction and common rules

The `soul-ori` package provides five commands that are aimed at emphasizing text parts. Each of the commands takes one argument that can either be the text itself or the name of a macro that contains text (e. g. `\so\text`)[4]. See table 1 for a complete command survey.

`\so{letterspacing}`	l e t t e r s p a c i n g
`\caps{CAPITALS, Small Capitals}`	CAPITALS, Small Capitals
`\ul{underlining}`	underlining
`\st{overstriking}`	~~overstriking~~
`\hl{highlighting}`	highlighting[5]

The `\hl` command does only highlight if the `color` package was loaded, otherwise it falls back to underlining.[6] The highlighting color is by default yellow, underlines and overstriking lines are by default black. The colors can be changed using the following commands:

`\setulcolor{red}`	set underlining color
`\setstcolor{green}`	set overstriking color
`\sethlcolor{blue}`	set highlighting color

`\setulcolor{}` and `\setstcolor{}` turn coloring off. There are only few colors predefined by the `color` package, but you can easily add custom color definitions. See the `color` package documentation [3] for further information.

```
\usepackage{color,soul}
\definecolor{lightblue}{rgb}{.90,.95,1}
\sethlcolor{lightblue}
...
\hl{this is highlighted in light blue color}
```

2.1 Some things work ...

The following examples may look boring and redundant, because they describe nothing else than common LaTeX notation with a few exceptions, but this is only the half story: The `soul-ori` package has to pre-process the argument before it can split it into characters and syllables, and all described constructs are only allowed because the package explicitly implements them.

§ 1 Accents:

Example: `\so{na\"\i␣ve}`

Accents can be used naturally. Support for the following accents is built-in: `\‘`, `\’`, `\^`, `\"`, `\~`, `\=`, `\.`, `\u`, `\v`, `\H`, `\t`, `\c`, `\d`, `\b`, and `\r`. Additionally, if the `german` package [8] is loaded you can also use the `"` accent command and write `\so{na"ive}`. See section 5.1 for how to add further accents.

[4] See § 25 for some additional information about the latter mode.

[5] The look of highlighting is nowhere demonstrated in this documentation, because it requires a Postscript aware output driver and would come out as ugly black bars on other devices, looking very much like censoring bars. Think of it as the effect of one of those coloring text markers.

[6] Note that you can also use LaTeX's `color` package with Plain TeX. See 6.1 for details.

§ 2 Quotes:

Example: `\so{‘‘quotes’’}`
The `soul-ori` package recognizes the quotes ligatures ‘‘, ’’ and ,,. The Spanish ligatures !‘ and ?‘ are not recognized and have, thus, to be written enclosed in braces like in `\caps{{!‘}Hola!}`.

§ 3 Mathematics:

Example: `\so{foo$x^3$bar}`
Mathematic formulas are allowed, as long as they are surrounded by $. Note that the LaTeX equivalent `\(...\)` does not work.

§ 4 Hyphens and dashes:

Example: `\so{re-sent}`
Explicit hyphens as well as en-dashes (--), em-dashes (---) and the `\slash` command work as usual.

§ 5 Newlines:

Example: `\so{new\\line}`
The \\ command fills the current line with white space and starts a new line. Spaces or linebreaks afterwards are ignored. Unlike the original LaTeX command `soul-ori`'s version does not handle optional parameters like in `\\*[1ex]`.

§ 6 Breaking lines:

Example: `\so{foo\linebreak␣bar}`
The `\linebreak` command breaks the line without filling it with white space at the end. `soul-ori`'s version does not handle optional parameters like in `\linebreak[1]`. `\break` can be used as a synonym.

§ 7 Unbreakable spaces:

Example: `\so{don't~break}`
The ~ command sets an unbreakable space.

§ 8 Grouping:

Example: `\so{Virchow{sche}}`
A pair of braces can be used to let a group of characters be seen as one entity, so that `soul-ori` does for instance not space it out. The contents must, however, not contain potential hyphenation points. (See § 9)

§ 9 Protecting:

Example: `\so{foo␣\mbox{little}␣bar}`
An `\mbox` also lets `soul-ori` see the contents as one item, but these may even contain hyphenation points. `\hbox` can be used as a synonym.

§ 10 Omitting:

Example: `\so{\soulomit{foo}}`
The contents of `\soulomit` bypass the soul core and are typeset as is, without being letterspaced or underlined. Hyphenation points are allowed within the argument. The current font remains active, but can be overridden with `\normalfont` etc.

§ 11 Font switching commands:
Example: `\so{foo␣\texttt{bar}}`
All standard TEX and LATEX font switching commands are allowed, as well as the `yfonts` package [9] font commands like `\textfrak` etc. Further commands have to be registered using the `\soulregister` command (see section 5.2).

§ 12 Breaking up ligatures:
Example: `\ul{Auf{}lage}`
Use `{}` or `\null` to break up ligatures like 'fl' in `\ul`, `\st` and `\hl` arguments. This doesn't make sense for `\so` and `\caps`, though, because they break up every unprotected (ungrouped/unboxed) ligature, anyway, and would then just add undesirable extra space around the additional item.

2.2 . . . others don't

Although the new `soul-ori` is much more robust and forgiving than versions prior to 2.0, there are still some things that are not allowed in arguments. This is due to the complex engine, which has to read and inspect every character before it can hand it over to TEX's paragraph builder.

§ 20 Grouping hyphenatable material:
Example: `\so{foo␣{little}␣bar}`
Grouped characters must not contain hyphenation points. Instead of `\so{foo {little}}` write `\so{foo \mbox{little}}`. You get a `'Reconstruction failed'` error and a black square like ■ in the DVI file where you violated this rule.

§ 21 Discretionary hyphens:
Example: `\so{Zu\discretionary{k-}{}{c}ker}`
The argument must not contain discretionary hyphens. Thus, you have to handle cases like the German word `Zu\discretionary{k-}{}{c}ker` by yourself.

§ 22 Nested soul commands:
Example: `\ul{foo␣\so{bar}␣baz}`
`soul-ori` commands must not be nested. If you really need such, put the inner stuff in a box and use this box. It will, of course, not get broken then.

```
\newbox\anyboxname
\sbox\anyboxname{ \so{the worst} }
\ul{This is by far{\usebox\anyboxname}example!}
```

yields:

This is by far t h e w o r s t example!

§ 23 Leaking font switches:
Example: `\def\foo{\bf␣bar}␣\so{\foo␣baz}`
A hidden font switching command that leaks into its neighborship causes a `'Reconstruction failed'` error. To avoid this either register the 'container' (`\soulregister{\foo}{0}`), or limit its scope as in `\def\foo{{\bf bar}}`. Note that both solutions yield slightly different results.

§ 24 Material that needs expansion:
Example: `\so{\romannumeral\year}`
In this example `\so` would try to put space between `\romannumeral` and `\year`, which can, of course, not work. You have to expand the argument before you feed it to `soul-ori`, or even better: Wrap the material up in a command sequence and let `soul-ori` expand it: `\def\x{\romannumeral\year} \so\x`. `soul-ori` tries hard to expand enough, yet not too much.

§ 25 Unexpandable material in command sequences:
Example: `\def\foo{\bar}␣\so\foo`
Some macros might not be expandable in an `\edef` definition[7] and have to be protected with `\noexpand` in front. This is automatically done for the following tokens: `~`, `\,`, `\TeX`, `\LaTeX`, `\S`, `\slash`, `\textregistered`, `\textcircled`, and `\copyright`, as well as for all registered fonts and accents. Instead of putting `\noexpand` manually in front of such commands, as in `\def\foo{foo {\noexpand\bar} bar} \so\foo`, you can also register them as special (see section 5.2).

§ 26 Other weird stuff:
Example: `\so{foo␣\verb|\bar|␣baz}`
`soul-ori` arguments must not contain LaTeX environments, command definitions, and fancy stuff like `\vadjust`. `soul-ori`'s `\footnote` command replacement does not support optional arguments. As long as you are writing simple, ordinary 'horizontal' material, you are on the safe side.

2.3 Troubleshooting

Unfortunately, there's just one helpful error message provided by the `soul-ori` package, that actually describes the underlying problem. All other messages are generated directly by TeX and show the low-level commands that TeX wasn't happy with. They'll hardly point you to the violated rule as described in the paragraphs above. If you get such a mysterious error message for a line that contains a `soul-ori` statement, then comment that statement out and see if the message still appears. `'Incomplete \ifcat'` is such a non-obvious message. If the message doesn't appear now, then check the argument for violations of the rules as listed in §§ 20–26.

2.3.1 `'Reconstruction failed'`

This message appears, if § 20 or § 23 was violated. It is caused by the fact that the reconstruction pass couldn't collect tokens with an overall width of the syllable that was measured by the analyzer. This does either occur when you grouped hyphenatable text or used an unregistered command that influences the syllable width. Font switching commands belong to the latter group. See the above cited sections for how to fix these problems.

[7] Try `\edef\x{\copyright}`. Yet `\copyright` works in `soul-ori` arguments, because it is explicitly taken care of by the package

	page	
`\so{letterspacing}`	8	l e t t e r s p a c i n g
`\caps{CAPITALS, Small Capitals}`	10	CAPITALS, Small Capitals
`\ul{underlining}`	12	underlining
`\st{striking out}`	12	~~striking out~~
`\hl{highlighting}`	12	highlighting
`\soulaccent{\cs}`	14	*add accent* `\cs` *to accent list*
`\soulregister{\cs}{0}`	15	*register command* `\cs`
`\sloppyword{text}`	19	*typeset* `text` *with stretchable spaces*
`\sodef\cs{1em}{2em}{3em}`	8	*define new spacing command* `\cs`
`\resetso`	8	*reset* `\so` *dimensions*
`\capsdef{////}{1em}{2em}{3em}`*	10	*define (default)* `\caps` *data entry*
`\capssave{name}`*	10	*save* `\caps` *database under name* `name`
`\capsselect{name}`*	10	*restore* `\caps` *database of name* `name`
`\capsreset`*	10	*clear caps database*
`\setul{1ex}{2ex}`	12	*set* `\ul` *dimensions*
`\resetul`	12	*reset* `\ul` *dimensions*
`\setuldepth{y}`	12	*set underline depth to depth of an* y
`\setuloverlap{1pt}`	13	*set underline overlap width*
`\setulcolor{red}`	12	*set underline color*
`\setstcolor{green}`	13	*set overstriking color*
`\sethlcolor{blue}`	13	*set highlighting color*

Table 1: List of all available commands. The number points to the page where the command is described. Those marked with a little asterisk are only available when the package is used together with LaTeX, because they rely on the *New Font Selection Scheme (NFSS)* used in LaTeX.

2.3.2 Missing characters

If you have redefined the internal font as described in section 5.3, you may notice that some characters are omitted without any error message being shown. This happens if you have chosen, let's say, a font with only 128 characters like the `cmtt10` font, but are using characters that aren't represented in this font, e. g. characters with codes greater than 127.

3 Letterspacing

3.1 How it works

`\so` The base macro for letterspacing is called `\so`. It typesets the given argument with *inter-letter space* between every two characters, *inner space* between words and *outer space* before and after the spaced out text. If we let "·" stand for *inter-letter space*, "∗" for *inner spaces* and "•" for *outer spaces*, then the input on the left side of the following table will yield the schematic output on the right side:

1.	`XX\so{aaa␣bbb␣ccc}YY`	XXa·a·a∗b·b·b∗c·c·cYY
2.	`XX␣\so{aaa␣bbb␣ccc}␣YY`	XX•a·a·a∗b·b·b∗c·c·c•YY
3.	`XX␣{\so{aaa␣bbb␣ccc}}␣YY`	XX•a·a·a∗b·b·b∗c·c·c•YY
4.	`XX␣\null{\so{aaa␣bbb␣ccc}}{}␣YY`	XX␣a·a·a∗b·b·b∗c·c·c␣YY

Case 1 shows how letterspacing macros (`\so` and `\caps`) behave if they aren't following or followed by a space: they omit outer space around the `soul-ori` statement. Case 2 is what you'll mostly need—letterspaced text amidst running text. Following and leading space get replaced by *outer space*. It doesn't matter if there are opening braces before or closing braces afterwards. `soul-ori` can see through both of them (case 3). Note that leading space has to be at least `5sp` wide to be recognized as space, because LaTeX uses tiny spaces generated by `\hskip1sp` as marker. Case 4 shows how to enforce normal spaces instead of *outer spaces:* Preceding space can be hidden by `\kern0pt` or `\null` or any character. Following space can also be hidden by any token, but note that a typical macro name like `\relax` or `\null` would also hide the space thereafter.

The values are predefined for typesetting facsimiles mainly with *Fraktur* fonts. You can define your own spacing macros or overwrite the original `\so` meaning
`\sodef` using the macro `\sodef`:

`\sodef`⟨*cmd*⟩`{`⟨*font*⟩`}{`⟨*inter-letter space*⟩`}{`⟨*inner space*⟩`}{`⟨*outer space*⟩`}`

The space dimensions, all of which are mandatory, should be defined in terms of `em` letting them grow and shrink with the respective fonts.

```
\sodef\an{}{.4em}{1em plus1em}{2em plus.1em minus.1em}
```

`\resetso` After that you can type '`\an{example}`' to get 'e x a m p l e'. The `\resetso` command resets `\so` to the default values.

3.2 Some examples

Ordinary text.	▪ `\so{electrical␣industry}` ▪ e l e c t r i c a l i n d u s t r y	▪ e l e c - t r i - c a l i n - d u s - t r y
Use `\-` *to mark hyphenation points.*	▪ `\so{man\-u\-script}` ▪ m a n u s c r i p t	▪ m a n - u - s c r i p t
Accents are recognized.	▪ `\so{le␣th\'e\^atre}` ▪ l e t h é â t r e	▪ l e t h é â t r e
`\mbox` and `\hbox` protect material that contains hyphenation points. The contents are treated as one, unbreakable entity.	▪ `\so{just␣an␣\mbox{example}}` ▪ j u s t a n example	▪ j u s t a n example
Punctuation marks are spaced out, if they are put into the group.	▪ `\so{inside.}␣\&␣\so{outside}.` ▪ i n s i d e . & o u t s i d e.	▪ i n - s i d e . & o u t - s i d e.
Space-out skips may be removed by typing `\<`. It's, however, desirable to put the quotation marks out of the argument.	▪ `\so{``\<Pennsylvania\<''}` ▪ “P e n n s y l v a n i a”	▪ “P e n n - s y l - v a - n i a”
Numbers should never be spaced out.	▪ `\so{1\<3␣December␣{1995}}` ▪ 13 D e c e m b e r 1995	▪ 13 D e - c e m - b e r 1995
Explicit hyphens like `-`*,* `--` *and* `---` *are recognized. `\slash` outputs a slash and enables TEX to break the line afterwards.*	▪ `\so{input\slash␣output}` ▪ i n p u t / o u t p u t	▪ i n - p u t / o u t - p u t
To keep TEX from breaking lines between the hyphen and 'jet' you have to protect the hyphen. This is no `soul-ori` *restriction but normal TEX behavior.*	▪ `\so{\dots␣and␣\mbox{-}jet}` ▪ . . . a n d - j e t	▪ . . . a n d - j e t

The ~ command inhibits line breaks.	▪ `\so{unbreakable~space}` ▪ unbreakable space	▪ un- break- able space
`\\` *works as usual. Additional arguments like* `*` *or vertical space are not accepted, though.*	▪ `\so{broken\\line}` ▪ broken line	▪ bro- ken line
`\break` *breaks the line without filling it with white space.*	▪ `\so{pretty␣awful\break␣test}` ▪ pretty awful test	▪ pretty aw- ful test

3.3 Typesetting capitals-and-small-capitals fonts

`\caps` There is a special letterspacing command called `\caps`, which differs from `\so` in that it switches to caps-and-small-caps font shape, defines only slight spacing and is able to select spacing value sets from a database. This is a requirement for high-quality typesetting [10]. The following lines show the effect of `\caps` in comparison with the normal textfont and with small-capitals shape:

`\normalfont`	DONAUDAMPFSCHIFFAHRTSGESELLSCHAFT
`\scshape`	DONAUDAMPFSCHIFFAHRTSGESELLSCHAFT
`\caps`	DONAUDAMPFSCHIFFAHRTSGESELLSCHAFT

The `\caps` font database is by default empty, i. e., it contains just a single default entry, which yields the result as shown in the example above. New font entries `\capsdef` may be added *on top* of this list using the `\capsdef` command, which takes five arguments: The first argument describes the font with *encoding, family, series, shape,* and *size,*[8] each optionally (e. g. `OT1/cmr/m/n/10` for this very font, or only `/ppl///12` for all *palatino* fonts at size 12 pt). The *size* entry may also contain a size range (`5-10`), where zero is assumed for an omitted lower boundary (`-10`) and a very, very big number for an omitted upper boundary (`5-`). The upper boundary is not included in the range, so, in the example below, all fonts with sizes greater or equal 5 pt and smaller than 15 pt are accepted (5 pt $\leq$ *size* $<$ 15 pt). The second argument may contain font switching commands such as `\scshape`, it may as well be empty or contain debugging commands (e. g. `\message{*}`). The remaining three, mandatory arguments are the spaces as described in section 3.1.

```
\capsdef{T1/ppl/m/n/5-15}{\scshape}{.16em}{.4em}{.2em}
```

The `\caps` command goes through the data list from top to bottom and picks up the first matching set, so the order of definition is essential. The last added entry is examined first, while the pre-defined default entry will be examined last and will match any font, if no entry was taken before.

To override the default values, just define a new default entry using the identifier `{////}`. This entry should be defined first, because no entry after it can be reached.

`\capsreset` The `\caps` database can be cleared with the `\capsreset` command and will `\capssave` only contain the default entry thereafter. The `\capssave` command saves the

[8] as defined by the NFSS, the "New Font Selection Scheme"

\capsselect whole current database under the given name. \capsselect restores such a database. This allows to predefine different groups of \caps data sets:

```
\capsreset
\capsdef{/cmss///12}{}{12pt}{23pt}{34pt}
\capsdef{/cmss///}{}{1em}{2em}{3em}
...
\capssave{wide}

%---------------------------------------
\capsreset
\capsdef{/cmss///}{}{.1em}{.2em}{.3em}
...
\capssave{narrow}

%---------------------------------------
{\capsselect{wide}
\title{\caps{Yet Another Silly Example}}
}
```

See the '`example.cfg`' file for a detailed example. If you have defined a bunch of sets for different fonts and sizes, you may lose control over what fonts are used
capsdefault by the package. With the package option capsdefault selected, \caps prints its argument underlined, if no set was specified for a particular font and the default set had to be used.

3.4 Typesetting Fraktur

Black letter fonts[9] deserve some additional considerations. As stated in section 1, the ligatures `ch`, `ck`, `sz` (\ss), and `tz` have to remain unbroken in spaced out *Fraktur* text. This may look strange at first glance, but you'll get used to it:

```
\textfrak{\so{S{ch}u{tz}vorri{ch}tung}}
```

You already know that grouping keeps the `soul` mechanism from separating such ligatures. This is quite important for `s:`, `a*`, and `"a`. As hyphenation is stronger than grouping, especially the `sz` may cause an error, if hyphenation happens to occur between the letters `s` and `z`. (TEX hyphenates the German word `auszer` wrongly like `aus-zer` instead of like `au-szer`, because the German hyphenation patterns do, for good reason, not see `sz` as '`\ss`'.) In such cases you can protect tokens with the sequence e. g. `\mbox{sz}` or a properly defined command. The `\ss` command, which is defined by the `yfonts` package, and similar commands will suffice as well.

3.5 Dirty tricks

Narrow columns are hard to set, because they don't allow much spacing flexibility, hence long words often cause overfull boxes. A macro could use `\so` to insert stretchability between the single characters. Table 2 shows some text typeset with such a macro at the left side and under *plain* conditions at the right side, both with a width of 6 pc.

[9] See the great black letter fonts, which Yannis Haralambous kindly provided, and the `oldgerm` and `yfonts` package [9] as their LATEX interfaces.

Some magazines and newspapers prefer this kind of spacing because it reduces hyphenation problems to a minimum. Unfortunately, such paragraphs aren't especially beautiful.	Some magazines and newspapers prefer this kind of spacing because it reduces hyphenation problems to a minimum. Unfortunately, such paragraphs aren't especially beautiful.	Some magazines and newspapers prefer this kind of spacing because it reduces hyphenation problems to a minimum. Unfortunately, such paragraphs aren't especially beautiful.

Table 2: Ragged-right, magazine style (using `soul-ori`), and block-aligned in comparison. But, frankly, none of them is really acceptable. (Don't do this at home, children!)

4 Underlining

The underlining macros are my answer to Prof. KNUTH's exercise 18.26 from his TEXbook [5]. `:-)` Most of what is said about the macro `\ul` is also true of the striking out macro `\st` and the highlighting macro `\hl`, both of which are in fact derived from the former.

`\ul` `\st` `\hl`

4.1 Settings

4.1.1 Underline depth and thickness

The predefined *underline depth* and *thickness* work well with most fonts. They can be changed using the macro `\setul`.

`\setul`

`\setul{`⟨*underline depth*⟩`}{`⟨*underline thickness*⟩`}`

Either dimension can be omitted, in which case there has to be an empty pair of braces. Both values should be defined in terms of `ex`, letting them grow and shrink with the respective fonts. The `\resetul` command restores the standard values.

`\resetul`

`\setuldepth` Another way to set the *underline depth* is to use the macro `\setuldepth`. It sets the depth such that the underline's upper edge lies 1 pt beneath the given argument's deepest depth. If the argument is empty, all letters—i. e. all characters whose `\catcode` currently equals 11—are taken. Examples:

```
\setuldepth{ygp}
\setuldepth\strut
\setuldepth{}
```

4.1.2 Line color

The underlines are by default black. The color can be changed by using the `\setulcolor` command. It takes one argument that can be any of the color spec-

`\setulcolor`

ifiers as described in the `color` package. This package has to be loaded explicitly.

```
\documentclass{article}
\usepackage{color,soul}
\definecolor{darkblue}{rgb}{0,0,0.5}
\setulcolor{darkblue}

\begin{document}
...
\ul{Cave: remove all the underlines!}
...
\end{document}
```

`\setstcolor` `\sethlcolor` The colors for overstriking lines and highlighting are likewise set with `\setstcolor` (default: black) and `\sethlcolor` (default: yellow). If the `color` package wasn't loaded, underlining and overstriking color are black, while highlighting is replaced by underlining.

4.1.3 The `dvips` problem

Underlining, ~~striking out~~ and highlighting build up their lines with many short line segments. If you used the '`dvips`' program with default settings, you would get little gaps on some places, because the *maxdrift* parameter allows the single objects to drift this many pixels from their real positions.

There are two ways to avoid the problem, where the `soul-ori` package chooses the second by default:

1. Set the *maxdrift* value to zero, e. g.: `dvips -e 0 file.dvi`. This is probably not a good idea, since the letters may then no longer be spaced equally on low resolution printers.

2. `\setuloverlap` Let the lines stick out by a certain amount on each side so that they overlap. This overlap amount can be set using the `\setuloverlap` command. It is set to 0.25 pt by default. `\setuloverlap{0pt}` turns overlapping off.

4.2 Some examples

Ordinary text.	▪ `\ul{electrical␣industry}` ▪ electrical industry	▪ elec- tri- cal in- dus- try
Use `\-` *to mark hyphenation points.*	▪ `\ul{man\-u\-script}` ▪ manuscript	▪ man- u- script
Accents are recognized.	▪ `\ul{le␣th\'e\^atre}` ▪ le théâtre	▪ le théâtre

`\mbox` and `\hbox` protect material that contains hyphenation points. The contents are treated as one, unbreakable entity.	▪ `\ul{just␣an␣\mbox{example}}` ▪ just an example	▪ just an example
Explicit hyphens like `-`*,* `--` *and* `---` *are recognized.* `\slash` *outputs a slash and enables TEX to break the line afterwards.*	▪ `\ul{input\slash␣output}` ▪ input/output	▪ in- put/ out- put
To keep TEX from breaking lines between the hyphen and 'jet' you have to protect the hyphen. This is no `soul-ori` *restriction but normal TEX behavior.*	▪ `\ul{\dots␣and␣\mbox{-}jet}` ▪ . . . and -jet	▪ . . . and -jet
The `~` *command inhibits line breaks.*	▪ `\ul{unbreakable~space}` ▪ unbreakable space	▪ un- break- able space
`\\` *works as usual. Additional arguments like* `*` *or vertical space are not accepted, though.*	▪ `\ul{broken\\line}` ▪ broken line	▪ bro- ken line
`\break` breaks the line without filling it with white space.	▪ `\ul{pretty␣awful\break␣test}` ▪ pretty awful test	▪ pretty aw- ful test
Italic correction needs to be set manually.	▪ `\ul{foo␣\emph{bar\/}␣baz}` ▪ foo *bar* baz	▪ foo *bar* baz

5 Customization

5.1 Adding accents

The `soul-ori` scanner generally sees every input token separately. It has to be taught that some tokens belong together. For accents this is done by registering
`\soulaccent` them via the `\soulaccent` macro.

`\soulaccent{`⟨*accent command*⟩`}`

The standard accents, however, are already pre-registered: `\‘`, `\’`, `\^`, `\"`, `\~`, `\=`, `\.`, `\u`, `\v`, `\H`, `\t`, `\c`, `\d`, `\b`, and `\r`. If used together with the `german` package, `soul-ori` automatically adds the `"` command. Let's assume you have defined `\%`

to put some weird accent on the next character. Simply put the following line into your `soul.cfg` file (see section 5.4):

```
\soulaccent{\%}
```

Note that active characters like the `"` command have already to be `\active` when they are stored or they won't be recognized later. This can be done temporarily, as in `{\catcode\‘"\active\soulaccent{"}}`.

5.2 Adding font commands

To convince `soul-ori` not to feed font switching (or other) commands to the analyzer, but rather to execute them immediately, they have to be registered, too.

`\soulregister` The `\soulregister` macro takes the name of a command name and either `0` or `1` for the number of arguments:

`\soulregister{`⟨*command name*⟩`}{`⟨*number of arguments*⟩`}`

If `\bf` and `\emph` weren't already registered, you would write the following into your `soul.cfg` configuration file:

```
\soulregister{\bf}{0}        % {\bf foo}
\soulregister{\emph}{1}      % \emph{bar}
```

All standard TEX and LATEX font commands, as well as the `yfonts` commands are already pre-registered:

```
\em, \rm, \bf, \it, \tt, \sc, \sl, \sf, \emph, \textrm,
\textsf, \texttt, \textmd, \textbf, \textup, \textsl,
\textit, \textsc, \textnormal, \rmfamily, \sffamily,
\ttfamily, \mdseries, \upshape, \slshape, \itshape,
\scshape, \normalfont, \tiny, \scriptsize, \footnotesize,
\small, \normalsize, \large, \Large, \LARGE, \huge, \Huge,
\MakeUppercase, \textsuperscript, \footnote,
\textfrak, \textswab, \textgoth, \frakfamily,
\swabfamily, \gothfamily
```

You can also register other commands as fonts, so the analyzer won't see them. This may be necessary for some macros that `soul-ori` refuses to typeset correctly. But note, that `\so` and `\caps` won't put their letter-skips around then.

5.3 Changing the internal font

The `soul-ori` package uses the `ectt1000` font while it analyzes the syllables. This font is used, because it has 256 mono-spaced characters without any kerning. It belongs to Jörg Knappen's EC-fonts, which should be part of every modern TEX installation. If TEX reports "`I can't find file ‘ectt1000’`" you don't seem to have this font installed. It is recommended that you install at least the file `ectt1000.tfm` which has less than 1.4 kB. Alternatively, you can let the `soul-ori` package use the `cmtt10` font that is part of any installation, or some other mono-spaced font:

```
\font\SOUL@tt=cmtt10
```

Note, however, that `soul-ori` does only handle characters, for which the internal font has a character with the same character code. As `cmtt10` contains only characters with codes 0 to 127, you can't typeset characters with codes 128 to 255. These 8-bit character codes are used by many fonts with non-ascii glyphs. So the `cmtt10` font will, for example, not work for `T2A` encoded cyrillic characters.

5.4 The configuration file

If you want to change the predefined settings or add new features, then create a file named '`soul.cfg`' and put it in a directory, where TEX can find it. This configuration file will then be loaded at the end of the `soul.sty` file, so you may redefine any settings or commands therein, select package options and even introduce new ones. But if you intend to give your documents to others, don't forget to give them the required configuration files, too! That's how such a file could look like:

```
% define macros for logical markup
\sodef\person{\scshape}{0.125em}{0.4583em}{0.5833em}

\sodef\SOUL@@@versal{\upshape}{0.125em}{0.4583em}{0.5833em}
\DeclareRobustCommand*\versal[1]{%
    \MakeUppercase{\SOUL@@@versal{#1}}%
}

% load the color package and set
% a different highlighting color
\RequirePackage{color}
\definecolor{lightblue}{rgb}{.90,.95,1}
\sethlcolor{lightblue}
\endinput
```

You can safely use the `\SOUL@@@` namespace for internal macros—it won't be used by the `soul-ori` package in the future.

6 Miscellaneous

6.1 Using `soul-ori` with other flavors of TEX

This documentation describes how to use `soul-ori` together with LATEX 2_ε, for which it is optimized. It works, however, with all other flavors of TEX, too. There are just some minor restrictions for Non-LATEX use:

The `\caps` command doesn't use a database, it is only a dumb definition with fixed values. It switches to `\capsfont`, which—unless defined explicitly like in the following example—won't really change the used font at all. The commands `\capsreset` and `\capssave` do nothing.

```
\font\capsfont=cmcsc10
\caps{Tschichold}
```

None of the commands are made 'robust', so they have to be explicitly protected in fragile environments like in `\write` statements. To make use of colored underlines or highlighting you have to use the `color` package wrapper from CTAN[10], instead of the `color` package directly:

```
\input color
\input soul.sty
\hl{highlighted}
\bye
```

`\capsdefault` The capsdefault package option is mapped to a simple command `\capsdefault`.

6.2 Using `soul-ori` commands for logical markup

It's generally a bad idea to use font style commands like `\textsc` in running text. There should always be some reasoning behind changing the style, such as "names of persons shall be typeset in a caps-and-small-caps font". So you declare in your text just that some words are the name of a person, while you define in the preamble or, even better, in a separate style file how to deal with persons:

```
\newcommand*\person{\textsc}
...
``I think it's a beautiful day to go to the zoo and feed
the ducks. To the lions.'' --~\person{Brian Kantor}
```

It's quite simple to use `soul-ori` commands that way:

```
\newcommand\comment*{\ul}             % or \let\comment=\ul
\sodef\person{\scshape}{0.125em}{0.4583em}{0.5833em}
```

Letterspacing commands like `\so` and `\caps` have to check whether they are followed by white space, in which case they replace that space by *outer space*. Note that `soul-ori` does look through closing braces. Hence you can conveniently bury a `soul-ori` command within another macro like in the following example. Use any other token to hide following space if necessary, for example the `\null` macro.

```
\DeclareRobustCommand*\versal[1]{%
    \MakeUppercase{\SOUL@@@versal{#1}}%
}
\sodef\SOUL@@@versal{\upshape}{0.125em}{0.4583em}{0.5833em}
```

But what if the `soul-ori` command is for some reason not the last one in that macro definition and thus cannot look ahead at the following token?

```
\newcommand*\somsg[1]{\so{#1}\message{#1}}
...
foo \somsg{bar} baz       % wrong spacing after 'bar'!
```

In this case you won't get the following space replaced by *outer space* because when `soul-ori` tries to look ahead, it only sees the token `\message` and consequently decides that there is no space to replace. You can get around this by explicitly calling the space scanner again.

[10] `CTAN:/macros/plain/graphics/{miniltx.tex,color.tex}`

```
\newcommand*\somsg[1]{{%
    \so{#1}%
    \message{bar}%
    \let\\\SOUL@socheck
    \\%
}}
```

However, `\SOUL@socheck` can't be used directly, because it would discard any normal space. `\\` doesn't have this problem. The additional pair of braces avoids that its definition leaks out of this macro. In the example above you could, of course, simply have put `\message` in front, so you hadn't needed to use the scanner macro `\SOUL@socheck` at all.

Many packages do already offer logical markup commands that default to some standard LaTeX font commands or to `\relax`. One example is the `jurabib` package [1], which makes the use of `soul-ori` a challenge. This package implements lots of formatting macros. Let's have a look at one of them, `\jbauthorfont`, which is used to typeset author names in citations. The attempt to simply define `\let\jbauthorfont\caps` fails, because the macro isn't directly applied to the author name as in `\jbauthorfont{Don Knuth}`, but to another command sequence: `\jbauthorfont{\jb@@author}`. Not even `\jb@@author` contains the name, but instead further commands that at last yield the requested name. That's why we have to expand the contents first. This is quite tricky, because we must not expand too much, either. Fortunately, we can offer the contents wrapped up in yet another macro, so that `soul-ori` knows that it has to use its own macro expansion mechanism:

```
\renewcommand*\jbauthorfont[1]{{%
    \def\x{#1}%
    \caps\x
}}
```

Some additional kerning after `\caps\x` wouldn't hurt, because the look-ahead scanner is blinded by further commands that follow in the `jurabib` package. Now we run into the next problem: cited names may contain commands that must not get expanded. We have to register them as special command:

```
\soulregister\jbbtasep{0}
...
```

But such registered commands bypass `soul-ori`'s kernel and we don't get the correct spacing before and afterwards. So we end up redefining `\jbbtasep`, whereby you should, of course, use variables instead of numbers:

```
\renewcommand*\jbbtasep{%
    \kern.06em
    \slash
    \hskip.06em
    \allowbreak
}
```

Another problem arises: bibliography entries that must not get teared apart are supposed to be enclosed in additional braces. This, however, won't work with `soul-ori` because of § 20. A simple trick will get you around that problem: define a dummy command that only outputs its argument, and register that command:

```
\newcommand*\together[1]{#1}
\soulregister\together{1}
```

Now you can write "`Author = {\together{Don Knuth}}`" and `jurabib` won't dare to reorder the parts of the name. And what if some name shouldn't get letterspaced at all? Overriding a conventional font style like `\textbf` that was globally set is trivial, you just have to specify the style that you prefer in that very bibliography entry. In our example, if we wanted to keep `soul-ori` from letterspacing a particular entry, although they are all formatted by our `\jbauthorfont` and hence fed to `\caps`, we'd use the following construction:

```
Author = {\soulomit{\normalfont\huge Donald E. Knuth}}
```

The `jurabib` package is probably one of the more demanding packages to collaborate with `soul-ori`. Everything else can just become easier.

6.3 Typesetting long words in narrow columns

Narrow columns are best set `flushleft`, because not even the best hyphenation algorithm can guarantee acceptable line breaks without overly stretched spaces. However, in some rare cases one may be *forced* to typeset block aligned. When typesetting in languages like German, where there are really long words, the `\sloppyword` macro might help a little bit. It adds enough stretchability between the single characters to make the hyphenation algorithm happy, but is still not as ugly as the example in section 3.5 demonstrates. In the following example the left column was typeset as "`Die \sloppyword{Donau...novelle} wird ...`":

\sloppyword

Die Donaudampfschiff-
fahrtsgesellschaftska-
pitänswitwenpensions-
gesetznovelle wird mit
sofortiger Wirkung außer
Kraft gesetzt.

Die Donaudampfschiff-
fahrtsgesellschaftska-
pitänswitwenpensions-
gesetznovelle wird mit
sofortiger Wirkung außer
Kraft gesetzt.

6.4 Using `soul-ori` commands in section headings

Letterspacing was often used for section titles in the past, mostly centered and with a closing period. The following example shows how to achieve this using the `titlesec` package [2]:

```
\newcommand*\periodafter[2]{#1{#2}.}
\titleformat{\section}[block]
    {\normalfont\centering}
    {\thesection.}
    {.66em}
    {\periodafter\so}
...
\section{Von den Maassen und Maassst\"aben}
```

This yields the following output:

1. Von den Maassen und Maassstäben.

The `\periodafter` macro adds a period to the title, but not to the entry in the table of contents. It takes the name of a command as argument, that shall be applied to the title, for example `\so`. Here's a more complicated and complete example:

```
\documentclass{article}
\usepackage[latin1]{inputenc}
\usepackage[T1]{fontenc}
\usepackage{german,soul}
\usepackage[indentfirst]{titlesec}

\newcommand*\sectitle[1]{%
    \MakeUppercase{\so{#1}.}\\[.66ex]
    \rule{13mm}{.4pt}}
\newcommand*\periodafter[2]{#1{#2.}}

\titleformat{\section}[display]
    {\normalfont\centering}
    {\S. \thesection.}
    {2ex}
    {\sectitle}

\titleformat{\subsection}[block]
    {\normalfont\centering\bfseries}
    {\thesection.}
    {.66em}
    {\periodafter\relax}

\begin{document}
\section{Von den Maassen und Maassst\"aben}
\subsection{Das L\"angenmaass im Allgemeinen}

Um L\"angen genau messen und vergleichen zu k\"onnen,
bedarf es einer gewissen, bestimmten Einheit, mit der
man untersucht, wie oft sie selbst, oder ihre Theile,
in der zu bestimmenden L\"ange enthalten sind.
...
\end{document}
```

This example gives you roughly the following output, which is a facsimile from [6].

§. 1.

VON DEN MAASSEN UND MAASSSTÄBEN.

1. Das Längenmaass im Allgemeinen.

Um Längen genau messen und vergleichen zu können, bedarf es einer gewissen, bestimmten Einheit, mit der man untersucht, wie oft sie selbst, oder ihre Theile, in der zu bestimmenden Länge enthalten sind.

Note that the definition of `\periodafter` decides if the closing period shall be spaced out with the title (1), or follow without space (2):

1. `\newcommand*\periodafter[2]{#1{#2.}}`
2. `\newcommand*\periodafter[2]{#1{#2}.}`

If you need to underline section titles, you can easily do it with the help of the `titlesec` package. The following example underlines the section title, but not the section number:

```
\titleformat{\section}
    {\LARGE\titlefont}
    {\thesection}
    {.66em}
    {\ul}
```

The `\titlefont` command is provided by the "KOMA script" package. You can write `\normalfont\sffamily\bfseries` instead. The following example does additionally underline the section number:

```
\titleformat{\section}
    {\LARGE\titlefont}
    {\ul{\thesection{\kern.66em}}}
    {0pt}
    {\ul}
```

7 How the package works

7.1 The kernel

L e t t e r s p a c i n g , underlining, ~~striking out~~ and highlighting use the same kernel. It lets a *word scanner* run over the given argument, which inspects every token. If a token is a command registered via `\soulregister`, it is executed immediately. Other tokens are only counted and trigger some action when a certain number is reached (quotes and dashes). Three subsequent '`-`', for example, trigger `\SOUL@everyexhyphen{---}`. A third group leads to special actions, like `\mbox` that starts reading-in a whole group to protect its contents and let them be seen as one entity. All other tokens, mostly characters and digits, are collected in a word register, which is passed to the analyzer, whenever a whole word was read in.

The analyzer typesets the word in a 1 sp ($= \frac{1}{65536}$ pt) wide `\vbox`, hence encouraging TEX to break lines at every possible hyphenation point. It uses the mono-spaced `\SOUL@tt` font (`ectt1000`), so as to avoid any inter-character kerning. Now the `\vbox` is decomposed splitting off `\hbox` after `\hbox` from the bottom. All boxes, each of which contains one syllable, are pushed onto a stack, which is provided by TEX's grouping mechanism. When returning from the recursion, box after box is fetched from the stack, its width measured and fed to the "reconstructor".

This reconstruction macro (`\SOUL@dosyllable`) starts to read tokens from the just analyzed word until the given syllable width is obtained. This is repeated for each syllable. Every time the engine reaches a relevant state, the corresponding driver macro is executed and, if necessary, provided with some data. There is a

macro that is executed for each token, one for each syllable, one for each space etc.

The engine itself doesn't know how to letterspace or to underline. It just tells the selected driver about the structure of the given argument. There's a default driver (`\SOUL@setup`) that does only set the interface macros to a reasonable default state, but doesn't really do anything. Further drivers can safely inherit these settings and only need to redefine what they want to change.

7.2 The interface

7.2.1 The registers

The package offers eight interface macros that can be used to define the required actions. Some of the macros receive data as macro parameter or in special *token* or *dimen* registers. Here is a list of all available registers:

`\SOUL@token` — This token register contains the current token. It has to be used as `\the\SOUL@token`. The macro `\SOUL@gettoken` reads the next token into `\SOUL@token` and can be used in any interface macro. If you don't want to lose the old meaning, you have to save it explicitly. `\SOUL@puttoken` pushes the token back into the queue, without changing `\SOUL@token`. You can only put one token back, otherwise you get an error message.

`\SOUL@lasttoken` — This token register contains the last token.

`\SOUL@syllable` — This token register contains all tokens that were already collected for the current syllable. When used in `\SOUL@everysyllable`, it contains the *whole* syllable.

`\SOUL@charkern` — This dimen register contains the kerning value between the current and the next character. Since most character pairs don't require a kerning value to be applied and the output in the logfile shouldn't be cluttered with `\kern0pt` it is recommended to write `\SOUL@setkern\SOUL@charkern`, which sets kerning for non-zero values only.

`\SOUL@hyphkern` — This dimen register contains the kerning value between the current character and the hyphen character or, when used in `\SOUL@everyexhyphen`, the kerning between the last character and the explicit hyphen.

7.2.2 The interface macros

The following list describes each of the interface macros and which registers it can rely on. The mark between label and description will be used in section 7.2.3 to show when the macros are executed. The addition `#1` means that the macro takes one argument.

`\SOUL@preamble`	P	executed once at the beginning
`\SOUL@postamble`	E	executed once at the end
`\SOUL@everytoken`	T	executed after scanning a token; It gets that token in `\SOUL@token` and has to care for inserting the kerning value `\SOUL@charkern` between this and the next character. To look at the next character, execute `\SOUL@gettoken`, which replaces `\SOUL@token` by the next token. This token has to be put back into the queue using `\SOUL@puttoken`.
`\SOUL@everysyllable`	S	This macro is executed after scanning a whole syllable. It gets the syllable in `\SOUL@syllable`.
`\SOUL@everyhyphen`	$-$	This macro is executed at every implicit hyphenation point. It is responsible for setting the hyphen and will likely do this in a `\discretionary` statement. It has to care about the kerning values. The registers `\SOUL@lasttoken`, `\SOUL@syllable`, `\SOUL@charkern` and `\SOUL@hyphkern` contain useful information. Note that `\discretionary` inserts `\exhyphenpenalty` if the first part of the discretionary is empty, and `\hyphenpenalty` else.
`\SOUL@everyexhyphen#1`	$=$	This macro is executed at every explicit hyphenation point. The hyphen 'character' (one of hyphen, en-dash, em-dash or `\slash`) is passed as parameter `#1`. A minimal implementation would be `{#1\penalty\exhyphenpenalty}`. The kerning value between the last character and the hyphen is passed in `\SOUL@hyphkern`, that between the hyphen and the next character in `\SOUL@charkern`. The last syllable can be found in `\SOUL@syllable`, the last character in `\SOUL@lasttoken`.
`\SOUL@everyspace#1`	$\sqcup$	This macro is executed between every two words. It is responsible for setting the space. The engine submits a `\penalty` setting as parameter `#1` that should be put in front of the space. The macro should at least do `{#1\space}`. Further information can be found in `\SOUL@lasttoken` and `\SOUL@syllable`. Note that this macro does not care for the leading and trailing space. This is the job of `\SOUL@preamble` and `\SOUL@postamble`.

7.2.3 Some examples

The above list's middle column shows a mark that indicates in the following examples, when the respective macros are executed:

$^{P}\mathrm{w}^{T}\mathrm{o}^{T}\mathrm{r}^{T}\mathrm{d}^{TSE}$	`\SOUL@everytoken`^{T} is executed for every token. `\SOUL@everysyllable`^{S} is *additionally* executed

for every syllable. You will mostly just want to use either of them.

^{P}o^{T}n^{T}e^{TS}␣^{T}t^{T}w^{T}o^{TSE} The macro `\SOUL@everyspace` is executed at every space within the `soul-ori` argument. It has to take one argument, that can either be empty or contain a penalty, that should be applied to the space.

^{P}e^{T}x^{TS-}a^{T}m^{TS-}p^{T}l^{T}e^{TSE} The macro `\SOUL@everyhyphen` is executed at every possible implicit hyphenation point.

^{P}b^{T}e^{T}t^{T}a$^{TS=}$-^{T}t^{T}e^{T}s^{T}t^{TSE} Explicit hyphens trigger `\SOUL@everyexhyphen`.

It's only natural that these examples, too, were automatically typeset by the `soul` package using a special driver:

```
\DeclareRobustCommand*\an{%
    \def\SOUL@preamble{$^{^P}$}%
    \def\SOUL@everyspace##1{##1\texttt{\char`\ }}%
    \def\SOUL@postamble{$^{^E}$}%
    \def\SOUL@everyhyphen{$^{^-}$}%
    \def\SOUL@everyexhyphen##1{##1$^{^=}$}%
    \def\SOUL@everysyllable{$^{^S}$}%
    \def\SOUL@everytoken{\the\SOUL@token$^{^T}$}%
    \def\SOUL@everylowerthan{$^{^L}$}%
    \SOUL@}
```

7.3 A driver example

Let's define a `soul-ori` driver that allows to typeset text with a `\cdot` at every potential hyphenation point. The name of the macro shall be `\sy` (for *syllables*). Since the `soul-ori` mechanism is highly fragile, we use the LaTeX command `\DeclareRobustCommand`, so that the `\sy` macro can be used even in section headings etc. The `\SOUL@setup` macro sets all interface macros to reasonable default definitions. This could of course be done manually, too. As we won't make use of `\SOUL@everytoken` and `\SOUL@postamble` and both default to `\relax`, anyway, we don't have to define them here.

```
\DeclareRobustCommand*\sy{%
    \SOUL@setup
```

We only set `\lefthyphenmin` and `\righthyphenmin` to zero at the beginning. All changes are restored automatically, so there's nothing to do at the end.

```
    \def\SOUL@preamble{\lefthyphenmin=0 \righthyphenmin=0 }%
```

We only want simple spaces. Note that these are not provided by default! `\SOUL@everyspace` may get a penalty to be applied to that space, so we set it before.

```
    \def\SOUL@everyspace##1{##1\space}%
```

There's nothing to do for `\SOUL@everytoken`, we rather let `\SOUL@everysyllable` handle a whole syllable at once. This has the advantage, that we don't have to deal with kerning values, because TEX takes care of that.

```
\def\SOUL@everysyllable{\the\SOUL@syllable}%
```

The TEX primitive `\discretionary` takes three arguments: 1. pre-hyphen material 2. post-hyphen material, and 3. no-hyphenation material.

```
\def\SOUL@everyhyphen{%
    \discretionary{%
        \SOUL@setkern\SOUL@hyphkern
        \SOUL@sethyphenchar
    }{}{%
        \hbox{\kern1pt$\cdot$}%
    }%
}%
```

Explicit hyphens like dashes and slashes shall be set normally. We just have to care for kerning. The hyphen has to be put in a box, because, as `\hyphenchar`, it would yield its own, internal `\discretionary`. We need to set ours instead, though.

```
\def\SOUL@everyexhyphen##1{%
    \SOUL@setkern\SOUL@hyphkern
    \hbox{##1}%
    \discretionary{}{}{%
        \SOUL@setkern\SOUL@charkern
    }%
}%
```

Now that the interface macros are defined, we can start the scanner.

```
    \SOUL@
}
```

This lit·tle macro will hard·ly be good e·nough for lin·guists, al·though it us·es TEX's ex·cel·lent hy·phen·ation al·go·rithm, but it is at least a nice al·ter·na·tive to the `\showhyphens` *com·mand.*

Acknowledgements

A big thank you goes to Stefan Ulrich for his tips and bug reports during the development of versions 1.* and for his lessons on high quality typesetting. The `\caps` mechanism was very much influenced by his suggestions. Thanks to Alexander Shibakov and Frank Mittelbach, who sent me a couple of bug reports and feature requests, and finally encouraged me to (almost) completely rewrite `soul-ori`. Thorsten Manegold contributed a series of bug reports, helping to fix `soul-ori`'s macro expander and hence making it work together with the `jurabib` package. Thanks to Axel Reichert, Anshuman Pandey, and Peter Kreynin for detailed bug reports. Rowland McDonnel gave useful hints for how to improve the documentation, but I'm afraid he will still not be satisfied, and rightfully so. If only documentation writing weren't that boring. ;-)

References

[1] Berger, Jens: *The jurabib package.* CTAN-Archive, 2002, v0.52h.

[2] Bezos, Javier: *The titlesec and titletoc package.* CTAN-Archive, 1999, v2.1.

[3] Carlisle, D. P.: *The color package.* CTAN-Archive, 1997, v1.0d.

[4] Duden, Volume 1. *Die Rechtschreibung.* Bibliographisches Institut, Mannheim–Wien–Zürich, 1986, 19th edition.

[5] Knuth, Donald E.: *The TEXbook.* Addison–Wesley Publishing Company, Reading/Massachusetts, 1989, 16th edition.

[6] Muszynski, Carl and Přihoda, Eduard: *Die Terrainlehre in Verbindung mit der Darstellung, Beurtheilung und Beschreibung des Terrains vom militärischen Standpunkte.* L. W. Seidel & Sohn, Wien, 1872.

[7] Normalverordnungsblatt für das k. u. k. Heer. *Exercier-Reglement für die k. u. k. Cavallerie, I. Theil.* Wien, k. k. Hof- und Staatsdruckerei, 1898, 4th edition.

[8] Raichle, Bernd: *The german package.* CTAN-Archive, 1998, v2.5e.

[9] Schmidt, Walter: *Ein Makropaket für die gebrochenen Schriften.* CTAN-Archive, 1998, v1.2.

[10] Tschichold, Jan: *Ausgewählte Aufsätze über Fragen der Gestalt des Buches und der Typographie.* Birkhäuser, Basel, 1987, 2nd edition.

[11] Willberg, Hans Peter and Forssmann, Friedrich: *Lesetypographie.* H. Schmidt, Mainz, 1997.

8 The implementation

The package preamble

This piece of code makes sure that the package is only loaded once. While this is guaranteed by LaTeX, we have to do it manually for all other flavors of TeX.

```
1 ⟨*package⟩
2 \expandafter\ifx\csname SOUL@\endcsname\relax\else
3     \expandafter\endinput
4 \fi
```

Fake some of the LaTeX commands if we were loaded by another flavor of TeX. This might break some previously loaded packages, though, if e. g. `\mbox` was already in use. But we don't care ...

```
5 \ifx\documentclass\SOULundefined
6     \chardef\atcode=\catcode`@
7     \catcode`\@=11
8     \def\DeclareRobustCommand*{\def}
9     \let\newcommand\DeclareRobustCommand
```

```
    \def\DeclareOption#1#2{\expandafter\def\csname#1\endcsname{#2}}
    \def\PackageError#1#2#3{{%
        \newlinechar`^^J%
        \errorcontextlines\z@
        \edef\\{\errhelp{#3}}\\%
        \errmessage{Package #1 error: #2}%
    }}
    \def\@height{height}
    \def\@depth{depth}
    \def\@width{width}
    \def\@plus{plus}
    \def\@minus{minus}
    \font\SOUL@tt=ectt1000
    \let\@xobeysp\space
    \let\linebreak\break
    \let\mbox\hbox
```

`soul-ori` tries to be a good LaTeX citizen if used under LaTeX and declares itself properly. Most command sequences in the package are protected by the `SOUL@` namespace, all other macros are first defined to be empty. This will give us an error message *now* if one of those was already used by another package.

```
\else
    \NeedsTeXFormat{LaTeX2e}
    \ProvidesPackage{soul-ori}
        [2023-06-14 v3.1 letterspacing/underlining  (mf)]
    \newfont\SOUL@tt{ectt1000}
    \newcommand*\sodef{}
    \newcommand*\resetso{}
    \newcommand*\capsdef{}
    \newcommand*\capsfont{}
    \newcommand*\setulcolor{}
    \newcommand*\setuloverlap{}
    \newcommand*\setul{}
    \newcommand*\resetul{}
    \newcommand*\setuldepth{}
    \newcommand*\setstcolor{}
    \newcommand*\sethlcolor{}
    \newcommand*\so{}
    \newcommand*\ul{}
    \newcommand*\st{}
    \newcommand*\hl{}
    \newcommand*\caps{}
    \newcommand*\soulaccent{}
    \newcommand*\soulregister{}
    \newcommand*\soulfont{}
    \newcommand*\soulomit{}
\fi
```

Other packages wouldn't be happy if we reserved piles of `\newtoks` and `\newdimen`, so we try to get away with their `\...def` counterparts where possible. Local registers are always even, while global ones are odd—this is a TeX convention.

```
\newtoks\SOUL@word
\newtoks\SOUL@lasttoken
\newtoks\SOUL@syllable
```

```
\newtoks\SOUL@cmds
\newtoks\SOUL@buffer
\newtoks\SOUL@token
\newdimen\SOUL@syllgoal
\newdimen\SOUL@syllwidth
\newdimen\SOUL@charkern
\newdimen\SOUL@hyphkern
\newdimen\SOUL@dimen
\newdimen\SOUL@dimeni
\newcount\SOUL@minus
\newcount\SOUL@comma
\newcount\SOUL@apo
\newcount\SOUL@grave
\newskip\SOUL@spaceskip
\newif\ifSOUL@ignorespaces
```

\soulomit
\SOUL@ignorem
\SOUL@ignore
\SOUL@stopm
\SOUL@stop
\SOUL@relaxm
\SOUL@lowerthanm
\SOUL@hyphenhintm

These macros are used as markers. To be able to check for such a marker with \ifx we have also to create a macro that contains the marker. \SOUL@spc shall contain a normal space with a \catcode of 10.

```
\def\soulomit#1{#1}
\def\SOUL@stopM{\SOUL@stop}
\let\SOUL@stop\relax
\def\SOUL@lowerthan{}
\def\SOUL@lowerthanM{\<}
\def\SOUL@hyphenhintM{\-}
\def\SOUL@n*{\let\SOUL@spc= }\SOUL@n* %
```

8.1 The kernel

\SOUL@

This macro is the entry to soul-ori. Using it does only make sense after setting up a soul-ori driver. The next token after the soul-ori command will be assigned to \SOUL@@. This can be some text enclosed in braces, or the name of a macro that contains text.

```
\def\SOUL@{%
    \futurelet\SOUL@@\SOUL@expand
}
```

\SOUL@expand

If the first token after the soul-ori command was an opening brace we start scanning. Otherwise, if the first token was a macro name, we expand that macro and call \SOUL@ with its contents again. Unfortunately, we have to exclude some macros therein from expansion.

```
\def\SOUL@expand{%
    \ifcat\bgroup\noexpand\SOUL@@
        \let\SOUL@n\SOUL@start
    \else
        \bgroup
            \def\\##1##2{\def##2{\noexpand##2}}%
            \the\SOUL@cmds
            \SOUL@buffer={%
                \\\TeX\\\LaTeX\\\soulomit\\\mbox\\\hbox\\\textregistered
                \\\slash\\\textcircled\\\copyright\\\S\\\,\\\<\\\>\\~%
                \\\\%
```

```
            }%
            \def\\##1{\def##1{\noexpand##1}}%
            \the\SOUL@buffer
            \let\protect\noexpand
            \xdef\SOUL@n##1{\noexpand\SOUL@start{\SOUL@@}}%
        \egroup
    \fi
    \SOUL@n
}
\long\def\SOUL@start#1{{%
    \let\<\SOUL@lowerthan
    \let\>\empty
    \def\soulomit{\noexpand\soulomit}%
    \gdef\SOUL@eventuallyexhyphen##1{}%
    \let\SOUL@soeventuallyskip\relax
    \SOUL@spaceskip=\fontdimen\tw@\font\@plus\fontdimen\thr@@\font
        \@minus\fontdimen4\font
    \SOUL@ignorespacesfalse
    \leavevmode
    \SOUL@preamble
    \SOUL@lasttoken={}%
    \SOUL@word={}%
    \SOUL@minus\z@
    \SOUL@comma\z@
    \SOUL@apo\z@
    \SOUL@grave\z@
    \SOUL@do{#1}%
    \SOUL@postamble
}}
\long\def\SOUL@do#1{%
    \SOUL@scan#1\SOUL@stop
}
```

8.2 The scanner

`\SOUL@scan` This is the entry point for the scanner. It calls `\SOUL@eval` and will in turn be called by `\SOUL@eval` again for every new token to be scanned.

```
\def\SOUL@scan{%
    \futurelet\SOUL@@\SOUL@eval
}
```

`\SOUL@eval` And here it is: the scanner's heart. It cares for quotes and dashes ligatures and handles all commands that must not be fed to the analyzer.

```
\def\SOUL@eval{%
    \def\SOUL@n*##1{\SOUL@scan}%
    \if\noexpand\SOUL@@\SOUL@spc
    \else
        \SOUL@ignorespacesfalse
    \fi
    \ifnum\SOUL@minus=\thr@@
        \SOUL@flushminus
    \else\ifnum\SOUL@comma=\tw@
        \SOUL@flushcomma
```

```
\else\ifnum\SOUL@apo=\tw@
    \SOUL@flushapo
\else\ifnum\SOUL@grave=\tw@
    \SOUL@flushgrave
\fi\fi\fi\fi
\ifx\SOUL@@-\else\SOUL@flushminus\fi
\ifx\SOUL@@,\else\SOUL@flushcomma\fi
\ifx\SOUL@@'\else\SOUL@flushapo\fi
\ifx\SOUL@@`\else\SOUL@flushgrave\fi
\ifx\SOUL@@-%
    \advance\SOUL@minus\@ne
\else\ifx\SOUL@@,%
    \advance\SOUL@comma\@ne
\else\ifx\SOUL@@'%
    \advance\SOUL@apo\@ne
\else\ifx\SOUL@@`%
    \advance\SOUL@grave\@ne
\else
    \SOUL@flushminus
    \SOUL@flushcomma
    \SOUL@flushapo
    \SOUL@flushgrave
    \ifx\SOUL@@\SOUL@stop
        \def\SOUL@n*{%
            \SOUL@doword
            \SOUL@eventuallyexhyphen\null
        }%
    \else\ifx\SOUL@@\par
        \def\SOUL@n*\par{\par\leavevmode\SOUL@scan}%
    \else\if\noexpand\SOUL@@\SOUL@spc
        \SOUL@doword
        \SOUL@eventuallyexhyphen\null
        \ifSOUL@ignorespaces
        \else
            \SOUL@everyspace{}%
        \fi
        \def\SOUL@n* {\SOUL@scan}%
    \else\ifx\SOUL@@\\%
        \SOUL@doword
        \SOUL@eventuallyexhyphen\null
        \SOUL@everyspace{\unskip\nobreak\hfil\break}%
        \SOUL@ignorespacestrue
    \else\ifx\SOUL@@~%
        \SOUL@doword
        \SOUL@eventuallyexhyphen\null
        \SOUL@everyspace{\nobreak}%
    \else\ifx\SOUL@@\slash
        \SOUL@doword
        \SOUL@eventuallyexhyphen{/}%
        \SOUL@exhyphen{/}%
    \else\ifx\SOUL@@\mbox
        \def\SOUL@n*{\SOUL@addprotect}%
    \else\ifx\SOUL@@\hbox
        \def\SOUL@n*{\SOUL@addprotect}%
```

```
        \else\ifx\SOUL@@\soulomit
            \def\SOUL@n*\soulomit##1{%
                \SOUL@doword
                {\spaceskip\SOUL@spaceskip##1}%
                \SOUL@scan
            }%
        \else\ifx\SOUL@@\break
            \SOUL@doword
            \break
        \else\ifx\SOUL@@\linebreak
            \SOUL@doword
            \SOUL@everyspace{\linebreak}%
        \else\ifcat\bgroup\noexpand\SOUL@@
            \def\SOUL@n*{\SOUL@addgroup{}}%
        \else\ifcat$\noexpand\SOUL@@
            \def\SOUL@n*{\SOUL@addmath}%
        \else
            \def\SOUL@n*{\SOUL@dotoken}%
        \fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
    \fi\fi\fi\fi
    \SOUL@n*%
}
```

\SOUL@flushminus
\SOUL@flushcomma
\SOUL@flushapo
\SOUL@flushgrave

As their names imply, these macros flush special tokens or token groups to the word register. They don't do anything if the respective counter equals zero. \SOUL@minus does also flush the word register, because hyphens disturb the analyzer.

```
\def\SOUL@flushminus{%
    \ifcase\SOUL@minus
    \else
        \SOUL@doword
        \SOUL@eventuallyexhyphen{-}%
        \ifcase\SOUL@minus
        \or
            \SOUL@exhyphen{-}%
        \or
            \SOUL@exhyphen{--}%
        \or
            \SOUL@exhyphen{---}%
        \fi
        \SOUL@minus\z@
    \fi
}
\def\SOUL@flushcomma{%
    \ifcase\SOUL@comma
    \or
        \edef\x{\SOUL@word={\the\SOUL@word,}}\x
    \or
        \edef\x{\SOUL@word={\the\SOUL@word{{,,}}}}\x
    \fi
    \SOUL@comma\z@
}
\def\SOUL@flushapo{%
    \ifcase\SOUL@apo
```

```
    \or
        \edef\x{\SOUL@word={\the\SOUL@word'}}\x
    \or
        \edef\x{\SOUL@word={\the\SOUL@word{{'}}}}\x
    \fi
    \SOUL@apo\z@
}
\def\SOUL@flushgrave{%
    \ifcase\SOUL@grave
    \or
        \edef\x{\SOUL@word={\the\SOUL@word`}}\x
    \or
        \edef\x{\SOUL@word={\the\SOUL@word{{``}}}}\x
    \fi
    \SOUL@grave\z@
}
```

`\SOUL@dotoken` Command sequences from the `\SOUL@cmds` list are handed over to `\SOUL@docmd`, everything else is added to `\SOUL@word`, which will be fed to the analyzer every time a word is completed. Since *robust* commands come with an additional space, we have also to examine if there's a space variant. Otherwise we couldn't detect pre-expanded formerly robust commands.

```
\def\SOUL@dotoken#1{%
    \def\SOUL@@{\SOUL@addtoken{#1}}%
    \def\\##1##2{%
        \edef\SOUL@x{\string#1}%
        \edef\SOUL@n{\string##2}%
        \ifx\SOUL@x\SOUL@n
            \def\SOUL@@{\SOUL@docmd{##1}{#1}}%
        \else
            \edef\SOUL@n{\string##2\space}%
            \ifx\SOUL@x\SOUL@n
                \def\SOUL@@{\SOUL@docmd{##1}{#1}}%
            \fi
        \fi
    }%
    \the\SOUL@cmds
    \SOUL@@
}
```

`\SOUL@docmd` Here we deal with commands that were registered with `\soulregister` or `\soulaccent` or were already predefined in `\SOUL@cmds`. Commands with identifier `9` are accents that are put in a group with their argument. Identifier `8` is reserved for the `\footnote` command, and `7` for the `\textsuperscript` or similar commands. The others are mostly (but not necessarily) font switching commands, which may (`1`) or may not (`0`) take an argument. A registered command leads to the current word buffer being flushed to the analyzer, after which the command itself is executed.

Font switching commands which take an argument need special treatment: They need to increment the level counter, so that `\SOUL@eval` knows where to stop scanning. Furthermore the scanner has to be enabled to see the next token after the opening brace.

```
\def\SOUL@docmd#1#2{%
    \ifx9#1%
        \def\SOUL@@{\SOUL@addgroup{#2}}%
    \else\ifx8#1%
        \SOUL@doword
        \def\SOUL@@##1{%
            \SOUL@token={\footnotemark}%
            \SOUL@everytoken
            \SOUL@syllable={\footnotemark}%
            \SOUL@everysyllable
            \footnotetext{##1}%
            \SOUL@doword
            \SOUL@scan
        }%
    \else\ifx7#1%
        \SOUL@doword
        \def\SOUL@@##1{%
            \SOUL@token={#2{##1}}%
            \SOUL@everytoken
            \SOUL@syllable={#2{##1}}%
            \SOUL@everysyllable
            \SOUL@doword
            \SOUL@scan
        }%
    \else\ifx1#1%
        \SOUL@doword
        \def\SOUL@@##1{%
            #2{\protect\SOUL@do{##1}}%
            \SOUL@scan
        }%
    \else
        \SOUL@doword
        #2%
        \let\SOUL@@\SOUL@scan
    \fi\fi\fi\fi
    \SOUL@@
}
```

\SOUL@addgroup
\SOUL@addmath
\SOUL@addprotect
\SOUL@addtoken

The macro names say it all. Each of these macros adds some token to the word buffer `\SOUL@word`. Setting `\protect` is necessary to make things like `\so{{a\itshape b}}` work.

```
\def\SOUL@addgroup#1#2{%
    {\let\protect\noexpand
    \edef\x{\global\SOUL@word={\the\SOUL@word{{\noexpand#1#2}}}}\x}%
    \SOUL@scan
}
\def\SOUL@addmath$#1$#{%
    {\let\protect\noexpand
    \edef\x{\global\SOUL@word={\the\SOUL@word{{\hbox{$#1$}}}}}\x}%
    \SOUL@scan
}
\def\SOUL@addprotect#1#2{%
    {\let\protect\noexpand
    \edef\x{\global\SOUL@word={\the\SOUL@word{{\hbox{#2}}}}}\x}%
```

```
    \SOUL@scan
}
\def\SOUL@addtoken#1{%
    \edef\x{\SOUL@word={\the\SOUL@word\noexpand#1}}\x
    \SOUL@scan
}
```

`\SOUL@exhyphen` Dealing with explicit hyphens can't be done before we know the following character, because we need to know if a kerning value has to be inserted, hence we delay the `\SOUL@everyexhyphen` call. Unfortunately, the word scanner has no look-ahead mechanism.

```
\def\SOUL@exhyphen#1{%
    \SOUL@getkern{\the\SOUL@lasttoken}{\SOUL@hyphkern}{#1}%
    \gdef\SOUL@eventuallyexhyphen##1{%
        \SOUL@getkern{#1}{\SOUL@charkern}{##1}%
        \SOUL@everyexhyphen{#1}%
        \gdef\SOUL@eventuallyexhyphen####1{}%
    }%
}
```

`\SOUL@cmds` Here is a list of pre-registered commands that the analyzer cannot handle, so the scanner has to look after them. Every entry consists of a handle (`\\`), an identifier and the macro name. The class identifier can be `9` for accents, `8` for the `\footnote` command, `7` for the `\textsuperscript` command, `0` for commands without arguments and `1` for commands that take one argument. Commands with two or more arguments are not supported.

```
\SOUL@cmds={%
    \\9\`\\9\'\\9\^\\9\"\\9\~\\9\=\\9\.%
    \\9\u\\9\v\\9\H\\9\t\\9\c\\9\d\\9\b\\9\r
    \\1\emph\\1\textrm\\1\textsf\\1\texttt\\1\textmd\\1\textbf
    \\1\textup\\1\textsl\\1\textit\\1\textsc\\1\textnormal
    \\0\rmfamily\\0\sffamily\\0\ttfamily\\0\mdseries\\0\upshape
    \\0\slshape\\0\itshape\\0\scshape\\0\normalfont
    \\0\em\\0\rm\\0\bf\\0\it\\0\tt\\0\sc\\0\sl\\0\sf
    \\0\tiny\\0\scriptsize\\0\footnotesize\\0\small
    \\0\normalsize\\0\large\\0\Large\\0\LARGE\\0\huge\\0\Huge
    \\1\MakeUppercase\\7\textsuperscript\\8\footnote
    \\1\textfrak\\1\textswab\\1\textgoth
    \\0\frakfamily\\0\swabfamily\\0\gothfamily
}
```

`\soulregister` `\soulfont` `\soulaccent` Register a font switching command (or some other command) for the scanner. The first argument is the macro name, the second is the number of arguments (`0` or `1`). Example: `\soulregister{\bold}{0}`. `\soulaccent` has only one argument—the accent macro name. Example: `\soulaccent{\~}`. It is a shortcut for `\soulregister{\~}{9}`. The `\soulfont` command is a synonym for `\soulregister` and is kept for compatibility reasons.

```
\def\soulregister#1#2{{%
    \edef\x{\global\SOUL@cmds={\the\SOUL@cmds
        \noexpand\\#2\noexpand#1}}\x
}}
\def\soulaccent#1{\soulregister{#1}9}
\let\soulfont\soulregister
```

8.3 The analyzer

`\SOUL@doword` The only way to find out, where a given word can be broken into syllables, is to let TEX actually typeset the word under conditions that enforce every possible hyphenation. The result is a paragraph with one line for every syllable.

```
\def\SOUL@doword{%
    \edef\x{\the\SOUL@word}%
    \ifx\x\empty
    \else
        \SOUL@buffer={}%
        \setbox\z@\vbox{%
            \SOUL@tt
            \hyphenchar\font`\-
            \hfuzz\maxdimen
            \hbadness\@M
            \pretolerance\m@ne
            \tolerance\@M
            \leftskip\z@
            \rightskip\z@
            \hsize1sp
            \everypar{}%
            \parfillskip\z@\@plus1fil
            \hyphenpenalty-\@M
            \noindent
            \hskip\z@
            \relax
            \the\SOUL@word}%
        \let\SOUL@errmsg\SOUL@error
        \let\-\relax
        \count@\m@ne
        \SOUL@analyze
        \SOUL@word={}%
    \fi
}
```

We store the hyphen width of the `ectt1000` font, because we will need it in `\SOUL@doword`. (`ectt1000` is a mono-spaced font, so every other character would have worked, too.)

```
\setbox\z@\hbox{\SOUL@tt-}
\newdimen\SOUL@ttwidth
\SOUL@ttwidth\wd\z@
\def\SOUL@sethyphenchar{%
    \ifnum\hyphenchar\font=\m@ne
    \else
        \char\hyphenchar\font
    \fi
}
```

`\SOUL@analyze` This macro decomposes the box that `\SOUL@doword` has built. Because we have to start at the bottom, we put every syllable onto the stack and execute ourselves recursively. If there are no syllables left, we return from the recursion and pick syllable after syllable from the stack again—this time from top to bottom—and hand the syllable width `\SOUL@syllgoal` over to `\SOUL@dosyllable`. All but the

last syllable end with the hyphen character, hence we subtract the hyphen width accordingly. After processing a syllable we calculate the hyphen kern (i. e. the kerning amount between the last character and the hyphen). This might be needed by `\SOUL@everyhyphen`, which we call now.

```
\def\SOUL@analyze{{%
    \setbox\z@\vbox{%
        \unvcopy\z@
        \unskip
        \unpenalty
        \global\setbox\@ne=\lastbox}%
    \ifvoid\@ne
    \else
        \setbox\@ne\hbox{\unhbox\@ne}%
        \SOUL@syllgoal=\wd\@ne
        \advance\count@\@ne
        \SOUL@analyze
        \SOUL@syllwidth\z@
        \SOUL@syllable={}%
        \ifnum\count@>\z@
            \advance\SOUL@syllgoal-\SOUL@ttwidth
            \SOUL@dosyllable
            \SOUL@getkern{\the\SOUL@lasttoken}{\SOUL@hyphkern}%
                {\SOUL@sethyphenchar}%
            \SOUL@everyhyphen
        \else
            \SOUL@dosyllable
        \fi
    \fi
}}
```

`\SOUL@dosyllable` This macro typesets token after token from `\SOUL@word` until `\SOUL@syllwidth` has reached the requested width `\SOUL@syllgoal`. Furthermore the kerning values are prepared in case `\SOUL@everytoken` needs them. The `\<` command used by `\so` and `\caps` needs some special treatment: It has to be checked for, even before we can end a syllable.

```
\def\SOUL@dosyllable{%
    \SOUL@gettoken
    \SOUL@eventuallyexhyphen{\the\SOUL@token}%
    \edef\x{\the\SOUL@token}%
    \ifx\x\SOUL@hyphenhintM
        \let\SOUL@n\SOUL@dosyllable
    \else\ifx\x\SOUL@lowerthanM
        \SOUL@gettoken
        \SOUL@getkern{\the\SOUL@lasttoken}{\SOUL@charkern}
            {\the\SOUL@token}%
        \SOUL@everylowerthan
        \SOUL@puttoken
        \let\SOUL@n\SOUL@dosyllable
    \else\ifdim\SOUL@syllwidth=\SOUL@syllgoal
        \SOUL@everysyllable
        \SOUL@puttoken
        \let\SOUL@n\relax
    \else\ifx\x\SOUL@stopM
```

```
        \SOUL@errmsg
        \global\let\SOUL@errmsg\relax
        \let\SOUL@n\relax
    \else
        \setbox\tw@\hbox{\SOUL@tt\the\SOUL@token}%
        \advance\SOUL@syllwidth\wd\tw@
        \global\SOUL@lasttoken=\SOUL@token
        \SOUL@gettoken
        \SOUL@getkern{\the\SOUL@lasttoken}{\SOUL@charkern}
            {\the\SOUL@token}%
        \SOUL@puttoken
        \global\SOUL@token=\SOUL@lasttoken
        \SOUL@everytoken
        \edef\x{\SOUL@syllable={\the\SOUL@syllable\the\SOUL@token}}\x
        \let\SOUL@n\SOUL@dosyllable
    \fi\fi\fi\fi
    \SOUL@n
}
```

`\SOUL@gettoken` Provide the next token in `\SOUL@token`. If there's already one in the buffer, use that one first.

```
\def\SOUL@gettoken{%
    \edef\x{\the\SOUL@buffer}%
    \ifx\x\empty
        \SOUL@nexttoken
    \else
        \global\SOUL@token=\SOUL@buffer
        \global\SOUL@buffer={}%
    \fi
}
```

`\SOUL@puttoken` The possibility to put tokens back makes the scanner design much cleaner. There's only room for one token, though, so we issue an error message if `\SOUL@puttoken` is told to put a token back while the buffer is still in use. Note that `\SOUL@debug` is actually undefined. This won't hurt as it can only happen during driver design. No user will ever see this message.

```
\def\SOUL@puttoken{%
    \edef\x{\the\SOUL@buffer}%
    \ifx\x\empty
        \global\SOUL@buffer=\SOUL@token
        \global\SOUL@token={}%
    \else
        \SOUL@debug{puttoken called twice}%
    \fi
}
```

`\SOUL@nexttoken` `\SOUL@splittoken` If the word buffer `\SOUL@word` is empty, deliver a `\SOUL@stop`, otherwise take the next token.

```
\def\SOUL@nexttoken{%
    \edef\x{\the\SOUL@word}%
    \ifx\x\empty
        \SOUL@token={\SOUL@stop}%
    \else
```

```
        \expandafter\SOUL@splittoken\the\SOUL@word\SOUL@stop
    \fi
}
\def\SOUL@splittoken#1#2\SOUL@stop{%
    \global\SOUL@token={#1}%
    \global\SOUL@word={#2}%
}
```

`\SOUL@getkern` Assign the kerning value between the first and the third argument to the second, which has to be a `\dimen` register. `\SOUL@getkern{A}{\dimen0}{V}` will assign the kerning value between 'A' and 'V' to `\dimen0`.

```
\def\SOUL@getkern#1#2#3{%
    \setbox\tw@\hbox{#1#3}%
    #2\wd\tw@
    \setbox\tw@\hbox{#1\null#3}%
    \advance#2-\wd\tw@
}
```

`\SOUL@setkern` Set a kerning value if it doesn't equal 0 pt. Of course, we could also set a zero value, but that would needlessly clutter the logfile.

```
\def\SOUL@setkern#1{\ifdim#1=\z@\else\kern#1\fi}
```

`\SOUL@error` This error message will be shown once for every word that couldn't be reconstructed by `\SOUL@dosyllable`.

```
\def\SOUL@error{%
    \vrule\@height.8em\@depth.2em\@width1em
    \PackageError{soul}{Reconstruction failed}{%
        I came across hyphenatable material enclosed in group
        braces,^^Jwhich I can't handle. Either drop the braces or
        make the material^^Junbreakable using an \string\mbox\space
        (\string\hbox). Note that a space^^Jalso counts as possible
        hyphenation point. See page 4 of the manual.^^JI'm leaving
        a black square so that you can see where I am right now.%
    }%
}
```

`\SOUL@setup` This is a null driver, that will be used as the basis for other drivers. These have then to redefine only interface commands that shall differ from the default.

```
\def\SOUL@setup{%
    \let\SOUL@preamble\relax
    \let\SOUL@postamble\relax
    \let\SOUL@everytoken\relax
    \let\SOUL@everysyllable\relax
    \def\SOUL@everyspace##1{##1\space}%
    \let\SOUL@everyhyphen\relax
    \def\SOUL@everyexhyphen##1{##1}%
    \let\SOUL@everylowerthan\relax
}
\SOUL@setup
```

8.4 The letterspacing driver

`\SOUL@sosetletterskip` A handy helper macro that sets the inter-letter skip with a draconian `\penalty`.

```
\def\SOUL@sosetletterskip{\nobreak\hskip\SOUL@soletterskip}
```

`\SOUL@sopreamble` If letterspacing (`\so` or `\caps`) follows a white space, we replace it with our *outer space*. LaTeX uses `\hskip1sp` as marker in tabular entries, so we ignore tiny skips.

```
\def\SOUL@sopreamble{%
    \ifdim\lastskip>5sp
        \unskip
        \hskip\SOUL@soouterskip
    \fi
    \spaceskip\SOUL@soinnerskip
}
```

`\SOUL@sopostamble` Start the look-ahead scanner `\SOUL@socheck` outside the `\SOUL@` scope. That's why we make the *outer space* globally available in `\skip@`.

```
\def\SOUL@sopostamble{%
    \global\skip@=\SOUL@soouterskip
    \aftergroup\SOUL@socheck
}
```

`\SOUL@socheck` `\SOUL@sodoouter` Read the next token after the `soul-ori` command into `\SOUL@@` and examine it. If it's some kind of space, replace it with *outer space* and the appropriate penalty, else if it's a closing brace, continue scanning. If it is neither: do nothing.

```
\def\SOUL@socheck{%
    \futurelet\SOUL@@\SOUL@sodoouter
}
\def\SOUL@sodoouter{%
    \def\SOUL@n*##1{\hskip\skip@}%
    \ifcat\egroup\noexpand\SOUL@@
        \unkern
        \egroup
        \def\SOUL@n*{\afterassignment\SOUL@socheck\let\SOUL@x=}%
    \else\ifx\SOUL@spc\SOUL@@
        \def\SOUL@n* {\hskip\skip@}%
    \else\ifx~\SOUL@@
        \def\SOUL@n*~{\nobreak\hskip\skip@}%
    \else\ifx\ \SOUL@@
    \else\ifx\space\SOUL@@
    \else\ifx\@xobeysp\SOUL@@
    \else
        \def\SOUL@n*{}%
        \let\SOUL@@\relax
    \fi\fi\fi\fi\fi\fi
    \SOUL@n*%
}
```

`\SOUL@soeverytoken` Typeset the token and put an unbreakable inter-letter skip thereafter. If the token is `\<` then remove the last skip instead. Gets the character kerning value between the actual and the next token in `\SOUL@charkern`.

```
\def\SOUL@soeverytoken{%
```

```
    \edef\x{\the\SOUL@token}%
    \ifx\x\SOUL@lowerthanM
    \else
        \global\let\SOUL@soeventuallyskip\SOUL@sosetletterskip
        \the\SOUL@token
        \SOUL@gettoken
        \edef\x{\the\SOUL@token}%
        \ifx\x\SOUL@stopM
        \else
            \SOUL@setkern\SOUL@charkern
            \SOUL@sosetletterskip
            \SOUL@puttoken
        \fi
    \fi
}
```

`\SOUL@soeveryspace` This macro sets an *inner space*. The argument may contain penalties and is used for the ~ command. This construction was needed to make colored underlines work, without having to put any of the coloring commands into the core. `\kern\z@` prevents in subsequent `\so` commands that the second discards the *outer space* of the first. To remove the space simply use `\unkern\unskip`.

```
\def\SOUL@soeveryspace#1{#1\space\kern\z@}
```

`\SOUL@soeveryhyphen` Sets implicit hyphens. The kerning value between the current token and the hyphen character is passed in `\SOUL@hyphkern`.

```
\def\SOUL@soeveryhyphen{%
    \discretionary{%
        \unkern
        \SOUL@setkern\SOUL@hyphkern
        \SOUL@sethyphenchar
    }{}{}%
}
```

`\SOUL@soeveryexhyphen` Sets the explicit hyphen that is passed as argument. `\SOUL@soeventuallyskip` equals `\SOUL@sosetletterskip`, except when a `\<` had been detected. This is necessary because `\SOUL@soeveryexhyphen` wouldn't know otherwise, that it follows a `\<`.

```
\def\SOUL@soeveryexhyphen#1{%
    \SOUL@setkern\SOUL@hyphkern
    \SOUL@soeventuallyskip
    \hbox{#1}%
    \discretionary{}{}{%
        \SOUL@setkern\SOUL@charkern
    }%
    \SOUL@sosetletterskip
    \global\let\SOUL@soeventuallyskip\relax
}
```

`\SOUL@soeverylowerthan` Let `\<` remove the last inter-letter skip. Set the kerning value between the token before and that after the `\<` command.

```
\def\SOUL@soeverylowerthan{%
    \unskip
```

```
    \unpenalty
    \global\let\SOUL@soeventuallyskip\relax
    \SOUL@setkern\SOUL@charkern
}
```

`\SOUL@sosetup` Override all interface macros by our letterspacing versions. The only unused macro is `\SOUL@everysyllable`.

```
\def\SOUL@sosetup{%
    \SOUL@setup
    \let\SOUL@preamble\SOUL@sopreamble
    \let\SOUL@postamble\SOUL@sopostamble
    \let\SOUL@everytoken\SOUL@soeverytoken
    \let\SOUL@everyspace\SOUL@soeveryspace
    \let\SOUL@everyhyphen\SOUL@soeveryhyphen
    \let\SOUL@everyexhyphen\SOUL@soeveryexhyphen
    \let\SOUL@everylowerthan\SOUL@soeverylowerthan
}
```

`\SOUL@setso` A handy macro for internal use.

```
\def\SOUL@setso#1#2#3{%
    \def\SOUL@soletterskip{#1}%
    \def\SOUL@soinnerskip{#2}%
    \def\SOUL@sooouterskip{#3}%
}
```

`\sodef` This macro assigns the letterspacing skips as well as an optional font switching command to a command sequence name. `\so` itself will be defined using this macro.

```
\def\sodef#1#2#3#4#5{%
    \DeclareRobustCommand*#1{\SOUL@sosetup
        \def\SOUL@preamble{%
            \SOUL@setso{#3}{#4}{#5}%
            #2%
            \SOUL@sopreamble
        }%
        \SOUL@
    }%
}
```

`\resetso` Let `\resetso` define reasonable default values for letterspacing.

```
\def\resetso{%
    \sodef\textso{}{.25em}{.65em\@plus.08em\@minus.06em}%
        {.55em\@plus.275em\@minus.183em}%
}
\resetso
```

`\sloppyword` Set up a letterspacing macro that inserts slightly stretchable space between the characters. This can be used to typeset long words in narrow columns, where ragged paragraphs are undesirable. See section 6.3.

```
\sodef\sloppyword{%
    \linepenalty10
    \hyphenpenalty10
```

```
    \adjdemerits\z@
    \doublehyphendemerits\z@
    \finalhyphendemerits\z@
    \emergencystretch.1em}%
    {\z@\@plus.1em}%
    {.33em\@plus.11em\@minus.11em}%
    {.33em\@plus.11em\@minus.11em}
```

8.5 The caps driver

`\caps` Unless run under LaTeX, make `\caps` just another simple letterspacing macro that selects a font `\capsfont` (defaulting to `\relax`) but doesn't have any special capabilities.

```
\ifx\documentclass\@undefined
\let\capsfont\relax
\let\capsreset\relax
\def\capsdef#1#2#3#4#5{}
\def\capssave#1{}
\def\capsselect#1{}
\sodef\textcaps{\capsfont}
    {.028em\@plus.005em\@minus.01em}%
    {.37em\@plus.1667em\@minus.111em}%
    {.37em\@plus.1em\@minus.14em}
```

`\capsreset` ... else, if run under LaTeX prepare a set of macros that maintain a database with certain letterspacing values for different fonts. `\capsreset` clears the database and inserts a default rule.

```
\else
\DeclareRobustCommand*\capsreset{%
    \let\SOUL@capsbase\empty
    \SOUL@capsdefault
}
```

`\capsdef` Add an entry to the database, which is of course nothing else than a TeX macro. See section "List macros" of appendix D in the TeXbook [5] for details.

```
\def\capsdef#1#2#3#4#5{{%
    \toks\z@{\\{#1/#2/#3/#4/#5}}%
    \toks\tw@=\expandafter{\SOUL@capsbase}%
    \xdef\SOUL@capsbase{\the\toks\z@\the\toks\tw@}%
}}
```

`\capssave` `\capsselect` Save the current database in a macro within the `SOUL@` namespace and let `\capsselect` restore this database.

```
\DeclareRobustCommand*\capssave[1]{%
    \expandafter\global\expandafter\let
        \csname SOUL@db@#1\endcsname\SOUL@capsbase
}
\DeclareRobustCommand*\capsselect[1]{%
    \expandafter\let\expandafter\SOUL@capsbase
        \csname SOUL@db@#1\endcsname
}
```

\SOUL@capsfind
\SOUL@caps

Go through the database entries and pick the first entry that matches the currently active font. Then define an internal macro that uses the respective spacing values in a macro that is equivalent to the \textso command.

```
\def\SOUL@capsfind#1/#2/#3/#4/#5/#6/#7/#8/#9/{%
    \let\SOUL@match=1%
    \SOUL@chk{#1}\f@encoding
    \SOUL@chk{#2}\f@family
    \SOUL@chk{#3}\f@series
    \SOUL@chk{#4}\f@shape
    \SOUL@dimchk{#5}\f@size
    \if\SOUL@match1%
        \let\\\@gobble
        \gdef\SOUL@caps{%
            \SOUL@sosetup
            \def\SOUL@preamble{\SOUL@setso{#7}{#8}{#9}#6%
                \SOUL@sopreamble}%
            \SOUL@}%
    \fi
}
```

\SOUL@chk

Sets the \SOUL@match flag if both parameters are equal. This is used for all NFSS elements except the font size.

```
\def\SOUL@chk#1#2{%
    \if$#1$%
    \else
        \def\SOUL@n{#1}%
        \ifx#2\SOUL@n\else\let\SOUL@match=0\fi
    \fi
}
```

\SOUL@dimchk
\SOUL@rangechk

We do not only want to check if a given font size #1 matches #2, but also if it fits into a given range. An omitted lower boundary is replaced by \z@ and an omitted upper boundary by \maxdimen. The first of a series of \SOUL@chk and \SOUL@dimchk statements, which detects that the arguments don't match, sets the \SOUL@match flag to zero. A value of 1 indicates that an entry in the font database matches the currently used font.

```
\def\SOUL@dimchk#1#2{\if$#1$\else\SOUL@rangechk{#2}#1--\@ne\@@\fi}
\def\SOUL@rangechk#1#2-#3-#4\@@{%
    \count@=#4%
    \ifnum\count@>\z@
        \ifdim#1\p@=#2\p@\else\let\SOUL@match=0\fi
    \else
        \SOUL@dimen=\if$#2$\z@\else#2\p@\fi
        \ifdim#1\p@<\SOUL@dimen\let\SOUL@match=0\fi
        \SOUL@dimen=\if$#3$\maxdimen\else#3\p@\fi
        \ifdim#1\p@<\SOUL@dimen\else\let\SOUL@match=0\fi
    \fi
}
```

\textcaps

Find a matching entry in the database and start the letterspacing mechanism with the given spacing values.

```
\DeclareRobustCommand*\textcaps{{%
```

```
    \def\\##1{\expandafter\SOUL@capsfind##1/}%
    \SOUL@capsbase
    \aftergroup\SOUL@caps
}}
```

`\SOUL@capsdefault` Define a default database entry and a default font.

```
\def\SOUL@capsdefault{%
    \capsdef{////}%
    \SOUL@capsdfltfnt
    {.028em\@plus.005em\@minus.01em}%
    {.37em\@plus.1667em\@minus.1em}%
    {.37em\@plus.111em\@minus.14em}%
}
\let\SOUL@capsdfltfnt\scshape
\capsreset
\fi
```

8.6 The underlining driver

`\SOUL@ulleaders` This macro sets the underline under the following `\hskip`.

```
\newdimen\SOUL@uldp
\newdimen\SOUL@ulht
\def\SOUL@ulleaders{%
    \leaders\hrule\@depth\SOUL@uldp\@height\SOUL@ulht\relax
}
```

`\SOUL@ulunderline` Set an underline under the given material. It draws the line first, and the given material afterwards. This is needed for highlighting, but gives less than optimal results for colored overstriking, which, however, will hardly ever be used, anyway.

```
\def\SOUL@ulunderline#1{{%
    \setbox\z@\hbox{#1}%
    \SOUL@dimen=\wd\z@
    \SOUL@dimeni=\SOUL@uloverlap
    \advance\SOUL@dimen2\SOUL@dimeni
    \rlap{%
        \null
        \kern-\SOUL@dimeni
        \SOUL@ulcolor{\SOUL@ulleaders\hskip\SOUL@dimen\kern\z@}%
    }%
    \unhcopy\z@
}}
```

`\SOUL@ulpreamble` Just set up the line dimensions and the space skip. Normally, `\spaceskip` is unset and not used by TEX. We need it, though, because we feed it to the `\leaders` primitive.

```
\def\SOUL@ulpreamble{%
    \SOUL@uldp=\SOUL@uldepth
    \SOUL@ulht=-\SOUL@uldp
    \advance\SOUL@uldp\SOUL@ulthickness
    \spaceskip\SOUL@spaceskip
}
```

\SOUL@uleverysyllable By using \SOUL@everysyllable we don't have to care about kerning values and get better results for highlighting, where negative kerning values would otherwise cut off characters.

```
\def\SOUL@uleverysyllable{%
    \SOUL@ulunderline{%
        \the\SOUL@syllable
        \SOUL@setkern\SOUL@charkern
    }%
}
```

\SOUL@uleveryspace Set a given penalty and an underlined \space equivalent. The \null prevents a nasty gap in \textfrak {a \textswab{b}}, while it doesn't seem to hurt in all other cases. I didn't investigate this.

```
\def\SOUL@uleveryspace#1{%
    \SOUL@ulcolor{%
        #1%
        \SOUL@ulleaders
        \hskip\spaceskip
        \kern\z@
    }%
    \null
}
```

\SOUL@uleveryhyphen If hyphenation takes place, output an underlined hyphen with the required hyphen kerning value.

```
\def\SOUL@uleveryhyphen{%
    \discretionary{%
        \unkern
        \SOUL@ulunderline{%
            \SOUL@setkern\SOUL@hyphkern
            \SOUL@sethyphenchar
        }%
    }{}{}%
}
```

\SOUL@uleveryexhyphen Underline the given hyphen, en-dash, em-dash or \slash and care for kerning.

```
\def\SOUL@uleveryexhyphen#1{%
    \SOUL@setkern\SOUL@hyphkern
    \SOUL@ulunderline{#1}%
    \discretionary{}{}{%
        \SOUL@setkern\SOUL@charkern
    }%
}
```

\SOUL@ulcolor \setulcolor Define the underline color or turn off coloring, in which case the lines are not just colored black, but remain uncolored. This makes them appear black, nevertheless, and has the advantage, that no Postscript \specials are cluttering the output.

```
\let\SOUL@ulcolor\relax
\def\setulcolor#1{%
    \if$#1$
        \let\SOUL@ulcolor\relax
    \else
```

```
        \def\SOUL@ulcolor{\textcolor{#1}}%
    \fi
}
```

`\setuloverlap` `\SOUL@uloverlap` Set the overlap amount, that helps to avoid gaps on sloppy output devices.

```
\def\setuloverlap#1{\def\SOUL@uloverlap{#1}}
\setuloverlap{.25pt}
```

`\SOUL@ulsetup` The underlining driver is quite simple. No need for `\SOUL@postamble` and `\SOUL@everytoken`.

```
\def\SOUL@ulsetup{%
    \SOUL@setup
    \let\SOUL@preamble\SOUL@ulpreamble
    \let\SOUL@everysyllable\SOUL@uleverysyllable
    \let\SOUL@everyspace\SOUL@uleveryspace
    \let\SOUL@everyhyphen\SOUL@uleveryhyphen
    \let\SOUL@everyexhyphen\SOUL@uleveryexhyphen
}
```

`\SOUL@textul` Describing self-explanatory macros is *so* boring!

```
\DeclareRobustCommand*\textul{\SOUL@ulsetup\SOUL@}
```

`\setul` `\SOUL@uldepth` `\SOUL@ulthickness` Set the underlining dimensions. Either value may be omitted and lets the respective macro keep its current contents.

```
\def\setul#1#2{%
    \if$#1$\else\def\SOUL@uldepth{#1}\fi
    \if$#2$\else\def\SOUL@ulthickness{#2}\fi
}
```

`\resetul` Set reasonable default values that fit most latin fonts.

```
\def\resetul{\setul{.65ex}{.1ex}}
\resetul
```

`\setuldepth` This macro sets all designated "letters" (`\catcode=11`) or the given material in a box and sets the underlining dimensions according to the box depth.

```
\def\setuldepth#1{{%
    \def\SOUL@n{#1}%
    \setbox\z@\hbox{%
        \tracinglostchars\z@
        \ifx\SOUL@n\empty
            \count@\z@
            \loop
                \ifnum\catcode\count@=11\char\count@\fi
            \ifnum\count@<\@cclv
                \advance\count@\@ne
            \repeat
        \else
            #1%
        \fi
    }%
    \SOUL@dimen\dp\z@
    \advance\SOUL@dimen\p@
    \xdef\SOUL@uldepth{\the\SOUL@dimen}%
}}
```

8.7 The ~~overstriking~~ driver

\SOUL@stpreamble Striking out is just underlining with a raised line of a different color. Hence we only need to define the preamble accordingly and let the underlining preamble finally do its job. Not that colored overstriking was especially useful, but we want at least to keep it black while we might want to set underlines in some fancy color.

```
\def\SOUL@stpreamble{%
    \SOUL@dimen\SOUL@ulthickness
    \SOUL@dimeni=-.5ex
    \advance\SOUL@dimeni-.5\SOUL@dimen
    \edef\SOUL@uldepth{\the\SOUL@dimeni}%
    \let\SOUL@ulcolor\SOUL@stcolor
    \SOUL@ulpreamble
}
```

\SOUL@stsetup We re-use the whole underlining setup and just replace the preamble with our modified version.

```
\def\SOUL@stsetup{%
    \SOUL@ulsetup
    \let\SOUL@preamble\SOUL@stpreamble
}
```

\textst These pretzels are making me thirsty ...

```
\DeclareRobustCommand*\textst{\SOUL@stsetup\SOUL@}
```

\SOUL@stcolor \setstcolor Set the overstriking color. This won't be used often, but is required in cases, where the underlines are colored. You wouldn't want to overstrike in the same color. Note that overstriking lines are drawn *beneath* the text, hence bright colors won't look good.

```
\let\SOUL@stcolor\relax
\def\setstcolor#1{%
    \if$#1$
        \let\SOUL@stcolor\relax
    \else
        \def\SOUL@stcolor{\textcolor{#1}}%
    \fi
}
```

8.8 The highlighting driver

\SOUL@hlpreamble This is nothing else than overstriking with very thick lines.

```
\def\SOUL@hlpreamble{%
    \setul{}{2.5ex}%
    \let\SOUL@stcolor\SOUL@hlcolor
    \SOUL@stpreamble
}
```

\SOUL@hlsetup No need to re-invent the wheel. Just use the overstriking setup with a different preamble.

```
\def\SOUL@hlsetup{%
    \SOUL@stsetup
    \let\SOUL@preamble\SOUL@hlpreamble
}
```

\texthl
\sethlcolor
\SOUL@hlcolor

Define the highlighting macro and the color setting macro with a simple default color. Yellow isn't really pleasing, but it's already predefined by the `color` package.

```
\DeclareRobustCommand*\texthl{\SOUL@hlsetup\SOUL@}
\def\sethlcolor#1{\if$#1$\else\def\SOUL@hlcolor{\textcolor{#1}}\fi}
\sethlcolor{yellow}
```

The package postamble

\so
\ul
\st
\hl
\caps

OK, I lied. The short macro names are just abbreviations for their longer counterpart. Some people might be used to `\text*` style commands to select a certain font. And then it doesn't hurt to reserve these early enough.

```
\let\so\textso
\let\ul\textul
\let\st\textst
\let\hl\texthl
\let\caps\textcaps
```

When used in an environment other than LaTeX and the `german` package was already loaded, define the double quotes as accent.

```
\ifx\documentclass\@undefined
    \ifx\mdqoff\@undefined
    \else
        \soulaccent{"}%
    \fi
    \catcode`\@=\atcode
```

If we have been loaded by a LaTeX environment and the `color` package wasn't also loaded, we disable all colors. Then we add the umlaut accent `"` if the `german` package is present. The quotes character has to be `\catcode`'d `\active` now, or it won't get recognized later. The capsdefault option overrides the `\caps` driver and lets `\SOUL@` set an underline. Finally load the local configuration, process the `capsdefault` option and exit.

```
\else
    \bgroup
        \catcode`\"\active
        \AtBeginDocument{%
            \@ifundefined{color}{%
                \let\SOUL@color\relax
                \let\setulcolor\@gobble
                \let\setstcolor\@gobble
                \let\sethlcolor\@gobble
                \let\hl\ul
            }{}
            \@ifundefined{mdqoff}{}{\soulaccent{"}}
        }
    \egroup
    \DeclareOption{capsdefault}{%
        \AtBeginDocument{%
            \def\SOUL@capsdfltfnt#1{%
                \SOUL@ulsetup
                \SOUL@ulpreamble
                \scshape
```

```
            }%
        }%
    }
    \InputIfFileExists{soul.cfg}%
        {\PackageInfo{soul}{Local config file soul.cfg used}}{}
    \ProcessOptions
\fi
\endinput
```

⟨/package⟩

```
$Id$
```