fistrum

Access to 150 paragraphs of Lorem Fistrum very dummy text[ab].

David Davó[c]

28 de febrero de 2023

Resumen

fistrum es un paquete de LaTeX derivado de lipsum que produce texto de ejemplo para usarlo en documentos y ejemplos. Los párrafos se han tomado con permiso de https://www.chiquitoipsum.com/.

Por favor, si encuentras un bug, alguna errata o tienes alguna sugerencia abre un *issue* en https://github.com/daviddavo/fistrum.

[a] Version: 0.1
[b] Basado en Chiquito Ipsum de Isabel Nieto, Carlos A. Hernández y Gauss Multimedia
[c] david@ddavo.me

1. Al ataquerl

Nota: La documentación es una copia de la de lipsum, pueden haber quedado erratas

To load the package, write

```
\usepackage{fistrum}
```

in the preamble of your document. Probably the most important macro provided by this package is `\fistrum`, which typesets the *Lorem fistrum* paragraphs. The first optional argument allows to specify the range of the paragraphs. For example, `\fistrum[4-57]` typesets the paragraphs 4 to 57 and accordingly, `\fistrum[23]` typesets the 23rd paragraph. Using `\fistrum` without its optional argument typesets the paragraphs 1–7 of *Lorem fistrum...*

`\fistrum`

As of version 2.0, `\fistrum` has a second optional argument which allows selecting a range of sentences from the paragraphs. To get the sentences four to eight from paragraphs three to nine, use `\fistrum[3-9][4-8]`. The sentences are counted from the first sentence of the first selected paragraph. In the previous example, sentence number 1 is the first sentence of paragraph number 3.

2. Usage

fistrum was intended to quickly provide a way to fill a page or two to analyze the page layout[1]. While it has grown in the meanwhile and now provides some more advanced features, it still is only intended to quickly provide text. If you want more features, look at the blindtext-package.

2.1. Package Options

fistrum outputs a range of paragraphs taken from the *Lorem fistrum...* dummy text. The package options control mainly the behaviour of the `\fistrum` and `\unpackfistrum` commands, and can be set at load-time with `\usepackage[`⟨*option*⟩`]{fistrum}`, or later in the document by using `\setfistrum{`⟨*option*⟩`}`.

[1] https://groups.google.com/d/topic/de.comp.text.tex/oPeL0jkrLfk

nopar = ⟨*boolean*⟩ (default: `false`)
Changes the initial default separator between each paragraph of `\fistrum` from `\par` to `\space`, and the other way around for `\fistrum*`.

text = ⟨*name*⟩ (default: `fistrum-es`)
Selects the dummy text ⟨*name*⟩ that is used by `\fistrum` and `\unpackfistrum` (see section 4).

language = ⟨*lang*⟩ (default: `latin`)
Sets the language to be used by `\fistrum` to typeset the currently active dummy text (see section 3.2). Changing the dummy text with the text option will also change the current language.

auto-lang = ⟨*boolean*⟩ (default: `true`)
Turns on/off automatic language switching. This changed since version 2.3, in which this option (didn't exist thus) was `false` by default. See section 3.2 for more details.

default-range = ⟨p_i-p_f⟩ (default: `1-7`)
Sets the default range of paragraphs produced by `\fistrum` when no optional argument is provided. The value to default-range obeys the ⟨*range*⟩ syntax described in section 3.1. If no value is given to default-range (that is, `\setfistrum{default-range}`), then the default is reset to `1-7`.

Besides these options, there are still ones that can be passed to the package to influence the paragraph and sentence separators and other such things. These options are detailed in section 3.3.

2.2. User Commands

`\fistrum` `\fistrum⟨*⟩[⟨par range⟩][⟨sentence range⟩]`

`\fistrum` outputs the ⟨*par range*⟩ from the currently active dummy text. If ⟨*par range*⟩ is not given or is empty, the default-range (initially `1-7`) is output. If a ⟨*sentence range*⟩ is given, the selected paragraphs are split into sentences, numbered starting from 1, and the specified range of sentences is taken out from those paragraphs. If the ⟨***⟩ version is used, a different set of separators is inserted around the paragraphs or sentences.

`\fistrum` changes the active language to that of the dummy text for typesetting, so the proper hyphenation patterns are used. See section 3.2. Section 3.1 explains the syntax of ranges, and section 3.3 explains the separators added around the pieces of text.

`\unpackfistrum` `\unpackfistrum⟨*⟩[⟨par range⟩][⟨sentence range⟩]`
`\fistrumexp` `...`
`\fistrumexp`

`\unpackfistrum` selects the paragraphs and/or sentences exactly as described for `\fistrum`, but instead of outputting them, it saves the selected text in the `\fistrumexp` macro. Additionally, `\unpackfistrum` ... `\fistrumexp` is not completely equivalent to `\fistrum` because it doesn't change languages as `\fistrum` does.

`\setfistrum` `\setfistrum{⟨key-val list⟩}`

Applies the ⟨*key-val list*⟩ of options to the package. The options are described in section 2.1 and in section 3.3.

2.3. Other commands

These commands exist for necessity or backwards comatibility, and should normally not be needed in user documents.

`\SetFistrumText` `\SetFistrumDefault{⟨name⟩}`

Loads the dummy text ⟨*name*⟩ (see section 4). This command does the same as option text, but it is kept for backwards compatibility.

`\SetFistrumDefault` `\SetFistrumDefault{⟨range⟩}`

Sets the default range for `\fistrum` and `\unpackfistrum`. This command does the same as option default-range, but it is kept for backwards compatibility.

3. General remarks on behaviour

Here are some topics that are general considerations about the behaviour of fistrum and its commands. These are technicalities that most end users don't care too much about, unless you are trying to do something beyond the usual "print me some dummy text".

3.1. Syntax of paragraph and sentence ranges

A ⟨*range*⟩ argument can either be blank, a single integer, or a proper integer range. If the ⟨*range*⟩ argument is blank, the commands behave as if the argument was not given at all. For example, `\fistrum[]` behaves exaclty like `\fistrum` and outputs the default paragraph range. Note that `\fistrum[][2-5]` does **not** behave as `\fistrum[2-5]`, but behaves as `\fistrum[1-7][2-5]` (assuming `default=range=1-7`), because the default value is then taken for the first argument. If the ⟨*range*⟩ argument is an integer, then only a single paragraph/sentence is selected.

If the argument contains a `-` (ASCII 45), it is interpreted as a *proper* range ⟨n_i⟩-⟨n_f⟩. In a proper range, if ⟨n_i⟩ is blank, it is taken to be the start of the possible range, and in the same way, if ⟨n_f⟩ is empty it is taken to be the end of the possible range. That is, `\fistrum[-9]` is the same as `\fistrum[1-9]`, and `\fistrum[5-]` is the same (assuming the standard 150-paragraph dummy text) as `\fistrum[5-150]`, and similarly, `\fistrum[-]` is the same as `\fistrum[1-150]`.

Only one `-` is allowed in a range, so if more than one `-` is given, an error is raised and no paragraphs/sentences are output. No paragraphs or sentences will be output also in case one of the ranges is reversed, so `\fistrum[2-1]` returns no paragraphs, as does `\fistrum[][2-1]` output no sentences, for example. Note that "returning no paragraphs/sentences" is not "the output is empty": that is mostly true, except that the `-before` and `-after` separators are still output (see section 3.3).

Finally, if a range spans more paragraphs or sentences than what the dummy text actually provides, the range is truncated so that it fits the available text. If the range in the argument does not intersect with the range provided by the dummy text, no paragraphs or sentences are output.

3.2. Hyphenation patterns

Since version 2.4, the command `\fistrum` automatically changes the hyphenation patterns when typesetting a dummy text, so that line-breaking looks better (see section **??**). This feature is on by default, so if you need the old behaviour you have to explicitly disable automatic language switching with `\setfistrum{auto-lang=false}`.

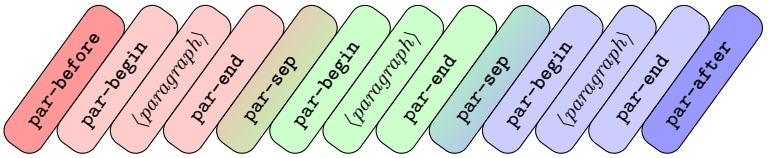
The language is defined individually for each dummy text (see section 4), but you may change it for the current dummy text by using `\setfistrum{language=`⟨*lang*⟩`}`. If you load another dummy text (for example with the text option), then the option language is also changed according to the dummy text loaded (see section 4).

3.3. Paragraph and sentence separators

As may be clear by now, fistrum has two modes of operation: sentence output, and paragraph output, selected by providing or not providing the second optional argument to `\fistrum`. In each mode, the dummy text is separated into chunks (paragraphs or sentences), which are counted, and then output accordingly.

When `\fistrum` (or `\unpackfistrum`) is used with a single (or no) optional argument, then a range of paragraphs is output, along with some "separators" (in the lack of a better name) between paragraphs, around each paragraph, and before and after the whole output. A schematic (very colorful, because I couldn't find a better visual) representation of the output is:

```
par-before
par-begin
par-sep
par-end
par-after
sentence-before
sentence-begin
sentence-sep
sentence-end
sentence-after
```



When `\fistrum` is called, the first thing it outputs is the par-before tokens. These tokens are output unconditionally, regardless of how many (if any) paragraph is output.

Then, before each paragraph in the range, `\fistrum` outputs the par-begin tokens, and then the actual text of the ⟨*paragraph*⟩, and then the par-end tokens. These tokens are output conditionally, if the paragraph text is output. If more than one paragraph is output, then the par-sep tokens are inserted between the par-end of one paragraph and the par-begin of the paragraph that follows.

Finally, at the end, the par-after tokens are inserted unconditionally at the end, same as for par-before.

As mentioned before, in case of an error parsing the range, the output will be no paragraphs, but the par-before and par-after tokens are still output.

The explanation above is equally valid for the starred variants. If `\fistrum*` is used, the par-before* tokens are inserted, and so on. It is also true for sentences (starred or otherwise), replacing `par` in the option names by `sentence`, so when you use, for example, `\fistrum[][1-9]`, the sentence-before tokens will be unconditionally inserted, and so on.

Note that, when `\fistrum` is used in sentence-mode (for example, with `\fistrum[1-3][1-9]`), only the `sentence-...` tokens are inserted in the output, regardless of how many paragraphs those sentences were collected from. In the same way, if paragraph-mode is being used, only `par-...` tokens are inserted.

3.3.1. Deprecated command-based syntax

Older versions of fistrum (from 2.0 to 2.3) provided 10 CamelCase commands for changing the separators, but the syntax was rather cumbersome to use, so the keyval syntax presented thus far was introduced in the hopes of making things a bit easier. The old commands will still exist for some time in the package, but with a deprecation warning. Changing to the keyval syntax is advised, so here is a correspondence table between the old and new syntaxes:

Old command	New key name
`\SetFistrumParListStart`	`par-before`
`\SetFistrumParListItemStart`	`par-begin`
`\SetFistrumParListItemSeparator`	`par-sep`
`\SetFistrumParListItemEnd`	`par-end`
`\SetFistrumParListEnd`	`par-after`
`\SetFistrumSentenceListStart`	`sentence-before`
`\SetFistrumSentenceListItemStart`	`sentence-begin`
`\SetFistrumSentenceListItemSeparator`	`sentence-sep`
`\SetFistrumSentenceListItemEnd`	`sentence-end`
`\SetFistrumSentenceListEnd`	`sentence-after`

Additionally, the command-based interface provided shortcuts `\SetFistrum`⟨*Thing*⟩`List(Item)Surrounders`, which are equivalent to just using the commands `\SetFistrum`⟨*Thing*⟩`List(Item)Start` then `\...End`. These don't provide any functionality, other than requiring a little less typing, so no key-val alternative was implemented. The `\...`⟨*Thing*⟩`...Surrounders` commands should be replaced by ⟨*thing*⟩`-before` and ⟨*thing*⟩`-after`, and the `\...`⟨*Thing*⟩`...ItemSurrounders` commands should be replaced by ⟨*thing*⟩`-begin` and ⟨*thing*⟩`-end`, as in the correspondence table below:

Old command	New key names
`\SetFistrumParListSurrounders`	`par-before`, `par-after`
`\SetFistrumParListItemSurrounders`	`par-begin`, `par-end`
`\SetFistrumSentenceListSurrounders`	`sentence-before`, `sentence-after`
`\SetFistrumSentenceListItemSurrounders`	`sentence-begin`, `sentence-end`

4. Loading and defining dummy texts

Starting with fistrum v2.2, a simple interface is provided to define and load other texts for the output of `\fistrum` and friends. This interface can, for example, be used to implement dummy texts in different languages without re-coding the logic implemented by fistrum.

`\NewFistrumPar` `\NewFistrumPar{⟨paragraph⟩}`

In order to provide a new text that will be used by fistrum, define the text by using a set of `\NewFistrumPar{⟨paragraph⟩}` commands in a file with the ending `.ftd.tex` (`ftd` means *fistrum text definition*[2]) to a location where your TEX system will find it. The ⟨*paragraph*⟩-argument is a single paragraph of the new text. Thus, the first occurence of `\NewFistrumPar` defines the first paragraph, the second occurence the second paragraph and so on.

`\SetFistrumLanguage` `\SetFistrumLanguage{⟨lang⟩}`

Additionally, tell fistrum the language of the dummy text using `\SetFistrumLanguage{⟨lang⟩}` somewhere in the `.ftd.tex` file.

To specify the new text as output for `\fistrum` and friends, use `\setfistrum{text=⟨name⟩}`, where ⟨*name*⟩ is the name of the file without the ending `.ftd.tex`, as given in the table below. When a new dummy text is loaded, the previous one is cleared, and the language is changed as well, according to the table.

File (`.ftd.tex`)	Language	Source	Description
`fistrum-la`	Latin	Chiquito	Contiene el texto *Lorem fistrum* tras seleccionar la opción latín en https://www.chiquitoipsum.com/.
`fistrum-es`	Spanish	Chiquito	Contiene el texto *Lorem fistrum* tras seleccionar la opción fistrum en https://www.chiquitoipsum.com/.

4.1. Guidelines on providing new dummy texts

`\SetFistrumText` more or less just uses an `\input` or, to be more precise, the LATEX3-variant `\file_input:n`, to load the `.ftd.tex` file. This means, that the file is not necessarily loaded in the preamble of the document and thus the contents of the file underlie the respective restrictions.

Should you want a new dummy text, create an issue in the GitHub repository[3] with the source for the dummy text.

Should you prefer to distribute the dummy text as a separate package, make sure that the text follows the layout of fistrum's dummy texts, so that everything works correctly. The dummy text definition file should contain a line with `\SetFistrumLanguage`, and then as many `\NewFistrumPar` entries as there are paragraphs in the dummy text. Make sure that the file has the `.ftd.tex` extension, and everything should work smoothly.

[2] To avoid name clashes with files using general languages as names, I chose to introduce the `.ftd.tex` file ending. I did not find a file with this ending in my `texmf`-tree, so I guess it is safe.

[3] https://github.com/daviddavo/fistrum

5. fistrum Implementation

```
1 ⟨*package⟩
2 ⟨@@=fistrum⟩
```

5.1. Variables

\g__fistrum_par_int Stores the number of paragraphs in the current text.

```
3 \int_new:N \g__fistrum_par_int
```

(*End definition for* `\g__fistrum_par_int`.)

\g__fistrum_language_tl Stores the language of the dummy text for hyphenation patterns.

```
4 \tl_new:N \g__fistrum_language_tl
```

(*End definition for* `\g__fistrum_language_tl`.)

\g_fistrum_default_range_tl The default range for fistrum paragraphs.

```
5 \tl_new:N \g_fistrum_default_range_tl
```

(*End definition for* `\g_fistrum_default_range_tl`.)

\l__fistrum_output_tl This variables is used to store the token list containing the selected output.

```
6 \tl_new:N \l__fistrum_output_tl
```

(*End definition for* `\l__fistrum_output_tl`.)

\g__fistrum_text_str Holds the current text loaded for the output of `\fistrum` and friends. Used to avoid loading the same text definition if it is already used.

```
7 \str_new:N \g__fistrum_text_str
```

(*End definition for* `\g__fistrum_text_str`.)

\l__fistrum_sep_set_str Holds the name of the active separator token set. By default it is empty to use the default separator set (empty).

```
8 \str_new:N \l__fistrum_sep_set_str
```

(*End definition for* `\l__fistrum_sep_set_str`.)

\l__fistrum_autolang_bool Boolean whether to change hyphenation patterns according to the dummy text language.

```
9 \bool_new:N \l__fistrum_autolang_bool
```

(*End definition for* `\l__fistrum_autolang_bool`.)

\q__fistrum_mark
\s__fistrum Quark and scan mark used throughout the package.

```
10 \quark_new:N \q__fistrum_mark
11 \scan_new:N \s__fistrum
```

(*End definition for* `\q__fistrum_mark` *and* `\s__fistrum`.)

\l__fistrum_tmpa_str
\l__fistrum_a_int
\l__fistrum_b_int Scratch variables.

```
12 \str_new:N \l__fistrum_tmpa_str
13 \int_new:N \l__fistrum_a_int
14 \int_new:N \l__fistrum_b_int
```

(*End definition for* `\l__fistrum_tmpa_str`, `\l__fistrum_a_int`, *and* `\l__fistrum_b_int`.)

__fistrum_tmp:w Scratch macro.

```
\cs_new_eq:NN \__fistrum_tmp:w ?
```

(*End definition for* `\__fistrum_tmp:w`.)

\l__fistrum_<thing>_<place>_<version>_tl These variables store the separators and delimiters added around the paragraphs and sentences, in the starred or nonstarred variants, as well as the generic version for runtime usage.

```
\clist_map_inline:nn { start, itemstart, itemseparator, itemend, end }
  {
    \clist_map_inline:nn { par, sentence }
      {
        \clist_map_inline:nn { { }, star, nostar }
          { \tl_new:c { l__fistrum_##1_#1_####1_tl } }
      }
    \tl_new:c { l__fistrum_par_#1_parsepar_tl }
  }
\tl_set:Nn \l__fistrum_par_itemseparator_parsepar_tl { ~ }
```

(*End definition for* `\l__fistrum_<thing>_<place>_<version>_tl`.)

5.2. Developer interface

__fistrum_parse_par_range:nNN
__fistrum_parse_par_range:eNN
__fistrum_parse_sentence_range:nNN
__fistrum_parse_sentence_range:eNN
__fistrum_parse_range_arg:nNNn
__fistrum_parse_range_arg:wnNNn
__fistrum_int_set:Nnn

Parses an argument that may be a single integer or an integer range separated by a `-`, and stores them into the integer registers `#2` and `#3`. If a number is blank, zero is used. If only a single number is given, `#3` is set equal to `#2`.

```
\cs_new_protected:Npn \__fistrum_parse_par_range:nNN #1 #2 #3
  {
    \tl_if_blank:nTF {#1}
      { \exp_args:NV \__fistrum_parse_range_arg:nNNn \g_fistrum_default_range_tl }
      { \__fistrum_parse_range_arg:nNNn {#1} }
          #2 #3 { \g__fistrum_par_int }
  }
\cs_new_protected:Npn \__fistrum_parse_sentence_range:nNN #1 #2 #3
  { \__fistrum_parse_range_arg:nNNn {#1} #2 #3 { \c_max_int } }
\cs_new_protected:Npn \__fistrum_parse_range_arg:nNNn #1
  {
    \exp_last_unbraced:No \__fistrum_parse_range_arg:wnNNn
      \tl_to_str:n { #1 - - } \s__fistrum {#1}
  }
\cs_new_protected:Npn \__fistrum_parse_range_arg:wnNNn
    #1 - #2 - #3 \s__fistrum #4 #5#6 #7
  {
    \str_if_eq:nnTF {#3} { - }
      {
        \__fistrum_int_set:Nnn #5 {#1} { 1 }
        \__fistrum_int_set:Nnn #6 {#2} {#7}
      }
      {
        \tl_if_empty:nTF {#3}
          {
            \__fistrum_int_set:Nnn #5 {#1} { \ERROR }
            \int_set_eq:NN #6 #5
```

```
          }
          {
            \msg_error:nnn { fistrum } { invalid-range } {#4}
            \__fistrum_parse_range_arg:nNNn { 2 - 1 } #5 #6 {#7}
          }
      }
  }
\cs_new_protected:Npn \__fistrum_int_set:Nnn #1 #2 #3
  { \int_set:Nn #1 { \tl_if_blank:nT {#2} {#3} #2 } }
\cs_generate_variant:Nn \__fistrum_parse_par_range:nNN { e }
\cs_generate_variant:Nn \__fistrum_parse_sentence_range:nNN { e }
```

(*End definition for* `\__fistrum_parse_par_range:nNN` *and others.*)

`\__fistrum_sep_item:nn` A shorthand to leave an (`\undexpanded`) token list.

```
\cs_new:Npn \__fistrum_sep_item:nn #1 #2
  { \exp_not:v { l__fistrum_#1_#2_ \l__fistrum_sep_set_str _tl } }
```

(*End definition for* `\__fistrum_sep_item:nn`.)

`\fistrum_get_range:nn` Expands to the paragraphs between ⟨*number*$_1$⟩ and ⟨*number*$_2$⟩ with the proper delimiters added. Text is returned in `\exp_not:n`, so this macro can be safely used in an `\edef`.

`\__fistrum_build_list:nn`
`\__fistrum_build_list_aux:n`
`\__fistrum_get_paragraph:ww`
`\__fistrum_get_paragraph_end:w`

```
\cs_new:Npn \fistrum_get_range:nn #1 #2
  {
    \__fistrum_sep_item:nn { par } { start }
    \use:e
      {
        \exp_not:N \__fistrum_get_paragraph:ww
        \__fistrum_build_list:nn {#1} {#2}
        \exp_not:N \q__fistrum_mark ;
        \exp_not:N \q__fistrum_mark ; \s__fistrum
      }
    \__fistrum_sep_item:nn { par } { end }
  }
\cs_new:Npn \__fistrum_build_list:nn #1 #2
  {
    \int_step_function:nnN
      { \int_max:nn {#1} { 1 } }
      { \int_min:nn {#2} { \g__fistrum_par_int } }
      \__fistrum_build_list_aux:n
  }
\cs_new:Npn \__fistrum_build_list_aux:n #1 { #1 ; }
\cs_new:Npn \__fistrum_get_paragraph:ww #1 ; #2 ;
  {
    \if_meaning:w \q__fistrum_mark #2
      \if_meaning:w \q__fistrum_mark #1
        \__fistrum_get_paragraph_end:w
      \else:
        \fistrum_get_paragraph:n {#1}
      \fi:
    \else:
      \fistrum_get_paragraph:n {#1}
      \__fistrum_sep_item:nn { par } { itemseparator }
    \fi:
```

```
    \__fistrum_get_paragraph:ww #2 ;
  }
\cs_new:Npn \__fistrum_get_paragraph_end:w #1 \s__fistrum { \fi: \fi: }
```

(*End definition for* `\fistrum_get_range:nn` *and others.*)

`\fistrum_get_paragraph:n` Expands to the paragraph ⟨*number*⟩ with the proper delimiters added. Text is returned in `\exp_not:n`, so this macro can be safely used in an `\edef`.

```
\cs_new:Npn \fistrum_get_paragraph:n #1
  {
    \__fistrum_sep_item:nn { par } { itemstart }
    \__fistrum_unexpanded_par:n {#1}
    \__fistrum_sep_item:nn { par } { itemend }
  }
```

(*End definition for* `\fistrum_get_paragraph:n`.)

`\__fistrum_unexpanded_par:n` Expands to the paragraph ⟨*number*⟩ wrapped in `\exp_not:n`. If ⟨*number*⟩ is out of range, it expands to nothing.

```
\cs_new:Npn \__fistrum_unexpanded_par:n #1
  {
    \bool_lazy_and:nnT
        { \int_compare_p:nNn { 0 } < {#1} }
        { \int_compare_p:nNn {#1}  < { \g__fistrum_par_int + 1 } }
      { \exp_not:v { g__fistrum_par_#1_tl } }
  }
```

(*End definition for* `\__fistrum_unexpanded_par:n`.)

`\fistrum_get_sentences:nnn` `\fistrum_get_sentences:nnV` `\__fistrum_get_sentences:nnnw` `\__fistrum_get_sentences_end:w` Expands to the sentences numbered between ⟨*number*$_1$⟩ and ⟨*number*$_2$⟩, inclusive, contained in the ⟨*text*⟩, and adding the proper separators.

```
\cs_new:Npn \fistrum_get_sentences:nnn #1 #2 #3
  {
    \__fistrum_sep_item:nn { sentence } { start }
    \exp_args:Ne \use_ii_i:nn { { \int_max:nn {#1} { 1 } } }
      { \__fistrum_get_sentences:nnnw { 1 } } {#2}
      #3 ~ \q__fistrum_mark .~ \s__fistrum
    \__fistrum_sep_item:nn { sentence } { end }
  }
\cs_new:Npn \__fistrum_get_sentences:nnnw #1 #2 #3 #4 .~
  {
    \int_compare:nNnT {#1} > {#3} { \__fistrum_get_sentences_end:w }
    \use:nn { \if_meaning:w \q__fistrum_mark } #4
      \exp_after:wN \__fistrum_get_sentences_end:w
    \else:
      \int_compare:nNnF {#1} < {#2}
        {
          \int_compare:nNnF {#1} = {#2}
            { \__fistrum_sep_item:nn { sentence } { itemseparator } }
          \__fistrum_sep_item:nn { sentence } { itemstart }
          \exp_not:n { #4 . }
          \__fistrum_sep_item:nn { sentence } { itemend }
        }
    \fi:
```

```
    \exp_args:Nf \__fistrum_get_sentences:nnnw { \int_eval:n { #1 + 1 } }
      {#2} {#3}
  }
\cs_new:Npn \__fistrum_get_sentences_end:w #1 \s__fistrum { }
\cs_generate_variant:Nn \fistrum_get_sentences:nnn { nnV }
```

(*End definition for* `\fistrum_get_sentences:nnn`, `\__fistrum_get_sentences:nnnw`, *and* `\__fistrum_get_sentences_end:w`.)

5.3. User- and developer-level commands

`\fistrumPar` Macro to typeset a single paragraph of *Lorem fistrum...* Was not officially available in version prior to 2.0.
`#1` : Number of the paragraph to typeset.
Implemented as follows:

```
\NewDocumentCommand \fistrumPar { m }
  {
    \__fistrum_deprecated:n { FistrumPar }
    \__fistrum_unexpanded_par:n {#1} \par
  }
```

(*End definition for* `\fistrumPar`.)

5.4. Tokens surrounding the *Lorem fistrum...* content

`\__fistrum_element_set:nnn` A general macro for setting starred/non-starred versions of several elements used between chunks of dummy text. Arguments are:
`#1` : Element name;
`#2` : Boolean true or false if the `*` variant was used;
`#3` : Value to set the element to.

```
\cs_new_protected:Npn \__fistrum_element_set:nnn #1 #2 #3
  { \tl_set:cn { l__fistrum_ #1 _ \IfBooleanF {#2} { no } star _tl } {#3} }
```

(*End definition for* `\__fistrum_element_set:nnn`.)

`\__fistrum_deprecated:n` Warns about deprecated commands and destroys itself.

```
\cs_new_protected:Npn \__fistrum_deprecated:n #1
  {
    \msg_warning:nnn { fistrum } { cmd-deprecated } {#1}
    \cs_gset_eq:NN \__fistrum_deprecated:n \use_none:n
  }
```

(*End definition for* `\__fistrum_deprecated:n`.)

`\SetFistrumParListStart` `\SetFistrumParListEnd` `\SetFistrumParListSurrounders` `\SetFistrumParListItemSeparator` `\SetFistrumParListItemStart` `\SetFistrumParListItemEnd` `\SetFistrumParListItemSurrounders` `\SetFistrumSentenceListStart` `\SetFistrumSentenceListEnd` `\SetFistrumSentenceListSurrounders` `\SetFistrumSentenceListItemSeparator` `\SetFistrumSentenceListItemStart` `\SetFistrumSentenceListItemEnd` `\SetFistrumSentenceListItemSurrounders` A dirty loop to quickly define the old command-based user-interface.

```
\cs_set_protected:Npn \__fistrum_tmp:w #1 #2 #3 #4
  {
    \str_set:Nx \l__fistrum_tmpa_str
      { #2 \tl_if_empty:nTF {#4} {#3} { start } }
    \use:e
      {
        \NewDocumentCommand \exp_not:c { SetFistrum #1 List #2 #3 }
            { s +m \tl_if_empty:nF {#4} { +m } }
```

```
          {
            \__fistrum_deprecated:n { SetFistrum #1 List #2 #3 }
            \__fistrum_element_set:nnn
              { \exp_args:Ne \str_lowercase:n { #1_\l__fistrum_tmpa_str } }
              {##1} {##2}
            \tl_if_empty:nT {#4} { \use_none:nnnn }
            \__fistrum_element_set:nnn { \str_lowercase:n { #1_#2 #4 } }
              {##1} {##3}
          }
      }
  }
\clist_map_inline:nn { Par, Sentence }
  {
    \clist_map_inline:nn
      { { Start } { }, { End } { }, { Surrounders } { end } }
      { \__fistrum_tmp:w {#1} { Item } ##1 \__fistrum_tmp:w {#1} { } ##1 }
    \__fistrum_tmp:w {#1} { Item } { Separator } { }
  }
```

(*End definition for* `\SetFistrumParListStart` *and others.*)

`\SetFistrumDefault` Command to change the default range used by `\fistrum` and friends.

⟨*range*⟩ Range to be used as default.

Implemented as:

```
\NewDocumentCommand \SetFistrumDefault { m }
  {
    \__fistrum_parse_par_range:eNN {#1} \l__fistrum_a_int \l__fistrum_b_int
    \tl_gset:Nx \g_fistrum_default_range_tl
      { \int_use:N \l__fistrum_a_int - \int_use:N \l__fistrum_b_int }
  }
```

(*End definition for* `\SetFistrumDefault`. *This function is documented on page 3.*)

The following macros are considered to be user-level commands and thus all lower-case.

`\fistrum` `#1` : Range-like string that specifies the number of the paragraphs taken from *Lorem fistrum...* If omitted, the value set by `\SetFistrumDefault` is used, which defaults to `1-7`.

`#2` : Sentences to be typeset from the range selected by ⟨*paragraph range*⟩. If sentences outside the number of sentences in ⟨*paragraph range*⟩ are specified, only existing sentences are typeset.

The difference between `\fistrum` and `\fistrum*` is the token(s) that are inserted after each paragraph (only if called without the second optional argument).

`\fistrum` and `\unpackfistrum` have the same interface and do almost the same thing, so both are implemented using a common macro `\__fistrum_do:nnnn` that does the heavy-lifting, and at the end executes the code in `#4`.

```
\NewDocumentCommand \fistrum { s O { \g_fistrum_default_range_tl } o }
  {
    \__fistrum_do:nnnn {#1} {#2} {#3}
      {
        \__fistrum_set_hyphens:
        \tl_use:N ##1
        \__fistrum_restore_hyphens:
      }
```

```
}
```

(*End definition for* `\fistrum`. *This function is documented on page 2.*)

`\unpackfistrum`
`\fistrumexp`

This command does the same as `\fistrum`, but instead of typesetting the paragraphs or sentences, it stores the expanded content in the `\fistrumexp` token list. The tokens between items of the list, set, for example, by using the package option space or by using the `\SetFistrum...List` commands, are x-expanded.

```
\NewDocumentCommand \unpackfistrum { s O { \g_fistrum_default_range_tl } o }
  { \__fistrum_do:nnnn {#1} {#2} {#3} { \tl_gset_eq:NN \fistrumexp ##1 } }
\cs_new_eq:NN \fistrumexp \prg_do_nothing:
```

(*End definition for* `\unpackfistrum` *and* `\fistrumexp`. *These functions are documented on page 2.*)

`\__fistrum_do:nnnn`
`\__fistrum_do:N`

This is the main macro for `\fistrum` and `\unpackfistrum`. It parses the paragraph range, sets the sentence/paragraph separators, then acts accordingly if a sentence range was provided.

```
\cs_new_protected:Npn \__fistrum_do:nnnn #1 #2 #3 #4
  {
    \cs_set_protected:Npn \__fistrum_do:N ##1 {#4}
    \__fistrum_parse_par_range:eNN {#2} \l__fistrum_a_int \l__fistrum_b_int
    \str_set_eq:NN \l__fistrum_tmpa_str \l__fistrum_sep_set_str
    \str_set:Nx \l__fistrum_sep_set_str { \IfBooleanF {#1} { no } star }
    \bool_lazy_or:nnTF
        { \tl_if_novalue_p:n {#3} }
        { \tl_if_blank_p:n {#3} }
      {
        \tl_set:Nx \l__fistrum_output_tl
          { \fistrum_get_range:nn { \l__fistrum_a_int } { \l__fistrum_b_int } }
      }
      {
        \str_set:Nn \l__fistrum_sep_set_str { parsepar }
        \tl_set:Nx \l__fistrum_output_tl
          { \fistrum_get_range:nn { \l__fistrum_a_int } { \l__fistrum_b_int } }
        \str_set:Nx \l__fistrum_sep_set_str { \IfBooleanF {#1} { no } star }
        \__fistrum_parse_sentence_range:eNN {#3} \l__fistrum_a_int \l__fistrum_b_int
        \tl_set:Nx \l__fistrum_output_tl
          {
            \fistrum_get_sentences:nnV { \l__fistrum_a_int } { \l__fistrum_b_int }
              \l__fistrum_output_tl
          }
      }
    \str_set_eq:NN \l__fistrum_sep_set_str \l__fistrum_tmpa_str
    \__fistrum_do:N \l__fistrum_output_tl
  }
\cs_new_eq:NN \__fistrum_do:N ?
```

(*End definition for* `\__fistrum_do:nnnn` *and* `\__fistrum_do:N`.)

`\__fistrum_set_hyphens:`
`\__fistrum_restore_hyphens:`

Selects the hyphenation patterns for the language of the dummy text, using `\hyphenrules` if that's defined. If `\hyphenrules` doesn't exist try setting hyphenation with `\__fistrum_set_hyphens_raw:`. Each `\__fistrum_set_hyphens_`⟨*method*⟩`:` function appropriately redefines `\__fistrum_restore_hypehens:` to reset the hyphenation patterns.

```
\cs_new_protected:Npn \__fistrum_set_hyphens:
  {
    \bool_if:NTF \l__fistrum_autolang_bool
      { \use:n } { \use_none:n }
      {
        \cs_if_exist:NTF \hyphenrules
          {
            \cs_if_exist:cTF { ver@polyglossia.sty }
              { \__fistrum_set_hyphens_polyglossia: }
              { \__fistrum_set_hyphens_babel: }
          }
          { \__fistrum_set_hyphens_raw: }
      }
  }
\cs_new_protected:Npn \__fistrum_restore_hyphens:
  { \prg_do_nothing: }
```

(*End definition for* `\__fistrum_set_hyphens:` *and* `\__fistrum_restore_hyphens:`.)

`\__fistrum_set_hyphens_babel:`

babel makes things pretty simple. We just check if `\l@⟨lang⟩` is defined, and if so, use `\hyphenrules` to set it, and once more to reset in `\__fistrum_restore_hyphens:`. `\hyphenrules` is actually an environment, but in babel its `\end` part does nothing, and its effect can be undone by just using another `\hyphenrules` on top of it.

If the language is not defined, the language either doesn't exist at all, or we are using LuaTEX. Both cases are handled by `\__fistrum_lang_not_available:`.

```
\cs_new_protected:Npn \__fistrum_set_hyphens_babel:
  {
    \cs_if_exist:cTF { l@ \g__fistrum_language_tl }
      {
        \exp_args:NV \hyphenrules \g__fistrum_language_tl
        \cs_set_protected:Npx \__fistrum_restore_hyphens:
          { \exp_not:N \hyphenrules { \languagename } }
      }
      { \__fistrum_lang_not_available: }
  }
```

(*End definition for* `\__fistrum_set_hyphens_babel:`.)

`\__fistrum_set_hyphens_polyglossia:`

polyglossia less friendly. We also check if the language is loaded (looking at `\⟨lang⟩@loaded`), and if it is, load it with the `hyphenrules` environment. Here we can't use the command form, as the `\end` part is not a no-op. This also means that an extra group is added around the dummy text, which causes issue `#1`[4] when used with wrapfig, for example. But not too much we can do about that for now.

In case the language is not loaded, fall back to `\__fistrum_set_hyphens_raw:` for a final attempt before giving up.

```
\cs_new_protected:Npn \__fistrum_set_hyphens_polyglossia:
  {
    \cs_if_exist:cTF { \g__fistrum_language_tl @loaded }
      {
        \exp_args:NnV \begin{hyphenrules} \g__fistrum_language_tl
        \cs_set_protected:Npn \__fistrum_restore_hyphens:
```

[4] https://github.com/daviddavo/fistrum/issues/1

```
            { \end{hyphenrules} }
        }
        { \__fistrum_set_hyphens_raw: }
    }
```

(*End definition for* `\__fistrum_set_hyphens_polyglossia:`.)

`\__fistrum_set_hyphens_raw:` If nothing else is available, try setting the language using `\language`⟨*number*⟩. This is always available, except with LuaTEX, which loads languages on-the-fly.

```
\cs_new_protected:Npn \__fistrum_set_hyphens_raw:
  {
    \cs_if_exist:cTF { l@ \g__fistrum_language_tl }
      {
        \use:x
          {
            \language \use:c { l@ \g__fistrum_language_tl }
            \cs_set_protected:Npn \__fistrum_restore_hyphens:
              { \language \int_eval:n { \language } \scan_stop: }
          }
      }
      { \__fistrum_lang_not_available: }
  }
```

(*End definition for* `\__fistrum_set_hyphens_raw:`.)

`\__fistrum_lang_not_available:` If the requested language is for some reason unavailable, warn the user, then fall back to the current language. If the requested language and `\languagename` are the same, we are probably running into this warning again because we have no languages loaded (e.g., testing environments). In that case, shut up because there's not much that can be done (and a warning that says "language `english` is not available; using `english` instead" is of little use).

```
\cs_new_protected:Npn \__fistrum_lang_not_available:
  {
    \str_if_eq:VVF \g__fistrum_language_tl \languagename
      {
        \msg_warning:nnx { fistrum } { missing-language }
          { \g__fistrum_language_tl }
        \tl_gset_eq:NN \g__fistrum_language_tl \languagename
      }
  }
```

(*End definition for* `\__fistrum_lang_not_available:`.)

`\NewFistrumPar` Developer-Level macro to add a paragraph to the dummy text used by `\fistrum` and related commands. To specify a new dummy text, see section 4.

```
\cs_new_protected:Npn \NewFistrumPar #1
  {
    \int_gincr:N \g__fistrum_par_int
    \tl_gclear_new:c { g__fistrum_par_ \int_use:N \g__fistrum_par_int _tl }
    \tl_gset:cn { g__fistrum_par_ \int_use:N \g__fistrum_par_int _tl } {#1}
  }
```

(*End definition for* `\NewFistrumPar`. *This function is documented on page* 6.)

\SetFistrumText

Used to select and load the text output by `\fistrum` and friends. See the section on loading and defining new outputs for `\fistrum` (section 4). It first checks whether the requested text is already loaded, and if not, it loads the corresponding fistrum text definition file, and clears remaining paragraphs from the previous text, in case their lengths differ.

```
\NewDocumentCommand \SetFistrumText { m }
  {
    \str_if_eq:VnF \g__fistrum_text_str {#1}
      {
        \tl_gset:Nn \g__fistrum_language_tl { english }
        \int_gzero:N \g__fistrum_par_int
        \file_input:n { #1.ftd }
        \str_gset:Nn \g__fistrum_text_str {#1}
      }
  }
```

(*End definition for* `\SetFistrumText`*. This function is documented on page 3.*)

\SetFistrumLanguage

This macro sets the language for hyphenation patterns of the dummy text. When a new fistrum text is read, this is reset.

```
\NewDocumentCommand \SetFistrumLanguage { m }
  { \tl_gset:Nn \g__fistrum_language_tl {#1} }
```

(*End definition for* `\SetFistrumLanguage`*. This function is documented on page 6.*)

5.5. Package options and defaults

\fistrumRestoreParList
\fistrumRestoreSentenceList
\fistrumRestoreAll
__fistrum_delim_restore:nnn
__fistrum_restore_par_list:
__fistrum_restore_sentence_list:

These are some auxiliaries for the package options and for setting up the default behaviour.

```
\cs_new_protected:Npn \__fistrum_delim_restore:nnn #1 #2 #3
  {
    \keys_set:nn { fistrum }
      {
        #1-before  = , #1-begin  = , #1-end  = , #1-after  = ,
        #1-before* = , #1-begin* = , #1-end* = , #1-after* = ,
        #1-sep = {#2}, #1-sep* = {#3}
      }
  }
\cs_new_protected:Nn \__fistrum_restore_sentence_list:
  { \__fistrum_delim_restore:nnn { sentence } { ~ } { ~ } }
\cs_new_eq:NN \__fistrum_restore_par_list: ?
\cs_new_protected:Npn \fistrumRestoreParList
  {
    \__fistrum_deprecated:n { FistrumRestoreParList }
    \__fistrum_restore_par_list:
  }
\cs_new_protected:Npn \fistrumRestoreSentenceList
  {
    \__fistrum_deprecated:n { FistrumRestoreSentenceList }
    \__fistrum_restore_sentence_list:
  }
\cs_new_protected:Npn \fistrumRestoreAll
  {
    \__fistrum_deprecated:n { FistrumRestoreAll }
    \__fistrum_restore_par_list: \__fistrum_restore_sentence_list:
  }
```

(*End definition for* `\fistrumRestoreParList` *and others.*)

`\setfistrum` Here are the options available at load-time and to `\setfistrum`.

```
\NewDocumentCommand \setfistrum { +m }
  { \keys_set:nn { fistrum } {#1} }
\keys_define:nn { fistrum }
  {
```

nopar is implemented as a choice key instead of a boolean so we can update the separators using `\__fistrum_delim_restore:nnn`. It's initially false, and the default is `true` so that `\usepackage[nopar]{fistrum}` works as it always did.

```
    nopar .choice: ,
    nopar / true .code:n =
      {
        \cs_gset_protected:Npn \__fistrum_restore_par_list:
          { \__fistrum_delim_restore:nnn { par } { ~ } { \par } }
      } ,
    nopar / false .code:n =
      {
        \cs_gset_protected:Nn \__fistrum_restore_par_list:
          { \__fistrum_delim_restore:nnn { par } { \par } { ~ } }
      } ,
    nopar .initial:n = false ,
    nopar .default:n = true  ,
```

auto-lang sets `\l__fistrum_autolang_bool`. It is initially `true`, changing the default behaviour from previous versions.

```
    auto-lang .bool_set:N = \l__fistrum_autolang_bool ,
    auto-lang .initial:n = true ,
    auto-lang .default:n = true ,
```

text just does `\SetFistrumText`. The `initial` value is not set here because this chunk of code is executed in expl3 syntax, then thetextloadswithoutspaces, so `\setfistrum{text=fistrum-la}` is used later.

```
    text .code:n = \SetFistrumText{#1} ,
    text .value_required:n = true ,
```

language sets the language to be used when typesetting.

```
    language .tl_gset:N = \g__fistrum_language_tl ,
    language .value_required:n = true ,
```

default-range does `\SetFistrumDefault`, initially `1-7`, as documented. It's default is also `1-7` so that the key has two meanings: `\setfistrum{default-range=`⟨*range*⟩`}` sets the range to the given value, while `\setfistrum{default-range}` sets the range to the "default default range". Pretty neat :)

```
    default-range .code:n = \SetFistrumDefault{#1} ,
    default-range .initial:n = 1-7 ,
    default-range .default:n = 1-7 ,
  }
```

This chunk defines the keys ⟨*thing*⟩-⟨*place*⟩[*], where ⟨*thing*⟩ is `par` or `sentence`, ⟨*place*⟩ is `before`, `begin`, `sep`, `end`, and `after`, which totals 10 keys, and another 10 with the `*` in the name. Each sets a token list called `\l__fistrum_`⟨*thing*⟩`_`⟨*place*⟩`_[no]star_tl`.

```
\cs_set_protected:Npn \__fistrum_tmp:w #1 #2 #3
  {
```

```
    \keys_define:nn { fistrum }
      {
        #1-before #2 .tl_set:c = l__fistrum_#1_start         _#3star_tl ,
        #1-begin  #2 .tl_set:c = l__fistrum_#1_itemstart     _#3star_tl ,
        #1-sep    #2 .tl_set:c = l__fistrum_#1_itemseparator _#3star_tl ,
        #1-end    #2 .tl_set:c = l__fistrum_#1_itemend       _#3star_tl ,
        #1-after  #2 .tl_set:c = l__fistrum_#1_end           _#3star_tl ,
      }
  }
\__fistrum_tmp:w { par } { } { no } \__fistrum_tmp:w { sentence } { } { no }
\__fistrum_tmp:w { par }  *  {    } \__fistrum_tmp:w { sentence }  *  {    }
```

(*End definition for* `\setfistrum`. *This function is documented on page* 2.)

Now turn `\ExplSyntaxOff` for a while, and load the default *Lorem fistrum...* text, then process the package options, and finally turn `\ExplSyntaxOn` again. Finally, call `\__fistrum_restore_par_list:` and `\__fistrum_restore_sentence_list:` to set the defaults (`\__fistrum_restore_par_list:` may have been redefined by nopar).

```
\ExplSyntaxOff
\setfistrum{text=fistrum-es}
\ProcessKeysOptions{fistrum}
\ExplSyntaxOn
\__fistrum_restore_par_list:
\__fistrum_restore_sentence_list:
```

5.6. Messages

Now define the messages used throughout the package.

```
\msg_new:nnn { fistrum } { invalid-range }
  { Invalid~number~or~range~'#1'. }
\msg_new:nnn { fistrum } { cmd-deprecated }
  {
    Command~'\iow_char:N\\#1'~deprecated. \\
    See~the~fistrum~documentation~for~help.
  }
\msg_new:nnn { fistrum } { missing-language }
  {
    Unknown~language~'#1'.~Hyphenation~patterns~for~
    '\languagename'~will~be~used~instead.
    \sys_if_engine_luatex:T
      {
        \\ \\
        \cs_if_exist:cTF { ver@polyglossia.sty }
          {
            With~polyglossia,~you~have~to~explicitly~load~languages~
            with~\iow_char:N\\setotherlanguage{#1}~or~similar.
          }
          {
            With~LuaTeX,~fistrum~requires~babel~to~get~proper~
            hyphenation~(you~can~use~
            \iow_char:N\\usepackage[base]{babel}).
          }
      }
  }
```

403 ⟨/package⟩