

SCONTENTS

Stores \LaTeX contents

v1.6 — 2019-10-26*

©2019 by Pablo González †

CTAN: <http://www.ctan.org/pkg/scontents>
GIT: <https://github.com/pablgonz/scontents>

Abstract

The `scontents` package stores valid \LaTeX code in $\langle\textit{sequences}\rangle$ using the `l3seq` module of `expl3`. The $\langle\textit{stored content}\rangle$, including *verbatim*, can be used as many times as desired in the document, additionally can be written to $\langle\textit{external files}\rangle$.

Contents

1	Motivation and Acknowledgments . . .	1	5	Other commands provided	6
2	License and Requirements	1	5.1	The command <code>\meaningsc</code>	6
3	The <code>scontents</code> package	2	5.2	The command <code>\countsc</code>	6
3.1	Description of the package and load . .	2	5.3	The command <code>\cleanseqsc</code>	6
3.2	The TAB character	2	6	The <code>scontents</code> package in action	7
3.3	Configuration of the options	2	7	Examples	8
3.4	Options Overview	3	7.1	From answers package	8
4	User interface	3	7.2	From filecontentsdef package	8
4.1	The environment <code>scontents</code>	3	7.3	From TeX-SX	8
4.2	The command <code>\Scontents</code>	4	7.4	Customization of <code>verbatimsc</code>	10
4.3	The command <code>\getstored</code>	4	8	Change history	12
4.4	The command <code>\foreachsc</code>	5	9	Index of Documentation	13
4.5	The command <code>\typestored</code>	5	10	Implementation	14
4.6	The environment <code>verbatimsc</code>	5	11	Index of Implementation	35

1 Motivation and Acknowledgments

In \LaTeX there is no direct way to record content for later use, although you can do this using `\macros`, recording $\langle\textit{verbatim content}\rangle$ is a problem, usually you can avoid this by creating external files or boxes. The general idea of this package is to try to imitate this implementation *buffers* that has ConTeXt which allows you to save content in memory, including *verbatim*, to be used later. The package `filecontentsdef` solves this problem and since `expl3` has an excellent way to manage data, ideas were combined giving rise to this package.

This package would not be possible without the great work of JEAN FRANÇOIS BURNOL who was kind enough to take my requirements into account and add the `filecontentsdefmacro` environment. Also a special thanks to Phelype Oleinik who has collaborated and adapted a large part of the code and all \LaTeX 3 team for their great work and to the different members of the TeX-SX community who have provided great answers and ideas. Here a note of the main ones:

1. Stack datastructure using LaTeX
2. LaTeX equivalent of ConTeXt buffers
3. Storing an array of strings in a command
4. Collecting contents of environment and store them for later retrieval
5. Collect contents of an environment (that contains *verbatim* content)

2 License and Requirements

Permission is granted to copy, distribute and/or modify this software under the terms of the LaTeX Project Public License (lpp), version 1.3 or later (<http://www.latex-project.org/lppl.txt>). The software has the status “maintained”.

The `scontents` package loads `expl3`, `xparse` and `l3keys2e`. This package can be used with `plain`, `context`, `xelatex`, `lualatex`, `pdflatex` and the classical workflow `latex»dvips»ps2pdf`.

*This file describes a documentation for v1.6, last revised 2019-10-26.

†E-mail: pablgonz@educarchile.cl

3 The `scontents` package

3.1 Description of the package and load

The `scontents` package allows to `<store contents>` in `<sequences>` or `<external files>`. In some ways it is similar to the `filecontentsdef` package, with the difference in which the `<content>` is stored. The idea behind this package is to get an approach to ConTeXt “*buffers*” by making use `<sequences>`.

The package is loaded in the usual way:

For \LaTeX users

```
\usepackage{scontents}
```

or

```
\usepackage[<key = val>]{scontents}
```

For plain \TeX users

```
\input scontents.tex
```

For ConTeXt users

```
\usemodule{scontents}
```

3.2 The TAB character

Some users use horizontal TABs “`␣`” from keyboard to indented the source code of the document and depending on the text editor used, some will use real TABs (“hard tabs”), others with “soft tabs”(␣ or ␣␣␣␣) or both.

At first glance it may seem the same, but the way in which TABs (“hard tabs”) are processed according to the context in which they are found within a file, both in `<reading>`¹ and `<writing>`² are different and may have adverse consequences.

In a standard \LaTeX document, the character TAB “`␣`” are treated as explicit spaces (in most contexts) and is the behavior when `<stored contents>`, but when `<writing files>` these are preserved.

With a \TeX Live distribution, the TAB character is “printable” for `latex`, `pdflatex` and `lualatex`, but if you use `xelatex` you must add the `-8bit` option on the command line, otherwise you will get \TeX -TAB (`^I`) in the `<output file>`.

As a general recommendation “Do not use TAB character unless strictly necessary”, for example within a `verbatim` environment that supports this character such as `Verbatim` of the package `fancyvrb` or `lstlisting` of the package `listings` or when you want to generate a `MakeFile` file.

3.3 Configuration of the options

Most of the options can be passed directly to the package or using the command `\setupsc`. All boolean keys can be passed using the equal sign “=” or just the name of the key, all unknown keys will return an error. In this section are described some of the options, a summary of all options is shown in section 3.4.

```
\setupsc {<keyval list>}
```

The command `\setupsc` sets the `<keys>` in a global way, it can be used both in the preamble and in the body of the document as many times as desired.

```
verb-font = {<font family>} (default: \ttfamily)
```

Sets the `` used to display the `<stored content>` for the commands `\typestored` and `\meaningsc`. This key is only available as a package option or using `\setupsc`.

```
store-all = {<seq name>} (default: not used)
```

It is a `<meta-key>` that sets the `store-env` key of the `scontents` environment and the `store-cmd` key of the `\Scontents` command. This key is only available as a package option or using `\setupsc`.

```
print-all = {<true | false>} (default: false)
```

It is a `<meta-key>` that sets the `print-env` key of the `scontents` environment and the `print-cmd` key of the `\Scontents` command. This key is only available as a package option or using `\setupsc`.

¹Check the answer given by Ulrich Diez in [Keyboard TAB character in argument v \(xparse\)](#).

²Check the answer given by Enrico Gregorio in [How to output a tabulation into a file](#).

`force-eol = {⟨true | false⟩}` (default: *false*)

Sets if the end of line for the ⟨*stored content*⟩ is hidden or not. This key is necessary only if the last line is the closing of some environment defined by the `fancyvrb` package as `\end{Verbatim}` or another environment that does not support a comments “%” after closing `\end{...}%`. This key is available for the `scontents` environment and the `\Scontents` command.

`width-tab = {⟨integer⟩}` (default: 1)

Sets the equivalence in ⟨*spaces*⟩ for the character TAB used when displaying stored content in *verbatim style*. The value must be a ⟨*positive integer*⟩. This key is available for the `\typestored` and the `\meaningsc` commands.

3.4 Options Overview

Summary of available options.

key	package	\setupsc	scontents	\Scontents	\Scontents*	\typestored	\meaningsc
store-env	✓	✓	✓	✗	✗	✗	✗
store-cmd	✓	✓	✗	✓	✓	✗	✗
print-env	✓	✓	✓	✗	✗	✗	✗
print-cmd	✓	✓	✗	✓	✓	✗	✗
print-all	✓	✓	✗	✗	✗	✗	✗
store-all	✓	✓	✗	✗	✗	✗	✗
write-env	✗	✗	✓	✗	✗	✗	✗
write-out	✗	✗	✓	✗	✗	✗	✗
width-tab	✓	✓	✗	✗	✗	✓	✓
force-eol	✓	✓	✓	✗	✓	✗	✗
verb-font	✓	✓	✗	✗	✗	✗	✗

4 User interface

The user interface consists in `scontents` environment, `\Scontents` and `\Scontents*` commands to ⟨*stored content*⟩ and `\getstored` command to get the ⟨*stored content*⟩ along with other utilities described in this documentation.

4.1 The environment scontents

`scontents` `\begin{scontents}[⟨keyval list⟩]`
 ⟨*env contents*⟩

`\end{scontents}`

The `scontents` environment allows you to ⟨*store*⟩ and ⟨*write*⟩ content, including *verbatim* material. After the package has been loaded, the environment can be used both in the preamble and in the body of the document.

For the correct operation `\begin{scontents}` and `\end{scontents}` must be in different lines, all ⟨*keys*⟩ must be passed separated by commas and “without separation” of the start of the environment.

Comments “%” or “any character” after `\begin{scontents}` or `[⟨keyval list⟩]` on the same line are not supported, the package will return an “error” message if this happens. In a similar way comments “%” or “any character” after `\end{scontents}` on the same line the package will return a “warning” message.

The environment can be ⟨*nested*⟩ if it is properly balanced and does not appear “literally” in commented lines or in some *verbatim* environment or command. As an example:

```
\begin{scontents}[store-env=outer]
This text is in the outer environment (before nested).
\begin{scontents}[store-env=inner]
This text is found in the inner environment (inside of nested).
\end{scontents}
This text is in the outer environment (after nested).
\end{scontents}
```

Of course, content stored in the ⟨*inner*⟩ sequence is only available after content stored in the ⟨*outer*⟩ sequence one has been retrieved, either by using the key `print-env` or `\getstored` command.

It is advisable to store content within sequences with different names, so as not to get lost in the order in which content is stored.

In plain \TeX there is not environments as in \LaTeX . Instead of using the environment `scontents`, one should use a *pseudo environment* delimited by `\scontents` and `\endscontents`. Con \TeX t users should use `\startcontents` and `\stopcontents`.

Options for environment

The environment options can be configured globally using option in package or the `\setupsc` command and locally using `[⟨key = val⟩]` in the environment. The key `force-eol` is available for this environment.

`store-env = {⟨seq name⟩}` (default: `contents`)

Sets the name of the `⟨sequence⟩` in which the contents will be stored. If the sequence does not exist, it will be created globally.

`print-env = {⟨true | false⟩}` (default: `false`)

Sets if the `⟨stored content⟩` is displayed or not at the time of running the environment. The content is extracted from the `⟨sequence⟩` in which it is stored.

`write-env = {⟨file.ext⟩}` (default: `not used`)

Sets the name of the `⟨external file⟩` in which the `⟨contents⟩` of the environment will be written. The `⟨file.ext⟩` will be created in the working directory, if `⟨file.ext⟩` exists it will be overwritten, relative or absolute paths are not supported. The characters TABs will be written in `⟨file.ext⟩` and the `⟨contents⟩` will be stored in the sequence established at that time. X_YLaTeX users using the TAB character must add `-8bit` at the command line, otherwise you will get T_EX-TAB (\^I) in `⟨file.ext⟩`.

`write-out = {⟨file.ext⟩}` (default: `not used`)

Sets the name of the `⟨external file⟩` in which the `⟨contents⟩` of the environment will be written. The `⟨file.ext⟩` will be created in the working directory, if `⟨file.ext⟩` exists it will be overwritten, relative or absolute paths are not supported. The characters TABs will be written in `⟨file.ext⟩`, the rest of the `⟨keys⟩` will not be available and the `⟨contents⟩` will NOT be stored in any sequence. X_YLaTeX users using the TAB character must add `-8bit` at the command line, otherwise you will get T_EX-TAB (\^I) in `⟨file.ext⟩`.

4.2 The command \Scontents

`\Scontents` `\Scontents[⟨key = val⟩]{⟨argument⟩}`
`\Scontents* [⟨key = val⟩]{⟨argument⟩}`
`\Scontents* [⟨key = val⟩]{⟨del⟩⟨argument⟩⟨del⟩}`

The `\Scontents` command reads the `{⟨argument⟩}` in standard mode. It is not possible to pass environments such as *verbatim*, but it is possible to use the implementation of `\verb` provided by the `fvextra` package for contents on one line and `\lstinline` from `listings` package, but it is preferable to use the starred version.

The `\Scontents*` command reads the `{⟨argument⟩}` under *verbatim* category code regimen. If its first delimiter is a brace, it will be assumed that the `{⟨argument⟩}` is nested into braces. Otherwise it will be assumed that the ending of that `⟨argument⟩` is delimited by that first delimiter `⟨del⟩` like command `\verb`.

Blank lines are preserved, escaped braces “\{” and “\}” must also be balanced if the argument is used with braces and TABs characters typed from the keyboard are converted into spaces.

Both versions can be used anywhere in the document and cannot be used as an `⟨argument⟩` for other command.

Options for command

The command options can be configured globally using option in package or the `\setupsc` command and locally using `[⟨key = val⟩]`. The key `force-eol` is available for this command.

`store-cmd = {⟨seq name⟩}` (default: `contents`)

Sets the name of the `⟨sequence⟩` in which the contents will be stored. If the sequence does not exist, it will be created globally.

`print-cmd = {⟨true | false⟩}` (default: `false`)

Sets if the `⟨stored content⟩` is displayed or not at the time of running the command. The content is extracted from the `⟨sequence⟩` in which it is stored.

4.3 The command \getstored

`\getstored` `\getstored[⟨index⟩]{⟨seq name⟩}`

The command `\getstored` gets the content stored in `{⟨seq name⟩}` according to the `⟨index⟩` in which it was stored. The command is robust and can be used as an `⟨argument⟩` for another command. If the optional argument is not passed it defaults to the first element stored in the `{⟨seq name⟩}`.

4.4 The command `\foreachsc`

```
\foreachsc [⟨key = val⟩]{⟨seq name⟩}
```

The command `\foreachsc` goes through and executes the command `\getstored` on the contents stored in `{⟨seq name⟩}`. If you pass without options run `\getstored` on all contents stored in `{⟨seq name⟩}`.

Options for command

`sep = {⟨code⟩}` (default: *empty*)

Establishes the separation between each content stored in `{⟨seq name⟩}`. For example, you can use `sep={\\[10pt]}` for vertical separation of stored contents.

`step = {⟨integer⟩}` (default: *1*)

Sets the increment (`⟨step⟩`) applied to the value set by key `start` for each element stored in the `{⟨seq name⟩}`. The value must be a *positive integer*.

`start = {⟨integer⟩}` (default: *1*)

Sets the *index* number of the `{⟨seq name⟩}` from which execution will start. The value must be a *positive integer*.

`stop = {⟨integer⟩}` (default: *total*)

Sets the *index* number of the `{⟨seq name⟩}` from which execution it will finish executing. The value must be a *positive integer*.

`before = {⟨code⟩}` (default: *empty*)

Sets the `{⟨code⟩}` that will be executed *before* each content stored in `{⟨seq name⟩}`. The `{⟨code⟩}` must be passed between braces.

`after = {⟨code⟩}` (default: *empty*)

Sets the `{⟨code⟩}` that will be executed *after* each content stored in `{⟨seq name⟩}`. The `{⟨code⟩}` must be passed between braces.

`wrapper = {⟨code⟩{#1} more code}` (default: *empty*)

Wraps the content stored in `{⟨seq name⟩}` referenced by `{⟨#1⟩}`. The `{⟨code⟩}` must be passed between braces. For example `\foreachsc[wrapper={\makebox[1em][l]{#1}}]{contents}`.

4.5 The command `\typestored`

```
\typestored [⟨index, width-tab = number⟩]{⟨seq name⟩}
```

The command `\typestored` internally places the content stored in the `{⟨seq name⟩}` into the `verbatimsc` environment. The *index* corresponds to the position in which the content is stored in the `{⟨seq name⟩}`. If the optional argument is not passed it defaults to the first element stored in the `{⟨seq name⟩}`. The key `width-tab` is available for this command.

The `verbatim` package is not compatible with the implementation of the `verbatimsc` environment used by this command, if you are a user of that package consider customizing the `verbatimsc` environment.

4.6 The environment `verbatimsc`

```
verbatimsc
```

Internal environment used by `\typestored` to display *verbatim style* contents.

One consideration to keep in mind is that this is a *representation* of the *stored content* in a *verbatim* environment and not a real *verbatim* environment.

The `verbatimsc` environment can be customized in the following ways:

Using the package `fancyvrb`:

```
\makeatletter
\let\verbatimsc\undefined
\let\endverbatimsc\undefined
\makeatother
\DefineVerbatimEnvironment{verbatimsc}{Verbatim}{numbers=left}
```

Using the package `minted`:

```
\makeatletter
\let\verbatimsc\undefined
\let\endverbatimsc\undefined
\makeatother
```

```

\usepackage{minted}
\newminted{tex}{linenos}
\newenvironment{verbatimsc}{\VerbatimEnvironment\begin{texcode}}{\end{texcode}}

```

Using the package listings:

```

\makeatletter
\let\verbatimsc@undefined
\let\endverbatimsc@undefined
\makeatother
\usepackage{listings}
\lstnewenvironment{verbatimsc}
{
  \lstset{
    basicstyle=\small\ttfamily,
    columns=fullflexible,
    language=[LaTeX]TeX,
    numbers=left,
    numberstyle=\tiny\color{gray},
    keywordstyle=\color{red}
  }
}{}

```

5 Other commands provided

5.1 The command `\meaningsc`

```
\meaningsc[⟨index, width-tab = number⟩]{⟨seq name⟩}
```

The command `\meaningsc` executes `\meaning` on the content stored in $\{\langle seq name \rangle\}$. The $\langle index \rangle$ corresponds to the position in which the content is stored in the $\{\langle seq name \rangle\}$.

If the optional argument is not passed it defaults to the first element stored in the $\{\langle seq name \rangle\}$. The key `width-tab` is available for this command.

5.2 The command `\countsc`

```
\countsc{\langle seq name \rangle}
```

The command `\countsc` count a number of contents stored in $\{\langle seq name \rangle\}$.

5.3 The command `\cleanseqsc`

```
\cleanseqsc{\langle seq name \rangle}
```

The command `\cleanseqsc` remove all contents stored in $\{\langle seq name \rangle\}$.

6 The `scontents` package in action

Remember the abstract on the first page?, this is it:

Abstract

The `scontents` package stores valid \TeX code in `<sequences>` using the `l3seq` module of `expl3`. The `<stored content>`, including *verbatim*, can be used as many times as desired in the document, additionally can be written to `<external files>`.

And the description of the package?

The `scontents` package allows to `<store contents>` in `<sequences>` or `<external files>`. In some ways it is similar to the `filecontentsdef` package, with the difference in which the `<content>` is stored. The idea behind this package is to get an approach to $\text{\texttt{ConTeXt}}$ “*buffers*” by making use `<sequences>`.

I’ve only written:

```
\begin{abstract}
The \mypkg{scontents} package stores valid \hologo{(La)TeX} code in
\mymeta{sequences} using the \mypkg{l3seq} module of \mypkg{expl3}.
The \mymeta{stored content}, including \emph{verbatim}, can be used
as many times as desired in the document, additionally can be written
to \mymeta{external files}.
\end{abstract}
```

and

```
The \mypkg{scontents} package allows to \mymeta{store contents} in
\mymeta{sequences} or \mymeta{external files}. In some ways it is
similar to the \mypkg{filecontentsdef} package, with the difference
in which the \mymeta{content} is stored. The idea behind this package
is to get an approach to \hologo{ConTeXt} \enquote{\emph{buffers}}
by making use \mymeta{sequences}.
```

Of course, I didn’t copy and paste. The real code they were written with is:

```
1 \begin{scontents}[store-env=abstract,print-env=true]
2 \begin{abstract}
3 The \mypkg{scontents} package stores valid \hologo{(La)TeX} code in
4 \mymeta{sequences} using the \mypkg{l3seq} module of \mypkg{expl3}.
5 The \mymeta{stored content}, including \emph{verbatim}, can be used
6 as many times as desired in the document, additionally can be written
7 to \mymeta{external files}.
8 \end{abstract}
9 \end{scontents}
```

and

```
1 \begin{scontents}[store-env=description, print-env=true]
2 The \mypkg{scontents} package allows to \mymeta{store contents} in
3 \mymeta{sequences} or \mymeta{external files}. In some ways it is
4 similar to the \mypkg{filecontentsdef} package, with the difference
5 in which the \mymeta{content} is stored. The idea behind this package
6 is to get an approach to \hologo{ConTeXt} \enquote{\emph{buffers}}
7 by making use \mymeta{sequences}.
8 \end{scontents}
```


I stored the content in memory and then ran `\getstored` and `\tpestored`. This is one of the ways you can use `scontents`.

7 Examples

These are some (adapted) examples that have served as inspiration for the creation of this package.

7.1 From answers package

Example 1

Adaptation of example 1 (ansexam1) of the package answers .


```

1 \documentclass[12pt,a4paper]{article}
2 \usepackage[store-cmd=solutions]{scontents}
3 \newtheorem{ex}{Exercise}
4 \begin{document}
5 \section{Problems}
6 \begin{ex}
7 First exercise
8 \Scontents{
9   First solution.
10 }
11 \end{ex}
12
13 \begin{ex}
14 Second exercise
15 \Scontents{
16   Second solution.
17 }
18 \end{ex}
19
20 \section{Solutions}
21 \foreachsc[sep={\\[10pt]}]{solutions}
22 \end{document}

```

7.2 From filecontentsdef package

Example 2

Adaptation of example from package filecontentsdef .

```

1 \documentclass{article}
2 \usepackage[store-env=defexercise,store-cmd=defexercise]{scontents}
3 \pagestyle{empty}
4 \begin{document}
5 % not starred
6 \Scontents{
7 Prove that  $[x^n+y^n=z^n]$  is not solvable in positive integers if  $n$  is at
8 most  $3$ .\par
9 }
10 % starred
11 \Scontents*|Refute the existence of black holes in less than  $140$  characters.|
12 % write environment to \jobname.txt
13 \begin{scontents}[write-env=\jobname.txt]
14 \def\NSA{NSA}%
15 Prove that factorization is easily done via probabilistic algorithms and
16 advance evidence from knowledge of the names of its employees in the
17 seventies that the \NSA\ has known that for 40 years.\par
18 \end{scontents}
19 % see all stored
20 \begin{itemize}
21 \foreachsc[before={\item }]{defexercise}
22 \end{itemize}
23 % \getstored are robust :)
24 \section{\getstored[2]{defexercise}}
25 \end{document}

```

7.3 From TeX-SX

Example 3

Adapted from [LaTeX equivalent of ConTeXt buffers](#) .

```

1 \documentclass{article}
2 \usepackage[store-cmd=tikz]{scontents}
3 \usepackage{tikz}
4 \pagestyle{empty}
5 \Scontents*\matrix{\node (a) {$a$} ; & \node (b) {$b$} ; \\ } ;}
6 \Scontents*\matrix[ampersand replacement=\&]
7 { \node (a) {$a$} ; \& \node (b) {$b$} ; \\ } ;}
8 \Scontents*\matrix{\node (a) {$a$} ; & \node (b) {$b$} ; \\ } ;}
9 \begin{document}
10 \section{tikzpicture}
11 \begin{tikzpicture}
12 \getstored[1]{tikz}
13 \end{tikzpicture}
14
15 \begin{tikzpicture}
16 \getstored[2]{tikz}
17 \end{tikzpicture}
18
19 \begin{tikzpicture}
20 \getstored[3]{tikz}
21 \end{tikzpicture}
22
23 \section{source}
24 \typestored[1]{tikz}
25 \typestored[2]{tikz}
26 \typestored[3]{tikz}
27 \end{document}

```

Example 4

Adapted from [Collecting contents of environment and store them for later retrieval](#) .

```

1 \documentclass{article}
2 \usepackage{scontents}
3 \pagestyle{empty}
4 \begin{document}
5 \begin{scontents}[store-env=a]
6 Something for a
7 \end{scontents}
8
9 \begin{scontents}[store-env=a]
10 Something for b
11 \end{scontents}
12
13 \begin{scontents}[store-env=a]
14 Something with no label
15 \end{scontents}
16
17 \textbf{Let's print them}
18
19 This is a: \getstored[1]{a}\par
20 This is b: \getstored[2]{a}
21
22 \textbf{Print all of them}\par
23 \foreachsc[sep={\\[10pt]}]{a}
24 \end{document}

```

Example 5

Adapted from [Collect contents of an environment \(that contains verbatim content\)](#) .

```

1 \documentclass{article}
2 \usepackage{scontents}
3 \pagestyle{empty}
4 \setlength{\parindent}{0pt}
5 \begin{document}
6 \section{Problem stated the first time}
7 \begin{scontents}[print-env=true,store-env=problem]
8 This is normal text.
9 \verb|This is from the verb command.|
10 \verb*|This is from the verb* command.|

```


```

11 This is normal text.
12 \begin{verbatim}
13 This is from the verbatim environment:
14 &{%{}}~
15 \end{verbatim}
16 \end{scontents}
17 \section{Problem restated}
18 \getstored[1]{problem}
19 \section{Problem restated once more}
20 \getstored[1]{problem}
21 \end{document}

```

7.4 Customization of verbatimsc

Example 6

Customization of `verbatimsc` using the `fancyvrb` and `tcolorbox` package .

```

1 \documentclass{article}
2 \usepackage{scontents}
3 \makeatletter
4 \let\verbatimsc@undefined
5 \let\endverbatimsc@undefined
6 \makeatother
7 \usepackage{fvextra}
8 \usepackage{xcolor}
9 \definecolor{mygray}{gray}{0.9}
10 \usepackage{tcolorbox}
11 \newenvironment{verbatimsc}%
12 {\VerbatimEnvironment
13 \begin{tcolorbox}[colback=mygray, boxsep=0pt, arc=0pt, boxrule=0pt]
14 \begin{Verbatim}[fontsize=\scriptsize, breaklines, breakafter=*, breaksymbolsep=0.5em,
15 breakaftersymbolpre={\,\tiny\ensuremath{\lfloor}}}%
16 {\end{Verbatim}}%
17 \end{tcolorbox}}
18 \setlength{\parindent}{0pt}
19 \pagestyle{empty}
20 \begin{document}
21 \section[Test \texttt{\textbackslash begin\{scontents\}} whit \texttt{fancyvrb}]
22 Test \verb+{scontents}+ \par
23
24 \begin{scontents}
25 Using \verb+scontents+ env no \verb+[key=val]+, save in seq \verb+contents+
26 with index 1.
27
28 Prove new \Verb*{ fancyvrb whit braces } and environment \verb+Verbatim*+
29 \begin{verbatim}
30 verbatim environment
31 \end{verbatim}
32 \end{scontents}
33
34 \section[Test \texttt{\textbackslash Scontents} whit \texttt{fancyvrb}]
35 \Scontents{ We have coded this in \LaTeX: $E=mc^2$..}
36
37 \section[Test \texttt{\textbackslash getstored}]
38 \getstored[1]{contents}\par
39 \getstored[2]{contents}
40
41 \section[Test \texttt{\textbackslash meanings}]
42 \meaningsc[1]{contents}\par
43 \meaningsc[2]{contents}
44
45 \section[Test \texttt{\textbackslash typestored}]
46 \typestored[1]{contents}
47 \typestored[2]{contents}
48 \end{document}

```

Example 7

Customization of `verbatimsc` using the `listings` package .

```

1 \documentclass{article}
2 \usepackage{scontents}
3 \makeatletter
4 \let\verbatimsc\@undefined
5 \let\endverbatimsc\@undefined
6 \makeatother
7 \usepackage{xcolor}
8 \usepackage{listings}
9 \lstnewenvironment{verbatimsc}
10 {
11   \lstset{
12     basicstyle=\small\ttfamily,
13     breaklines=true,
14     columns=fullflexible,
15     language=[LaTeX]TeX,
16     numbers=left,
17     numbersep=1em,
18     numberstyle=\tiny\color{gray},
19     keywordstyle=\color{red}
20   }
21 }{}
22 \setlength{\parindent}{0pt}
23 \pagestyle{empty}
24 \begin{document}
25 \section[Test \texttt{\textbackslash begin\{scontents\}} whit \texttt{listings}]
26 Test \verb+{scontents}+ \par
27
28 \begin{scontents}
29 Using \verb+scontents+ env no \verb+[key=val]+, save in seq \verb+contents+ with index 1.\par
30
31 Prove \lstinline[basicstyle=\ttfamily]| \lstinline | and environment \verb+Verbatim*+
32 \begin{verbatim}
33   verbatim environment
34 \end{verbatim}
35 \end{scontents}
36
37 \section[Test \texttt{\textbackslash Scontents*} whit \texttt{listings}]
38
39 \Scontents*+ We have coded this in \lstinline[basicstyle=\ttfamily]|\LaTeX:  $E=mc^2$ 
40 and more.+
41
42 \section[Test \texttt{\textbackslash getstored}]
43 \getstored[2]{contents}\par
44 \getstored[1]{contents}
45
46 \section[Test \texttt{\textbackslash typestored}]
47 \typestored[1]{contents}
48 \typestored[2]{contents}
49 \end{document}

```

Example 8

Customization of `verbatimsc` using the `minted` package .

```

1 % need --shell-escape
2 \documentclass{article}
3 \usepackage{scontents}
4 \makeatletter
5 \let\verbatimsc\@undefined
6 \let\endverbatimsc\@undefined
7 \makeatother
8 \usepackage{minted}
9 \newminted{tex}{linenos}
10 \newenvironment{verbatimsc}{\VerbatimEnvironment\begin{texcode}}{\end{texcode}}
11 \pagestyle{empty}
12 \begin{document}
13 \section[Test \texttt{\textbackslash begin\{scontents\}} whit \texttt{minted}]
14 Test \verb+{scontents}+ \par
15
16 \begin{scontents}[write-env=usingtab.tex,force-eol=true]

```

```

17 Using \verb+scontents+ env no \verb+[key=val]+, save in seq \verb+contents+
18 with index 1.\par
19
20 Prove new \Verb*{ new fvextra whit braces } and environment \verb+Verbatim*+
21 % Real TABs here :)
22 \begin{Verbatim}[obeytabs, showtabs, tab=\rightarrowfill, tabcolor=red,showspaces,
    spacecolor=blue]
23 No tab
24         One real tab
25             Two real Tab plus         one tab
26 \end{Verbatim}
27 \end{scontents}
28
29 \section{Test \texttt{\textbackslash Scontents} whit \texttt{minted}}
30
31 \Scontents{ We have coded \par this in \LaTeX:  $E=mc^2$ .}
32
33 \section{Test \texttt{\textbackslash getstored}}
34 \getstored[1]{contents}\par
35 \getstored[2]{contents}
36
37 \section{Test \texttt{\textbackslash typestored}}
38 \typestored[1]{contents}
39 \typestored[2]{contents}
40 \end{document}

```

8 Change history

In this section you will find some (not all) of the changes in `scontents` development, from the first public implementation using the `filecontentsdef` package to the current version with only `expl3`.

- v1.6 (ctan), 2019-10-26**
 - The internal behavior of `\getstored` has been modified.
 - The internal behavior of `\foreachsc` has been modified.
 - Corrected file extension for ConTeXt.
- v1.5 (ctan), 2019-10-24**
 - Add support for plain TeX and ConTeXt.
 - Split internal code for optimization.
 - Add support for vertical spaces in `key=val`.
 - Add `\foreachsc` command.
 - Check if `verbatim` package is loaded.
- v1.4 (ctan), 2019-10-03**
 - Add `store-all` key.
 - Messages and keys were separated.
 - Restructuring of documentation.
 - Now the version of `expl3` is checked instead of `xparse`.
 - The internal behavior of `force-eol` has been modified.
- v1.3 (ctan), 2019-09-24**
 - The environment can now nest.
 - Added `force-eol`, `verb-font` and `width-tab` keys.
 - The extra space has been removed when you run `\getstored`.
 - Internal code has been rewritten more efficiently.
 - Remove `\typestored`.
 - Remove `filecontentsdef` dependency.
 - Changing `\regex_replace_all:` for `\tl_replace_all:`.
- v1.2 (ctan), 2019-08-28**
 - Restructuring of documentation.
 - Added copy of `\tex_scantokens:`.
- v1.1 (ctan), 2019-08-12**
 - Extension of documentation.
 - Replace `\tex_endinput:D` by `\file_input_stop:`.
- v1.0 (ctan), 2019-07-30**
 - First public release,

9 Index of Documentation

C

Commands provide by `scontents`

<code>\Scontents*</code>	3, 4
<code>\Scontents</code>	2-4
<code>\cleanseqsc</code>	6
<code>\countsc</code>	6
<code>\endscontents</code>	3
<code>\foreachsc</code>	5
<code>\getstored</code>	3, 4
<code>\meaningsc</code>	2, 3, 6
<code>\scontents</code>	3
<code>\setupsc</code>	2, 4
<code>\startscontents</code>	3
<code>\stopscontents</code>	3
<code>\tpestored</code>	2, 3, 5

E

Environment provide by `scontents`:

<code>scontents</code>	2, 3
<code>verbatimsc</code>	5, 10, 11

Environments

<code>Verbatim</code>	2
<code>filecontentsdefmacro</code>	1
<code>lstlisting</code>	2

K

Keys

<code>after</code>	5
<code>before</code>	5
<code>force-eol</code>	3
<code>print-all</code>	2
<code>print-cmd</code>	4
<code>print-env</code>	4
<code>sep</code>	5
<code>start</code>	5
<code>step</code>	5
<code>stop</code>	5

<code>store-all</code>	2
<code>store-cmd</code>	4
<code>store-env</code>	4
<code>verb-font</code>	2
<code>width-tab</code>	3
<code>wrapper</code>	5
<code>write-env</code>	4
<code>write-out</code>	4

L

<code>\lstinline</code>	4
-------------------------	---

M

<code>\meaning</code>	6
-----------------------	---

P

Packages

<code>answers</code>	8
<code>expl3</code>	1, 7, 12
<code>fancyvrb</code>	2, 3, 5, 10
<code>filecontentsdef</code>	1, 2, 7, 8, 12
<code>fvextra</code>	4
<code>l3keys2e</code>	1
<code>l3seq</code>	1, 7
<code>listings</code>	2, 4, 6, 10
<code>minted</code>	5, 11
<code>scontents</code>	1, 2, 7, 12
<code>tcolorbox</code>	10
<code>verbatim</code>	5
<code>xparse</code>	1

S

<code>\setupsc</code>	3
-----------------------	---

V

<code>\Verb</code>	4
<code>\verb</code>	4

10 Implementation

10.1 Declaration of the package

First we set up the module name for l3doc:

```
1 <@@=scontents>
```

Now we define some common macros to hold the package date and version:

```
2 <loader>\def\SccontentsFileDate{2019-10-26}%
3 <core>\def\SccontentsCoreFileDate{2019-10-26}%
4 <*loader>
5 \def\SccontentsFileVersion{1.6}%
6 \def\SccontentsFileDescription{Stores LaTeX contents in memory or files}%
```

The \LaTeX loader is fairly simple: just load the dependencies, load the core code, and then set interfaces up.

We also check if the verbatim package is loaded and show a compatibility warning.

```
7 <*latex>
8 \RequirePackage{expl3,xparse,l3keys2e}[2019/05/28]
9 \ProvidesExplPackage
10   {scontents} {\SccontentsFileDate} {\SccontentsFileVersion} {\SccontentsFileDescription}
11   \@ifpackageloaded { verbatim }
12   {
13     \iow_term:n
14     {
15       The~implementation~of~the~'verbatimsc'~environment~\|
16       used~by~\tl_to_str:n{\typestored}~is~not~compatible~with~package\|
17       'verbatim'.~Review~the~documentation~and~redefine~\|
18       the~'verbatimsc'~environment.
19     }
20   } { }
21 </latex>
```

The Plain and Con \TeX t loaders are similar (probably because I don't know how to make a proper Con \TeX t module :-). We define a \LaTeX -style ver@scontents.sty macro with version info (just in case):

```
22 <*!latex>
23 <context>\writestatus{loading}{User Module scontents v\SccontentsFileVersion}
24 <context>\unprotect
25 \input expl3-generic.tex
26 \ExplSyntaxOn
27 \tl_gset:cx { ver @ scontents . sty } { \SccontentsFileDate\space
28   v\SccontentsFileVersion\space \SccontentsFileDescription }
29 \iow_log:x { Package: ~ scontents ~ \use:c { ver @ scontents . sty } }
30 </!latex>
```

In Plain, check that the package isn't being loaded twice (\LaTeX and Con \TeX t already defend against that):

```
31 <*plain>
32 \msg_gset:nnn { scontents } { already-loaded }
33 { The~'scontents'~package~is~already~loaded.~Aborting~input~\msg_line_context:. }
34 \cs_if_exist:NT \__scontents_rescan_tokens:n
35 {
36   \msg_warning:nn { scontents } { already-loaded }
37   \ExplSyntaxOff
38   \file_input_stop:
39 }
40 </plain>
```

```
\g__scontents_end_verbatimsc_tl
\c__scontents_end_env_tl
```

A token list to match when ending verbatimsc and scontents environments.

```
41 \tl_new:N \g__scontents_end_verbatimsc_tl
42 \tl_gset_rescan:Nnn
43   \g__scontents_end_verbatimsc_tl
44   {
45     \char_set_catcode_other:N \|
46 <*latex>
47     \char_set_catcode_other:N \|
48     \char_set_catcode_other:N \|
49 </latex>
```

```

50 }
51 <latex> { \end{verbatim} }
52 <plain> { \endverbatim }
53 <context> { \stopverbatim }
54 \tl_const:Nx \c__scontents_end_env_tl
55 {
56   \c_backslash_str
57 <latex|plain> end
58 <context> stop
59 <latex> \c_left_brace_str
60   scontents
61 <latex> \c_right_brace_str
62 }

```

(End definition for `\g__scontents_end_verbatimsc_tl` and `\c__scontents_end_env_tl`.)

Now we load the core `scontents` code:

```

63 \file_input:n { scontents-code.tex }

```

Sometimes we need to detect the format from within a macro:

```

64 \cs_new:Npn \__scontents_format_case:nnn #1 #2 #3
65 <latex> {#1} % LaTeX
66 <plain> {#2} % Plain/Generic
67 <context> {#3} % ConTeXt

```

Checking that the package was loaded with the proper loader code. This code was copied from `expl3-code.tex`.

```

68 </loader>
69 <*core>
70 \begingroup
71   \def\next{\endgroup}%
72   \expandafter\ifx\csname PackageError\endcsname\relax
73     \begingroup
74       \def\next{\endgroup\endgroup}%
75       \def\PackageError#1#2#3%
76         {%
77           \endgroup
78           \errhelp{#3}%
79           \errmessage{#1 Error: #2!}%
80         }%
81   \fi
82   \expandafter\ifx\csname ScontentsFileDate\endcsname\relax
83     \def\next
84       {%
85         \PackageError{scontents}{No scontents loader detected}
86         {%
87           You have attempted to use the scontents code directly rather than using
88           the correct loader. Loading of scontents will abort.
89         }%
90       \endgroup
91     \endinput
92   }
93   \else
94     \ifx\ScontentsFileDate\ScontentsCoreFileDate
95     \else
96       \def\next
97         {%
98           \PackageError{scontents}{Mismatched~scontents~files~detected}
99           {%
100             You~have~attempted~to~load~scontents~with~mismatched~files:~
101             probably~you~have~one~or~more~files~'locally~installed'~which~
102             are~in~conflict.~Loading~of~scontents~will~abort.
103           }%
104         \endgroup
105       \endinput
106     }%
107   \fi
108 \fi
109 \next

```

10.2 Definition of common keys

We create some common *(keys)* that will be used by the options passed to the package as well as by the environments and commands defined.

```

110 \keys_define:nn { scontents }
111 {
112   store-env .tl_set:N      = \l__scontents_name_seq_env_tl,
113   store-env .initial:n     = contents,
114   store-env .value_required:n = true,
115   store-cmd .tl_set:N      = \l__scontents_name_seq_cmd_tl,
116   store-cmd .initial:n     = contents,
117   store-cmd .value_required:n = true,
118   verb-font .tl_set:N      = \l__scontents_verb_font_tl,
119   verb-font .value_required:n = true,
120   print-env .bool_set:N    = \l__scontents_print_env_bool,
121   print-env .initial:n     = false,
122   print-env .default:n     = true,
123   print-cmd .bool_set:N    = \l__scontents_print_cmd_bool,
124   print-cmd .initial:n     = false,
125   print-cmd .default:n     = true,
126   force-eol .bool_set:N    = \l__scontents_forced_eol_bool,
127   force-eol .initial:n     = false,
128   force-eol .default:n     = true,
129   width-tab .int_set:N     = \l__scontents_tab_width_int,
130   width-tab .initial:n     = 1,
131   width-tab .value_required:n = true,
132   print-all .meta:n       = { print-env = #1 , print-cmd = #1 },
133   print-all .default:n     = true,
134   store-all .meta:n       = { store-env = #1 , store-cmd = #1 },
135   store-all .value_required:n = true
136 }
137 </core>
138 <loader>\keys_define:nn { scontents }
139 <latex> { verb-font .initial:n = \ttfamily }
140 <plain | context> { verb-font .initial:n = \tt }

```

In \LaTeX mode we load `l3keys2e` process the *(keys)* as options passed on to the package, the package `l3keys2e` will verify the *(keys)* and will return an error when they are *unknown*.

```

141 <latex>\ProcessKeysOptions { scontents }
142 <*core>

```

10.3 Internal variables

Now we declare the internal variables we will use.

```

\l__scontents_macro_tmp_tl  \l__scontents_macro_tmp_tl is a temporary token list to hold the contents of the macro/environment,
\l__scontents_fname_out_tl  \l__scontents_fname_out_tl is used as the name of the output file, when there's one, \l__-
\l__scontents_temp_tl       scontents_file_tl holds the contents of an environment as it's being read, and \l__scontents_-
\l__scontents_file_tl       temp_tl and \g__scontents_temp_tl are generic temporary token lists.
\g__scontents_temp_tl
\l__scontents_foreach_name_seq_tl \l__scontents_foreach_name_seq_tl is the name assigned to the sequence on which the loop will be
\l__scontents_foreach_name_seq_tl made, \l__scontents_foreach_before_tl and \l__scontents_foreach_after_tl are token lists in
\l__scontents_foreach_before_tl which the assigned material will be placed before and after the execution of the \foreachsc loop.
\l__scontents_foreach_after_tl

```

```

143 \tl_new:N \l__scontents_macro_tmp_tl
144 \tl_new:N \l__scontents_fname_out_tl
145 \tl_new:N \l__scontents_temp_tl
146 \tl_new:N \l__scontents_file_tl
147 \tl_new:N \g__scontents_temp_tl
148 \tl_new:N \l__scontents_foreach_name_seq_tl
149 \tl_new:N \l__scontents_foreach_before_tl
150 \tl_new:N \l__scontents_foreach_after_tl

```

(End definition for `\l__scontents_macro_tmp_tl` and others.)

```

\l__scontents_seq_item_int  \l__scontents_seq_item_int stores the index in the sequence of the item requested to \typestored
\l__scontents_env_nesting_int or \meaningsc. \l__scontents_env_nesting_int stores the current nesting level of the scontents
\l__scontents_tmpa_int      environment. \l__scontents_foreach_stop_int will save the value at which the \foreachsc loop
\l__scontents_foreach_stop_int will stop.

```

```

151 \int_new:N \l__scontents_foreach_stop_int
152 \int_new:N \l__scontents_seq_item_int
153 \int_new:N \l__scontents_env_nesting_int
154 \int_new:N \l__scontents_tmpa_int

```

(End definition for `\l__scontents_seq_item_int` and others.)

`\l__scontents_writing_bool` The boolean `\l__scontents_writing_bool` keeps track of whether we should write to a file, and `\l__scontents_storing_bool` determines whether it is in write-only mode when the `write-out` option is used.

```

155 \bool_new:N \l__scontents_writing_bool
156 \bool_set_false:N \l__scontents_writing_bool
157 \bool_new:N \l__scontents_storing_bool
158 \bool_set_true:N \l__scontents_storing_bool

```

(End definition for `\l__scontents_writing_bool` and `\l__scontents_storing_bool`.)

`\l__scontents_foreach_before_bool` Boolean variables used by the `\foreachsc` loop.

`\l__scontents_foreach_stop_bool`
`\l__scontents_foreach_wrapper_bool`

```

159 \bool_new:N \l__scontents_foreach_before_bool
160 \bool_set_false:N \l__scontents_foreach_before_bool
161 \bool_new:N \l__scontents_foreach_after_bool
162 \bool_set_false:N \l__scontents_foreach_after_bool
163 \bool_new:N \l__scontents_foreach_stop_bool
164 \bool_set_false:N \l__scontents_foreach_stop_bool
165 \bool_new:N \l__scontents_foreach_wrapper_bool
166 \bool_set_false:N \l__scontents_foreach_wrapper_bool

```

(End definition for `\l__scontents_foreach_before_bool`, `\l__scontents_foreach_after_bool`, `\l__scontents_foreach_stop_bool`, and `\l__scontents_foreach_wrapper_bool`.)

`\l__scontents_`

`\l__scontents_foreach_print_seq` The `\l__scontents_foreach_print_seq` is the sequence used by `\foreachsc`.

```

167 \seq_new:N \l__scontents_foreach_print_seq

```

(End definition for `\l__scontents_foreach_print_seq`.)

`\c__scontents_hidden_space_str` `\c__scontents_hidden_space_str` is a constant *string* to used to hide the *forced space* added by T_EX when recording content in a macro. This *string* contains the *reserved phrase* “`^^Ascheol%`” which is added to the end of the argument stored in `seq` when the key `force-eol` is false.

```

168 \str_const:Nx \c__scontents_hidden_space_str
169 { \c_percent_str \c_circumflex_str \c_circumflex_str A scheol \c_percent_str }

```

(End definition for `\c__scontents_hidden_space_str`.)

`\q__scontents_stop` Some quarks used along the code as macro delimiters.
`\q__scontents_mark`

```

170 \quark_new:N \q__scontents_stop
171 \quark_new:N \q__scontents_mark

```

(End definition for `\q__scontents_stop` and `\q__scontents_mark`.)

`\l__scontents_file_iow` An output stream for saving the contents of an environment to a file.

```

172 \iow_new:N \l__scontents_file_iow

```

(End definition for `\l__scontents_file_iow`.)

`__scontents_rescan_tokens:n` `\tl_rescan:n` doesn't fit the needs of this package because it does not allow catcode changes inside the argument, so verbatim commands used inside one of `scontents`'s commands/environments will not work. Here we create a private copy of `\tex_scantokens:D` which will serve our purposes.

```

173 \cs_new_protected:Npn \__scontents_rescan_tokens:n #1 { \tex_scantokens:D {#1} }
174 \cs_generate_variant:Nn \__scontents_rescan_tokens:n { V, x }

```

(End definition for `__scontents_rescan_tokens:n`.)

`__scontents_tab:` Control sequences to replace tab (`^I`) and form feed (`^L`) characters.
`__scontents_par:`

```
175 \cs_new:Npx \__scontents_tab: { \c_space_tl }
176 \cs_new:Npn \__scontents_par: { ^^J ^^J }
```

(End definition for `__scontents_tab:` and `__scontents_par:`.)

`\tl_remove_once:NV` Some nonstandard variants.

```
\tl_replace_all:Nxx
\tl_replace_all:Nxn
\tl_replace_all:Nnx
\tl_if_empty:FTF
177 \cs_generate_variant:Nn \tl_remove_once:Nn { NV }
178 \cs_generate_variant:Nn \tl_replace_all:Nnn { Nx, Nxx, Nnx }
179 \cs_generate_variant:Nn \msg_error:nnnn { nnx }
180 \prg_generate_conditional_variant:Nnn \tl_if_empty:n { f } { TF }
```

(End definition for `\tl_remove_once:NV`, `\tl_replace_all:Nxx`, and `\tl_if_empty:FTF`.)

10.4 Defining keys for the environment and commands

We add the `<keys>` divided into subgroups to handle errors and *unknown* `<keys>` separately.

10.4.1 Keys for environment `scontents`

We define a set of `<keys>` for environment `scontents`.

```
181 \keys_define:nn { scontents / scontents }
182 {
183   write-env .code:n = {
184     \bool_set_true:N \__scontents_writing_bool
185     \tl_set:Nn \__scontents_fname_out_tl {#1}
186   },
187   write-out .code:n = {
188     \bool_set_false:N \__scontents_storing_bool
189     \bool_set_true:N \__scontents_writing_bool
190     \tl_set:Nn \__scontents_fname_out_tl {#1}
191   },
192   write-env .value_required:n = true,
193   write-out .value_required:n = true,
194   print-env .meta:nn = { scontents } { print-env = #1 },
195   print-env .default:n = true,
196   store-env .meta:nn = { scontents } { store-env = #1 },
197   force-eol .meta:nn = { scontents } { force-eol = #1 },
198   force-eol .default:n = true,
199   unknown .code:n = { \__scontents_parse_environment_keys:n {#1} }
200 }
```

10.4.2 Keys for command `\Scontents`

We define a set of `<keys>` for commands `\Scontents` and `\Scontents*`.

```
201 \keys_define:nn { scontents / Scontents }
202 {
203   print-cmd .meta:nn = { scontents } { print-cmd = #1 },
204   print-cmd .default:n = true,
205   store-cmd .meta:nn = { scontents } { store-cmd = #1 },
206   force-eol .meta:nn = { scontents } { force-eol = #1 },
207   force-eol .default:n = true,
208   unknown .code:n = { \__scontents_parse_command_keys:n {#1} }
209 }
```

10.4.3 Keys for command `\foreachsc`

We define a set of `<keys>` for command `\foreachsc`.

```
210 \keys_define:nn { scontents / foreachsc }
211 {
212   before .code:n = {
213     \bool_set_true:N \__scontents_foreach_before_bool
214     \tl_set:Nn \__scontents_foreach_before_tl {#1}
215   },
216   before .value_required:n = true,
217   after .code:n = {
218     \bool_set_true:N \__scontents_foreach_after_bool
```

```

219                                     \tl_set:Nn \__scontents_foreach_after_tl {#1}
220                                     },
221   after   .value_required:n = true,
222   start   .int_set:N         = \__scontents_foreach_start_int,
223   start   .value_required:n = true,
224   start   .initial:n         = 1,
225   stop    .code:n            = {
226                                     \bool_set_true:N \__scontents_foreach_stop_bool
227                                     \int_set:Nn \__scontents_foreach_stop_int {#1}
228                                     },
229   stop    .value_required:n = true,
230   step    .int_set:N         = \__scontents_foreach_step_int,
231   step    .value_required:n = true,
232   step    .initial:n         = 1,
233   wrapper .code:n            = {
234                                     \bool_set_true:N \__scontents_foreach_wrapper_bool
235                                     \cs_set_protected:Npn \__scontents_foreach_wrapper:n ##1 {#1}
236                                     },
237   wrapper .value_required:n = true,
238   sep     .tl_set:N          = \__scontents_foreach_sep_tl,
239   sep     .initial:n         = {},
240   sep     .value_required:n = true,
241   unknown .code:n            = { \__scontents_parse_foreach_keys:n {#1} }
242 }

```

10.4.4 Key for commands \typestored and \meaningsc

We define a *key* for command `\typestored` and `\meaningsc`. Both commands accept the same type of optional arguments, just define a common *key*.

```

243 \keys_define:nn { scontents / typemeaning }
244 {
245   width-tab .meta:nn = { scontents } { width-tab = #1 },
246   unknown   .code:n   = { \__scontents_parse_type_meaning_key:n {#1} }
247 }

```

10.5 Handling undefined keys

The *keys* are stored in the token list variable `\l_keys_key_tl`, and the value (if any) is passed as an argument to each *function*.

10.5.1 Undefined keys for environment scontents

We check the *keys* passed to the environment `scontents` and process it with `__scontents_parse_environment_keys:n` if the *key* is *unknown* we return an error message.

```

248 \cs_new_protected:Npn \__scontents_parse_environment_keys:n #1
249 { \exp_args:NV \__scontents_parse_environment_keys:nn \l_keys_key_tl {#1} }
250 \cs_new_protected:Npn \__scontents_parse_environment_keys:nn #1#2
251 {
252   \tl_if_blank:nTF {#2}
253   { \msg_error:nnn { scontents } { env-key-unknown } {#1} }
254   { \msg_error:nnnn { scontents } { env-key-value-unknown } {#1} {#2} }
255 }

```

(End definition for `__scontents_parse_environment_keys:n` and `__scontents_parse_environment_keys:nn`.)

10.5.2 Undefined keys for \Scontents and \Scontents*

We check the *keys* passed to commands `\Scontents` or `\Scontents*` and process it with `__scontents_parse_command_keys:n` if the *key* is *unknown* we return an error message.

```

256 \cs_new_protected:Npn \__scontents_parse_command_keys:n #1
257 { \exp_args:NV \__scontents_parse_command_keys:nn \l_keys_key_tl {#1} }
258 \cs_new_protected:Npn \__scontents_parse_command_keys:nn #1#2
259 {
260   \tl_if_blank:nTF {#2}
261   { \msg_error:nnn { scontents } { cmd-key-unknown } {#1} }
262   { \msg_error:nnnn { scontents } { cmd-key-value-unknown } {#1} {#2} }
263 }

```

(End definition for `__scontents_parse_command_keys:n` and `__scontents_parse_command_keys:nn`.)

10.5.3 Undefined keys for `\foreachsc`

`__scontents_parse_foreach_keys:n`
`__scontents_parse_foreach_keys:nn`

We check the $\langle key \rangle$ passed to command `\foreachsc` and process it with `__scontents_parse_foreach_keys:n`, if the $\langle key \rangle$ is *unknown* we return an error message.

```

264 \cs_new_protected:Npn \__scontents_parse_foreach_keys:nn #1#2
265 {
266   \tl_if_blank:nTF {#2}
267     { \msg_error:nnn { scontents } { for-key-unknown } {#1} }
268     { \msg_error:nnnn { scontents } { for-key-value-unknown } {#1} {#2} }
269   }
270 \cs_new_protected:Npn \__scontents_parse_foreach_keys:n #1
271   { \exp_args:NV \__scontents_parse_foreach_keys:nn \l_keys_key_tl {#1} }

```

(End definition for `__scontents_parse_foreach_keys:n` and `__scontents_parse_foreach_keys:nn`.)

10.5.4 Undefined keys for `\typestored` and `\meaningsc`

`__scontents_parse_type_meaning_key:n`
`__scontents_parse_type_meaning_key:nn`

The commands `\typestored` and `\meaningsc` accept an optional argument for setting the `width-tab` to print the stored contents. However their optional argument also contains the number of the item to retrieve from the stored sequence. To avoid the awkward `\typestored[][\langle options \rangle]{...}` syntax, we'll make the commands have a single optional argument which is processed by `l3keys`, and the unknown keys are brought here to `__scontents_parse_type_meaning_key:n` to process.

First we check if the $\langle key \rangle$ is an integer using `\int_to_roman:n`. If it is, we check that the value passed to the key is blank (otherwise something odd as `1=1` might have been used). If everything is correct, then set the value of the integer which holds the $\langle index \rangle$. Otherwise raise an error about an *unknown* option.

```

272 \cs_new_protected:Npn \__scontents_parse_type_meaning_key:n #1
273   { \exp_args:NV \__scontents_parse_type_meaning_key:nn \l_keys_key_tl {#1} }
274 \cs_new_protected:Npn \__scontents_parse_type_meaning_key:nn #1#2
275   {
276     \tl_if_empty:ftF { \int_to_roman:n { -0 #1 } }
277     {
278       \tl_if_blank:nTF {#2}
279         { \int_set:Nn \__scontents_seq_item_int {#1} }
280         { \msg_error:nnnn { scontents } { type-key-value-unknown } {#1} {#2} }
281       }
282     {
283       \tl_if_blank:nTF {#2}
284         { \msg_error:nnn { scontents } { type-key-unknown } {#1} }
285         { \msg_error:nnnn { scontents } { type-key-value-unknown } {#1} {#2} }
286       }
287     }

```

(End definition for `__scontents_parse_type_meaning_key:n` and `__scontents_parse_type_meaning_key:nn`.)

10.6 Compatibility layer with Plain

When loading the package outside of \LaTeX we can't usually use `xparse`. However since `xparse` doesn't actually hold any dependency with \LaTeX except for package-loading commands, we can emulate those commands (much like in `miniltx`) so that `xparse` is loadable in any format.

The bunch of macros below is adapted from the \LaTeX kernel (greatly simplified).

```

288 </core>
289 <*loader&!latex>
290 \seq_new:N \__scontents_compat_seq
291 \cs_new_protected:Npn \__scontents_compat_redefine:Npn #1
292   {
293     \seq_put_right:Nn \__scontents_compat_seq {#1}
294     \cs_set_eq:cN { __scontents_saved_\cs_to_str:N #1: } #1
295     \cs_new_protected:Npn #1
296   }
297 \cs_new_protected:Npn \__scontents_compat_restore:
298   { \seq_map_function:NN \__scontents_compat_seq \__scontents_compat_restore:N }
299 \cs_new_protected:Npn \__scontents_compat_restore:N #1
300   {
301     \cs_set_eq:Nc #1 { __scontents_saved_\cs_to_str:N #1: }
302     \cs_undefine:c { __scontents_saved_\cs_to_str:N #1: }
303   }
304 \cs_generate_variant:Nn \__scontents_compat_redefine:Npn { c }

```

```

305 \cs_new_protected:Npn \__scontents_optarg:nn #1 #2
306   { \peek_charcode_ignore_spaces:NTF [ {#1} {#1[#2]} ] }
307 \cs_new_protected:Npn \__scontents_stararg:nn #1 #2
308   { \peek_charcode_remove_ignore_spaces:NTF * {#1} {#2} }
309 \__scontents_compat_redefine:Npn \RequirePackage
310   { \__scontents_optarg:nn { \__scontents_require_auxi:wn } { } }
311 \cs_new_protected:Npn \__scontents_require_auxi:wn [#1] #2
312   { \__scontents_optarg:nn { \__scontents_require_auxii:wnw [{#1}]{#2} } { } }
313 \cs_new:Npn \__scontents_zap_space:ww #1~#2
314   {
315     #1 \if_meaning:w #2 \q_mark
316     \exp_after:wN \use_none:n
317     \else:
318       \exp_after:wN \__scontents_zap_space:ww
319     \fi: #2
320   }
321 \cs_new_protected:Npn \__scontents_require_auxii:wnw [#1] #2 [#3]
322   {
323     \tl_set:Nx \l__scontents_temp_tl { \__scontents_zap_space:ww #2 ~ \q_mark }
324     \clist_map_function:NN \l__scontents_temp_tl \__scontents_require_auxiii:n
325   }
326 \cs_new_protected:Npn \__scontents_require_auxiii:n #1
327   { \str_if_eq:eeF {expl3} {#1} { \msg_error:nnn { scontents } { invalid-package } {#1} } }
328 \msg_new:nnn { scontents } { invalid-package }
329   { Package~‘#1’~invalid~in~scontents.~This~is~an~error~in~scontents. }
330 \__scontents_compat_redefine:cpn { @ifpackagelater } #1
331   { \exp_args:Nc \__scontents_package_later_aux:Nn { ver@#1.sty } }
332 \cs_new_protected:Npn \__scontents_package_later_aux:Nn #1 #2
333   {
334     \int_compare:nNnTF
335       { \exp_after:wN \__scontents_parse_version:w #1 //00 \q_mark } <
336       { \exp_after:wN \__scontents_parse_version:w #2 //00 \q_mark }
337   }
338 \cs_new:Npn \__scontents_parse_version:w #1 { \__scontents_parse_version_auxi:w 0#1 }
339 \cs_new:Npn \__scontents_parse_version_auxi:w #1/#2/#3#4#5 \q_mark
340   { \__scontents_parse_version_auxii:w #1-#2-#3#4 \q_mark }
341 \cs_new:Npn \__scontents_parse_version_auxii:w #1-#2-#3#4#5 \q_mark
342   { \tl_if_blank:nF {#2} {#1} #2 #3 #4 }
343 \__scontents_compat_redefine:Npn \ProvidesExplPackage #1 #2 #3 #4
344   { \__scontents_provides_aux:nn {#1} { #2 \ifx\relax#3\relax\else v#3\space\fi #4 } }
345 \cs_new_protected:Npn \__scontents_provides_aux:nn #1 #2
346   {
347     \tl_gset:cx { ver@#1.sty } {#2}
348     \iow_log:n { Package~#1:~#2 }
349     \ExplSyntaxOn
350   }
351 \__scontents_compat_redefine:Npn \DeclareOption
352   { \__scontents_stararg:nn { \use_none:n } { \use_none:nn } }
353 \__scontents_compat_redefine:Npn \ProcessOptions
354   { \__scontents_stararg:nn { } { } }

```

Now that the compatibility layer is defined, we can finally load xparse. xparse expects to be loaded with `\ExplSyntaxOff` (not much harm would be done otherwise, but just to be on the safe side).

Within xparse a `\RequirePackage{expl3}` is done. We can ignore that since we have already loaded `expl3`. Next, a `\@ifpackagelater` test is done: we do that test too to ensure that xparse is compatible with the current running version of `expl3`. The following `\ProvidesExplPackage` simply defines `\ver@xparse.sty` for any other package that might use it, and then does `\ExplSyntaxOn`. At the end of the package, xparse parses (heh) the package options. Since we don’t have those in non- \TeX formats, they are ignored. Okay, so load xparse:

```

355 \int_set:Nn \l__scontents_tmpa_int { \char_value_catcode:n { '\@ } }
356 \char_set_catcode_letter:N \@
357 \exp_after:wN
358 \ExplSyntaxOff
359 \file_input:n { xparse.sty }
360 \ExplSyntaxOn
361 \char_set_catcode:nn { '\@ } { \l__scontents_tmpa_int }
362 \__scontents_compat_restore:
363 </loader&!latex>
364 <*core>

```

(actually we don't need to do `\ExplSyntaxOn` there because we don't have \TeX 's full package loading mechanism, so the `expl3` syntax remains active after `xparse` is loaded, but it doesn't harm either).

10.7 Programming of the sequences

The storage of the package is done using `seq` variables. Here we set up the macros that will manage the variables.

`__scontents_append_contents:nn` `__scontents_append_contents:nn` creates a `seq` variable if one didn't exist and appends the contents in the argument to the right of the sequence.

```

365 \cs_new_protected:Npn \__scontents_append_contents:nn #1#2
366 {
367   \tl_if_blank:Nt {#1}
368   { \msg_error:nn { scontents } { empty-store-cmd } }
369   \seq_if_exist:cF { g__scontents_name_#1_seq }
370   { \seq_new:c { g__scontents_name_#1_seq } }
371   \seq_gput_right:cn { g__scontents_name_#1_seq } {#2}
372 }
373 \cs_generate_variant:Nn \__scontents_append_contents:nn { Vx }

```

(End definition for `__scontents_append_contents:nn`.)

`__scontents_getfrom_seq:nn` `__scontents_getfrom_seq:nn` retrieves the saved item from the sequence.

```

374 \cs_new:Npn \__scontents_getfrom_seq:nn #1#2
375 {
376   \seq_if_exist:cTF { g__scontents_name_#2_seq }
377   {
378     \exp_args:Nf \__scontents_getfrom_seq:nnn
379     { \seq_count:c { g__scontents_name_#2_seq } }
380     {#1} {#2}
381   }
382   { \msg_expandable_error:nnn { scontents } { undefined-storage } {#2} }
383 }
384 \cs_new:Npn \__scontents_getfrom_seq:nnn #1#2#3
385 {
386   \bool_lazy_or:nnTF
387   { \int_compare_p:nNn {#2} = { 0 } }
388   { \int_compare_p:nNn { \int_abs:n {#2} } > {#1} }
389   { \msg_expandable_error:nnnn { scontents } { index-out-of-range } {#2} {#3} {#1} }
390   { \seq_item:cn { g__scontents_name_#3_seq } {#2} }
391 }

```

(End definition for `__scontents_getfrom_seq:nn` and `__scontents_getfrom_seq:nnn`.)

`__scontents_lastfrom_seq:n` `__scontents_lastfrom_seq:n` retrieves the last saved item from the sequence when `\l__scontents_print_env_bool` or `\l__scontents_print_cmd_bool` is true.

```

392 \cs_new_protected:Npn \__scontents_lastfrom_seq:n #1
393 {
394   \tl_gset:Nx \g__scontents_temp_tl { \seq_item:cn { g__scontents_name_#1_seq } {-1} }
395   \group_insert_after:N \__scontents_rescan_tokens:V
396   \group_insert_after:N \g__scontents_temp_tl
397   \group_insert_after:N \tl_gclear:N
398   \group_insert_after:N \g__scontents_temp_tl
399 }

```

(End definition for `__scontents_lastfrom_seq:n`.)

`__scontents_store_to_seq:NN` The `__scontents_store_to_seq:NN` writes the recorded contents in `#1` to the log and stores it in `#2`.

```

400 \cs_new_protected:Npn \__scontents_store_to_seq:NN #1#2
401 {
402   \tl_log:N #1
403   \__scontents_append_contents:Vx #2 { \exp_not:V #1 }
404 }

```

(End definition for `__scontents_store_to_seq:NN`.)

10.8 Construction of environment scontents

We define the environment scontents, next to the system [$\langle key = val \rangle$]. The environment is divided into two parts.

```

scontents This is the main environment used in the document.
\contents
\startcontents 405 </core>
\endscontents 406 <*loader>
\stopscontents 407 <!context>\NewDocumentEnvironment { scontents } { }
408 <context>\cs_new_protected:Npn \startcontents
409 {
410   <plain | context> \group_begin:
411     \__scontents_scontents_env_begin:
412   }
413 <context>\cs_new_protected:Npn \stopscontents
414 {
415   \__scontents_scontents_env_end:
416   <plain | context> \group_end:
417 }
418 </loader>
419 <*core>
420 \cs_new_protected:Npn \__scontents_scontents_env_begin:
421 {
422   \char_set_catcode_active:N ^^M
423   \__scontents_start_environment:w
424 }
425 \cs_new_protected:Npn \__scontents_scontents_env_end:
426 {
427   \__scontents_stop_environment:
428   \__scontents_atend_environment:
429 }
```

(End definition for scontents and others. These functions are documented on page 3.)

10.8.1 key val for environment

Define a [$\langle key = val \rangle$] for environment scontents

The macro __scontents_grab_optional:w is called from the scontents environment with the tokens following the \begin{scontents} when the next character is a [. This function is defined using xparse to exploit its delimited argument processor.

The function is called from a context where ^^M is active, so __scontents_normalise_line_ends:N is used to replace active ^^M characters by spaces.

```

430 </core>
431 <*loader>
432 \NewDocumentCommand \__scontents_grab_optional:w { r[] }
433 { \__scontents_grab_optional:n {#1} }
434 </loader>
435 <*core>
436 \cs_new_protected:Npn \__scontents_grab_optional:n #1
437 {
438   \tl_if_novalue:nF {#1}
439   {
440     \tl_set:Nn \__scontents_temp_tl {#1}
441     \__scontents_normalise_line_ends:N \__scontents_temp_tl
442     \keys_set:nV { scontents / scontents } \__scontents_temp_tl
443   }
444   \__scontents_start_after_option:w
445 }
```

(End definition for __scontents_grab_optional:n and __scontents_grab_optional:w.)

10.8.2 The environment itself

Here we make ^^I, ^^L and ^^M active characters so that the end of line can be “seen” to be used as a delimiter, and T_EX doesn’t try to eliminate space-like characters.

First we check if the immediate next token after \begin{scontents} is a [. If it is, then __scontents_grab_optional:w is called to do the heavy lifting. __scontents_grab_optional:w processes the optional argument and calls __scontents_start_after_option:w.

`__scontents_start_after_option:w` also checks for trailing tokens after the optional argument and issues an error if any.

In all cases `__scontents_check_line_and_process:xn` is called to check that the rest of the `\begin{scontents}` is empty and then process the environment. `__scontents_check_line_and_process:xn` calls the `__scontents_file_tl_write_start:V` function, which will then read the contents of the environment and optionally store them in a token list or write to an external file.

When that's done, `__scontents_file_write_stop:N` does the cleanup and the read token list is smuggled out of the verbatim group. This part of the code is inspired and adapted from the code of the package `xsimverb` by Clemens Niederberger.

```

446 \group_begin:
447   \char_set_catcode_active:N \^^I
448   \char_set_catcode_active:N \^^L
449   \char_set_catcode_active:N \^^M
450   \cs_new_protected:Npn \__scontents_normalise_line_ends:N #1
451     { \tl_replace_all:Nnn #1 { ^^M } { ~ } }
452   \cs_new_protected:Npn \__scontents_start_environment:w #1 ^^M
453     {
454       \tl_if_head_is_N_type:nTF {#1}
455       {
456         \str_if_eq:eeTF { \tl_head:n {#1} } { [ ] }
457         { \__scontents_grab_optional:w #1 ^^M }
458         { \__scontents_check_line_and_process:xn { } {#1} }
459       }
460       { \__scontents_check_line_and_process:xn { } {#1} }
461     }
462   \cs_new_protected:Npn \__scontents_start_after_option:w #1 ^^M
463     { \__scontents_check_line_and_process:xn { [...] } {#1} }
464   \cs_new_protected:Npn \__scontents_check_line_and_process:xn #1 #2
465     {
466       \tl_if_blank:nF {#2}
467       {
468         \msg_error:nnxn { scontents } { junk-after-begin }
469         { after~\c_backslash_str begin{scontents} #1 } {#2}
470       }
471       \__scontents_make_control_chars_active:
472       \group_begin:
473       \__scontents_file_tl_write_start:V \__scontents_fname_out_tl
474     }
475   \cs_new_protected:Npn \__scontents_stop_environment:
476     {
477       \__scontents_file_write_stop:N \__scontents_macro_tmp_tl
478       \exp_args:NNNV
479       \group_end:
480       \tl_set:Nn \__scontents_macro_tmp_tl \__scontents_macro_tmp_tl
481       \tl_if_empty:NT \__scontents_macro_tmp_tl
482       { \msg_warning:nnn { scontents } { empty-environment } }
483     }

```

(End definition for `__scontents_start_environment:w` and others.)

```

\__scontents_file_tl_write_start:n
\__scontents_file_tl_write_start:V
\__scontents_verb_processor_iterate:w
\__scontents_file_write_stop:N
\__scontents_remove_leading_nl:n
\__scontents_remove_leading_nl:w

```

This is the main macro to collect the contents of a verbatim environment. The macro starts a group, opens the output file, if necessary, sets verbatim catcodes, and then issues `^^M` (set equal to `__scontents_ret:w`) to read the environment line by line until reaching its end. The output token list will be appended with an active `^^J` character and the line just read, and this line is written to the output file, if any. At the end of the environment the output file is closed (if it was open), and the output token list is smuggled out of the verbatim group. A leading `^^J` is removed from the token list using `__scontents_remove_leading_nl:n` (which expects an active `^^J` token at the head of the token list; a low level T_EX error is raised otherwise).

```

484   \cs_new_protected:Npn \__scontents_file_tl_write_start:n #1
485     {
486       \group_begin:
487       \bool_if:NT \__scontents_writing_bool
488       {
489         \file_if_exist:nTF {#1}
490         { \msg_warning:nnx { scontents } { rewriting-file } {#1} }
491         { \msg_warning:nnx { scontents } { writing-file } {#1} }

```

```

492         \iow_open:Nn \l__scontents_file_iow {#1}
493     }
494     \tl_clear:N \l__scontents_file_tl
495     \seq_map_function:NN \l_char_special_seq \char_set_catcode_other:N
496     \int_step_function:nnN { 128 } { 255 } \char_set_catcode_letter:n
497     \cs_set_protected:Npx \__scontents_ret:w ##1 ^^M
498     {
499         \exp_not:N \__scontents_verb_processor_iterate:w
500         ##1 \c__scontents_end_env_tl
501         \c__scontents_end_env_tl
502         \exp_not:N \q__scontents_stop
503     }
504     \__scontents_make_control_chars_active:
505     \__scontents_ret:w
506 }
507 \use:x
508 {
509     \cs_new:Npn \exp_not:N \__scontents_verb_processor_iterate:w
510     ##1 \c__scontents_end_env_tl
511     ##2 \c__scontents_end_env_tl
512     ##3 \exp_not:N \q__scontents_stop
513 } {
514     \tl_if_blank:nTF {#3}
515     {
516         \__scontents_analyse_nesting:n {#1}
517         \__scontents_verb_processor_output:n {#1}
518     }
519     {
520         \__scontents_if_nested:TF
521         {
522             \__scontents_nesting_decr:
523             \__scontents_verb_processor_output:x
524             { \exp_not:n {#1} \c__scontents_end_env_tl \exp_not:n {#2} }
525         }
526         {
527             \tl_if_blank:nF {#1}
528             { \__scontents_verb_processor_output:n {#1} }
529             \cs_set_protected:Npx \__scontents_ret:w
530             {
531                 \__scontents_format_case:nnn
532                 { \exp_not:N \end{scontents} } % LaTeX
533                 { \endscontents } % Plain/Generic
534                 { \stopscontents } % ConTeXt
535                 \bool_lazy_or:nnF
536                 { \tl_if_blank_p:n {#2} }
537                 { \str_if_eq_p:ee {#2} { \c_percent_str } }
538                 {
539                     \str_if_eq:VnF \c__scontents_hidden_space_str {#2}
540                     { \msg_warning:nnn { scontents } { rescanning-text } {#2} }
541                     \__scontents_rescan_tokens:n {#2}
542                 }
543             }
544             \char_set_active_eq:NN ^^M \__scontents_ret:w
545         }
546     }
547     ^^M
548 }
549 \cs_new_protected:Npn \__scontents_file_write_stop:N #1
550 {
551     \bool_if:NT \l__scontents_writing_bool
552     { \iow_close:N \l__scontents_file_iow }
553     \use:x
554     {
555         \group_end:
556         \bool_if:NT \l__scontents_storing_bool
557         {
558             \tl_set:Nn \exp_not:N #1
559             { \exp_args:NV \__scontents_remove_leading_nl:n \l__scontents_file_tl }
560         }
561     }

```

```

562 }
563 \cs_new:Npn \__scontents_remove_leading_nl:n #1
564 {
565   \tl_if_head_is_N_type:nTF {#1}
566   {
567     \exp_args:Nf
568       \__scontents_remove_leading_nl:nn
569       { \tl_head:n {#1} } {#1}
570   }
571   { \exp_not:n {#1} }
572 }
573 \cs_new:Npn \__scontents_remove_leading_nl:nn #1 #2
574 {
575   \token_if_eq_meaning:NNTF ^^J #1
576   { \exp_not:o { \__scontents_remove_leading_nl:w #2 } }
577   { \exp_not:n {#2} }
578 }
579 \cs_new:Npn \__scontents_remove_leading_nl:w ^^J { }

```

(End definition for `__scontents_file_tl_write_start:n` and others.)

`__scontents_verb_processor_output:n` `__scontents_verb_processor_output:x` does the output of each line read, to a token list and to a file, depending on the booleans `__scontents_writing_bool` and `__scontents_storing_bool`.

`__scontents_analyse_nesting:n` `__scontents_analyse_nesting:w` looks for nested `\begin{scontents}` and adds to a `__scontents_env_nesting_int` counter. The `__scontents_if_nested:` conditional tests if we're in a nested environment, and `__scontents_nesting_decr:` reduces the nesting level, if an `\end{scontents}` is found.

`__scontents_use_none_delimit_by_q_stop:w` `__scontents_if_nested:TF` Multiple `\end{scontents}` in the same line are not supported...

```

580 \cs_new_protected:Npn \__scontents_verb_processor_output:n #1
581 {
582   \bool_if:NT \__scontents_writing_bool
583   { \iow_now:Nn \__scontents_file_iow {#1} }
584   \bool_if:NT \__scontents_storing_bool
585   { \tl_put_right:Nn \__scontents_file_tl { ^^J #1 } }
586 }
587 \cs_generate_variant:Nn \__scontents_verb_processor_output:n { x }
588 \cs_new_protected:Npx \__scontents_analyse_nesting:n #1
589 {
590   \int_zero:N \__scontents_tmpa_int
591   \exp_not:N \__scontents_analyse_nesting:w #1
592   \c_backslash_str begin
593     \c_left_brace_str \exp_not:N \__scontents_mark \c_right_brace_str
594   \exp_not:N \__scontents_stop
595   \int_compare:nNnT { \__scontents_tmpa_int } > { 1 }
596   { \msg_warning:nn { scontents } { multiple-begin } }
597 }
598 \use:x
599 {
600   \cs_new_protected:Npn \exp_not:N \__scontents_analyse_nesting:w #1
601   \c_backslash_str begin \c_left_brace_str #2 \c_right_brace_str
602 } {
603   \if_meaning:w \__scontents_mark #2
604     \exp_after:wN \use_i:nn
605   \else:
606     \exp_after:wN \use_ii:nn
607   \fi:
608   { \__scontents_use_none_delimit_by_q_stop:w }
609   {
610     \str_if_eq:eeT {#2} {scontents}
611     {
612       \int_incr:N \__scontents_env_nesting_int
613       \int_incr:N \__scontents_tmpa_int
614       \__scontents_analyse_nesting:w
615     }
616     \__scontents_analyse_nesting:w
617   }
618 }
619 \cs_new_protected:Npn \__scontents_nesting_decr:
620 { \int_decr:N \__scontents_env_nesting_int }

```

```

621 \prg_new_protected_conditional:Npnn \__scontents_if_nested: { TF }
622 {
623   \int_compare:nNnTF { \l__scontents_env_nesting_int } > { \c_zero_int }
624   { \prg_return_true: }
625   { \prg_return_false: }
626 }
627 \cs_new:Npn \__scontents_use_none_delimit_by_q_stop:w #1 \q__scontents_stop { }
628 \group_end:
629 \cs_generate_variant:Nn \__scontents_file_tl_write_start:n { V }

```

(End definition for __scontents_verb_processor_output:n and others.)

10.8.3 Recording of the content in the sequence

__scontents_atend_environment: Finishes the environment by optionally calling __scontents_store_to_seq: and then clearing the temporary token list.

```

630 \cs_new_protected:Npn \__scontents_atend_environment:
631 {
632   \bool_if:NT \l__scontents_storing_bool
633   {
634     \bool_if:NF \l__scontents_forced_eol_bool
635     { \tl_put_right:Nx \l__scontents_macro_tmp_tl { \c__scontents_hidden_space_str } }
636     \__scontents_store_to_seq:NN \l__scontents_macro_tmp_tl \l__scontents_name_seq_env_tl
637     \bool_if:NT \l__scontents_print_env_bool
638     { \__scontents_lastfrom_seq:n \l__scontents_name_seq_env_tl }
639   }
640 }
641 </core>

```

(End definition for __scontents_atend_environment:.)

\verbatim In Plain we emulate ~~La~~^{TeX}'s verbatim environment.

```

\endverbatim
642 <*plain>
643 \bool_new:N \l__scontents_temp_bool
644 \cs_new_protected:Npn \verbatim
645 { \__scontents_verbatim_aux: \frenchspacing \__scontents_vobeyspaces: \__scontents_xverb: }
646 \cs_new_protected:Npn \__scontents_verbatim_aux:
647 {
648   \skip_vertical:N \parskip
649   \int_set:Nn \parindent { 0pt }
650   \skip_set:Nn \parfillskip { 0pt plus 1fil }
651   \int_set:Nn \parskip { 0pt plus 0pt minus 0pt }
652   \tex_par:D
653   \bool_set_false:N \l__scontents_temp_bool
654   \cs_set:Npn \par
655   {
656     \bool_if:NTF \l__scontents_temp_bool
657     {
658       \mode_leave_vertical:
659       \null
660       \tex_par:D
661       \penalty \interlinepenalty
662     }
663     {
664       \bool_set_true:N \l__scontents_temp_bool
665       \mode_if_horizontal:T
666       { \tex_par:D \penalty \interlinepenalty }
667     }
668   }
669   \cs_set_eq:NN \do \char_set_catcode_other:N
670   \dospecials \obeylines
671   \tl_use:N \l__scontents_verb_font_tl
672   \cs_set_eq:NN \do \__scontents_do_noligs:N
673   \__scontents_nolig_list:
674   \tex_everypar:D \exp_after:wN
675   { \tex_the:D \tex_everypar:D \tex_unpenalty:D }
676 }
677 \cs_new_protected:Npn \__scontents_nolig_list:
678 { \do\'\do\<\do\>\do\,\do\'\do\~ }

```

```

679 \cs_new_protected:Npn \__scontents_vobeyspaces:
680 { \__scontents_set_active_eq:NN \ \__scontents_xobeysp: }
681 \cs_new_protected:Npn \__scontents_xobeysp:
682 { \mode_leave_vertical: \nobreak \ }
683 </plain>

```

(End definition for `\verbatim` and `\endverbatim`.)

`\dospecials` xparse also requires L^AT_EX's `\dospecials`. In case it doesn't exist (at the time `scontents` is loaded) we define `\dospecials` to use the `\l_char_special_seq`.

```

684 <!!latex>
685 \cs_if_exist:NF \dospecials
686 {
687   \cs_new:Npn \dospecials
688     { \seq_map_function:NN \l_char_special_seq \do }
689 }
690 </!!latex>

```

(End definition for `\dospecials`.)

10.9 The command `\Scontents`

User command to *<stored content>*, adapted from <https://tex.stackexchange.com/a/500281/7832>.

`\Scontents` The `\Scontents` macro starts by parsing an optional argument and then delegates to `__scontents_verb_arg:w` or `__scontents_norm_arg:n` depending whether a star (*) argument is present. `__scontents_norm_arg:n` grabs a normal argument, adds it to the `seq` variable, and optionally prints it. `__scontents_verb_arg:w` grabs a verbatim argument using xparse's +v argument parser.

```

691 <*loader>
692 \NewDocumentCommand { \Scontents }{!s !0{ } }
693 { \__scontents_Scontents_internal:nn {#1} {#2} }
694 </loader>
695 <*core>
696 \cs_new_protected:Npn \__scontents_Scontents_internal:nn #1 #2
697 {
698   \group_begin:
699   \tl_if_novalue:NF {#2}
700     { \keys_set:nn { scontents / Scontents } {#2} }
701   \char_set_catcode_active:n { 9 }
702   \bool_if:NTF #1
703     { \__scontents_verb_arg:w }
704     { \__scontents_norm_arg:n }
705 }
706 \cs_new_protected:Npn \__scontents_norm_arg:n #1
707 {
708   \tl_set:Nx \l__scontents_temp_tl { \exp_not:n {#1} }
709   \tl_put_right:Nx \l__scontents_temp_tl { \c__scontents_hidden_space_str }
710   \__scontents_store_to_seq:NN \l__scontents_temp_tl \l__scontents_name_seq_cmd_tl
711   \bool_if:NT \l__scontents_print_cmd_bool
712     { \__scontents_lastfrom_seq:n \l__scontents_name_seq_cmd_tl }
713   \group_end:
714 }
715 </core>
716 <*loader>
717 \NewDocumentCommand { \__scontents_verb_arg:w }{+v }
718 { \__scontents_verb_arg_internal:n {#1} }
719 </loader>
720 <*core>
721 \cs_new_protected:Npn \__scontents_verb_arg_internal:n #1
722 {
723   \tl_set:Nx \l__scontents_temp_tl { \exp_not:n {#1} }
724   \tl_replace_all:Nxx \l__scontents_temp_tl { \iow_char:N ^^M } { \iow_char:N ^^J }
725   \bool_if:NF \l__scontents_forced_eol_bool
726     { \tl_put_right:Nx \l__scontents_temp_tl { \c__scontents_hidden_space_str } }
727   \__scontents_store_to_seq:NN \l__scontents_temp_tl \l__scontents_name_seq_cmd_tl
728   \bool_if:NT \l__scontents_print_cmd_bool
729     { \__scontents_lastfrom_seq:n \l__scontents_name_seq_cmd_tl }

```

```

730 \group_end:
731 }

```

(End definition for `\Scontents` and others. This function is documented on page 4.)

10.10 The command `\getstored`

`\getstored` User command `\getstored` to extract *stored content* in seq (robust).

```

732 </core>
733 <*loader>
734 \NewDocumentCommand { \getstored } { 0{1} m }
735 { \__scontents_getstored_internal:nn {#1} {#2} }
736 </loader>
737 <*core>
738 \cs_new_protected:Npn \__scontents_getstored_internal:nn #1 #2
739 {
740   \group_begin:
741   \int_set:Nn \tex_newlinechar:D { '\^^J }
742   \__scontents_rescan_tokens:x
743   {
744     \endgroup % This assumes \catcode'\=\0... Things might go off otherwise.
745     \__scontents_getfrom_seq:nn {#1} {#2}
746   }
747 }

```

(End definition for `\getstored`. This function is documented on page 4.)

10.11 The command `\foreachsc`

`\foreachsc` User command `\foreachsc` to loop over *stored content* in seq.

```

748 </core>
749 <*loader>
750 \NewDocumentCommand { \foreachsc } { o m }
751 { \__scontents_foreachsc_internal:nn {#1} {#2} }
752 </loader>
753 <*core>
754 \cs_new_protected:Npn \__scontents_foreachsc_internal:nn #1 #2
755 {
756   \group_begin:
757   \tl_if_novalue:nF {#1} { \keys_set:nn { scontents / foreachsc } {#1} }
758   \tl_set:Nn \l__scontents_foreach_name_seq_tl {#2}
759   \seq_clear:N \l__scontents_foreach_print_seq
760   \bool_if:NF \l__scontents_foreach_stop_bool
761   {
762     \int_set:Nn \l__scontents_foreach_stop_int
763     { \seq_count:c { g__scontents_name_#2_seq } }
764   }
765   \int_step_function:nnnN
766   { \l__scontents_foreach_start_int }
767   { \l__scontents_foreach_step_int }
768   { \l__scontents_foreach_stop_int }
769   \__scontents_foreach_add_body:n
770   \tl_gset:Nx \g__scontents_temp_tl
771   { \seq_use:Nn \l__scontents_foreach_print_seq { \tl_use:N \l__scontents_foreach_sep_tl } }
772   \group_end:
773   \exp_after:wN \tl_gclear:N
774   \exp_after:wN \g__scontents_temp_tl
775   \g__scontents_temp_tl
776 }
777 \cs_new_protected:Npn \__scontents_foreach_add_body:n #1
778 {
779   \seq_put_right:Nx \l__scontents_foreach_print_seq
780   {
781     \bool_if:NT \l__scontents_foreach_before_bool
782     { \exp_not:V \l__scontents_foreach_before_tl }
783     \bool_if:NTF \l__scontents_foreach_wrapper_bool
784     { \__scontents_foreach_wrapper:n }
785     { \use:n }

```

```

786         { \getstored [#1] { \tl_use:N \l__scontents_foreach_name_seq_tl } }
787         \bool_if:NT \l__scontents_foreach_after_bool
788         { \exp_not:V \l__scontents_foreach_after_tl }
789     }
790 }

```

(End definition for `\foreachsc`. This function is documented on page 5.)

10.12 The command `\typestored`

This implementation is an adaptation taken from answer by Phelype Oleinik in (<https://tex.stackexchange.com/a/497651/7832>).

```

\typestored The \typestored commands fetches a buffer from memory, prints it to the log file, and then calls \__-
\__scontents_verb_print:N scontents_verb_print:N.
\__scontents_xverb:w
\verbatimsc
791 </core>
792 <*loader>
793 \NewDocumentCommand { \typestored } { o m }
794 { \__scontents_typestored_internal:nn {#1} {#2} }
795 </loader>
796 <*core>
797 \cs_new_protected:Npn \__scontents_typestored_internal:nn #1 #2
798 {
799     \group_begin:
800     \int_set:Nn \l__scontents_seq_item_int { 1 }
801     \tl_if_novalue:nF {#1} { \keys_set:nn { scontents / typemeaning } {#1} }
802     \tl_set:Nx \l__scontents_temp_tl
803     { \exp_args:NV \__scontents_getfrom_seq:nn \l__scontents_seq_item_int {#2} }
804     \tl_remove_once:NV \l__scontents_temp_tl \c__scontents_hidden_space_str
805     \tl_log:N \l__scontents_temp_tl
806     \tl_if_empty:NF \l__scontents_temp_tl
807     { \__scontents_verb_print:N \l__scontents_temp_tl }
808     \group_end:
809 }

```

The `__scontents_verb_print:N` macro is defined with active carriage return (ASCII 13) characters to mimick an actual verbatim environment “on the loose”. The contents of the environment are placed in a `verbatimsc` environment and rescanned using `__scontents_rescan_tokens:x`.

```

810 \group_begin:
811 \char_set_catcode_active:N ^^M
812 \cs_new_protected:Npn \__scontents_verb_print:N #1
813 {
814     \tl_if_blank:VT #1
815     { \msg_error:nnn { scontents } { empty-variable } {#1} }
816     \cs_set_eq:NN \__scontents_verb_print_EOL: ^^M
817     \cs_set_eq:NN ^^M \scan_stop:
818     \cs_set_eq:cN { do@noligs } \__scontents_do_noligs:N
819     \int_set:Nn \tex_newlinechar:D { '^^J }
820     \__scontents_rescan_tokens:x
821     {
822         \__scontents_format_case:nnn
823         { \exp_not:N \begin{verbatimsc} } % LaTeX
824         { \verbatimsc } % Plain/Generic
825         { \startverbatimsc } % ConTeXt
826         ^^M
827         \exp_not:V #1 ^^M
828         \g__scontents_end_verbatimsc_tl
829     }
830     \cs_set_eq:NN ^^M \__scontents_verb_print_EOL:
831 }
832 \group_end:

```

Finally, the `verbatimsc` environment is defined.

```

833 \cs_new_protected:Npn \__scontents_xverb:
834 {
835     \char_set_catcode_active:n { 9 }
836     \char_set_active_eq:nN { 9 } \__scontents_tabs_to_spaces:
837     \__scontents_xverb:w

```

```

838 }
839 </core>
840 <*loader>
841 \use:x
842 {
843   \cs_new_protected:Npn \exp_not:N \__scontents_xverb:w
844     ##1 \g_scontents_end_verbatimsc_tl
845   <latex> { ##1 \exp_not:N \end{verbatimsc} }
846   <plain> { ##1 \exp_not:N \endverbatimsc }
847   <context> { ##1 \exp_not:N \stopverbatimsc }
848 }
849 <!*context>
850 \NewDocumentEnvironment { verbatimsc } { }
851 {
852   <plain> \group_begin:
853   <latex> \cs_set_eq:cN { @xverbatim } \__scontents_xverb:
854     \verbatim
855 }
856 {
857   <plain> \group_end:
858 }
859 </!context>
860 <context>\definetyping[verbatimsc]
861 </loader>
862 <*core>

```

(End definition for \typestored and others. These functions are documented on page 5.)

10.13 Some auxiliaries

`__scontents_tabs_to_spaces:` In a verbatim context the TAB character is made active and set equal to `__scontents_tabs_to_spaces:`, to produce as many spaces as the `width-tab` key was set to.

```

863 \cs_new:Npn \__scontents_tabs_to_spaces:
864   { \prg_replicate:nn { \l__scontents_tab_width_int } { ~ } }

```

(End definition for `__scontents_tabs_to_spaces:`.)

`__scontents_do_noligs:N` `__scontents_do_noligs:N` is an alternative definition for $\text{\LaTeX}_{2\epsilon}$'s `\do@noligs` which makes sure to not consume following space tokens. The $\text{\LaTeX}_{2\epsilon}$ version ends with `\char‘#1`, which leaves \TeX still looking for an *optional space*. This version uses `\char_generate:nn` to ensure that doesn't happen.

```

865 \cs_new:Npn \__scontents_do_noligs:N #1
866   {
867     \char_set_catcode_active:N #1
868     \char_set_active_eq:Nc #1 { __scontents_active_char_ \token_to_str:N #1 : }
869     \cs_set:cpx { __scontents_active_char_ \token_to_str:N #1 : }
870     {
871       \mode_leave_vertical:
872       \tex_kern:D \c_zero_dim
873       \char_generate:nn { ‘#1 } { 12 }
874     }
875   }

```

(End definition for `__scontents_do_noligs:N`.)

`__scontents_set_active_eq:NN` `__scontents_make_control_chars_active:` Shortcut definitions for common catcode changes. The `^^L` needs a special treatment in non- \LaTeX mode because in Plain \TeX it is an `\outer` token.

```

876 \cs_new_protected:Npn \__scontents_set_active_eq:NN #1
877   {
878     \char_set_catcode_active:N #1
879     \char_set_active_eq:NN #1
880   }
881 </core>
882 <*loader>
883 \group_begin:
884 <plain> \char_set_catcode_active:n { ‘\* }
885   \cs_new_protected:Npn \__scontents_plain_disable_outer_par:
886 <*plain>

```

```

887 {
888   \group_begin:
889     \char_set_lccode:nn { '\* } { '^L }
890     \tex_lowercase:D { \group_end:
891       \tex_let:D * \scan_stop:
892     }
893   }
894 \</plain>
895 \<latex | context> { }
896 \group_end:
897 \</loader>
898 \<*core>
899 \group_begin:
900   \char_set_catcode_active:N \*
901   \cs_new_protected:Npn \__scontents_make_control_chars_active:
902   {
903     \__scontents_plain_disable_outer_par:
904     \__scontents_set_active_eq:NN \^^I \__scontents_tab:
905     \__scontents_set_active_eq:NN \^^L \__scontents_par:
906     \__scontents_set_active_eq:NN \^^M \__scontents_ret:w
907   }
908 \group_end:

```

(End definition for __scontents_set_active_eq:NN and __scontents_make_control_chars_active:.)

10.14 The command \setupsc

User command \setupsc to setup module.

\setupsc A user-level wrapper for \keys_set:nn{ scontents }.

```

909 \</core>
910 \<*loader>
911 \NewDocumentCommand { \setupsc } { +m }
912 { \keys_set:nn { scontents } {#1} }
913 \</loader>
914 \<*core>

```

(End definition for \setupsc. This function is documented on page 2.)

10.15 The command \meaningsc

\meaningsc User command \meaningsc to see content stored in seq.

```

915 \</core>
916 \<*loader>
917 \NewDocumentCommand { \meaningsc } { o m }
918 { \__scontents_meaningsc_internal:nn {#1} {#2} }
919 \</loader>
920 \<*core>
921 \cs_new_protected:Npn \__scontents_meaningsc_internal:nn #1 #2
922 {
923   \group_begin:
924     \int_set:Nn \l__scontents_seq_item_int { 1 }
925     \tl_if_novalue:nF {#1} { \keys_set:nn { scontents / typemeaning } {#1} }
926     \__scontents_meaningsc:n {#2}
927   \group_end:
928 }
929 \group_begin:
930   \char_set_catcode_active:N \^^I
931   \cs_new_protected:Npn \__scontents_meaningsc:n #1
932   {
933     \tl_set:Nx \l__scontents_temp_tl
934     { \exp_args:NV \__scontents_getfrom_seq:nn \l__scontents_seq_item_int {#1} }
935     \tl_replace_all:Nxn \l__scontents_temp_tl { \iow_char:N \^^J } { ~ }
936     \tl_remove_once:NV \l__scontents_temp_tl \c__scontents_hidden_space_str
937     \tl_log:N \l__scontents_temp_tl
938     \tl_use:N \l__scontents_verb_font_tl
939     \tl_replace_all:Nxn \l__scontents_temp_tl { \^^I } { \__scontents_tabs_to_spaces: }
940     \cs_replacement_spec:N \l__scontents_temp_tl
941   }

```

942 `\group_end:`

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

10.16 The command `\countsc`

`\countsc` User command `\countsc` to count number of contents stored in seq.

```
943 </core>
944 <*loader>
945 \NewExpandableDocumentCommand { \countsc } { m }
946 { \seq_count:c { g__scontents_name_#1_seq } }
947 </loader>
948 <*core>
```

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

10.17 The command `\cleanseqsc`

`\cleanseqsc` A user command `\cleanseqsc` to clear (remove) a defined seq.

```
949 </core>
950 <*loader>
951 \NewDocumentCommand { \cleanseqsc } { m }
952 { \seq_clear_new:c { g__scontents_name_#1_seq } }
953 </loader>
954 <*core>
```

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

10.18 Warning and error messages

Warning and error messages used throughout the package.

```
955 \msg_new:nnn { scontents } { junk-after-begin }
956 {
957     Junk~characters~#1~\msg_line_context: :
958     \\ \\
959     #2
960 }
961 \msg_new:nnn { scontents } { empty-stored-content }
962 { Empty~value~for~key~'getstored'~\msg_line_context:. }
963 \msg_new:nnn { scontents } { empty-variable }
964 { Variable~'#1'~empty~\msg_line_context:. }
965 \msg_new:nnn { scontents } { rewriting-file }
966 { Overwriting ~ file ~ '#1' }
967 \msg_new:nnn { scontents } { writing-file }
968 { Writing ~ file ~ '#1' }
969 \msg_new:nnn { scontents } { rescanning-text }
970 { Rescanning~text~'#1'~after~\c_backslash_str end{scontents}~\msg_line_context:. }
971 \msg_new:nnn { scontents } { multiple-begin }
972 { Multiple~\c_backslash_str begin{scontents}~\msg_line_context:. }
973 \msg_new:nnn { scontents } { undefined-storage }
974 { Storage~named~'#1'~is~not~defined. }
975 \msg_new:nnn { scontents } { index-out-of-range }
976 {
977     \int_compare:nNnTF {#1} = { 0 }
978     { Index~of~sequence~cannot~be~zero. }
979     {
980         Index~'#1'~out~of~range~for~'#2',~
981         \int_compare:nNnTF {#1} > { 0 }
982         { Max = } { Min = - } #3.
983     }
984 }
985 \msg_new:nnnn { scontents } { env-key-unknown }
986 { The~key~'#1'~is~unknown~by~environment~'scontents'~and~is~being~ignored. }
987 {
988     The~environment~'scontents'~does~not~have~a~key~called~'#1'.\\
989     Check~that~you~have~spelled~the~key~name~correctly.
990 }
991 \msg_new:nnnn { scontents } { env-key-value-unknown }
```

```

992 { The~key~'#1=#2'~is~unknown~by~environment~'scontents'~and~is~being~ignored. }
993 {
994   The~environment~'scontents'~does~not~have~a~key~called~'#1'.\\
995   Check~that~you~have~spelled~the~key~name~correctly.
996 }
997 \msg_new:nnnn { scontents } { cmd-key-unknown }
998 { The~key~'#1'~is~unknown~by~'\c_backslash_str Scontents'~and~is~being~ignored.}
999 {
1000   The~command~'\c_backslash_str Scontents'~does~not~have~a~key~called~'#1'.\\
1001   Check~that~you~have~spelled~the~key~name~correctly.
1002 }
1003 \msg_new:nnnn { scontents } { cmd-key-value-unknown }
1004 { The~key~'#1=#2'~is~unknown~by~'\c_backslash_str Scontents'~and~is~being~ignored. }
1005 {
1006   The~command~'\c_backslash_str Scontents'~does~not~have~a~key~called~'#1'.\\
1007   Check~that~you~have~spelled~the~key~name~correctly.
1008 }
1009 \msg_new:nnnn { scontents } { for-key-unknown }
1010 { The~key~'#1'~is~unknown~by~'\c_backslash_str foreachsc'~and~is~being~ignored.}
1011 {
1012   The~command~'\c_backslash_str foreachsc'~does~not~have~a~key~called~'#1'.\\
1013   Check~that~you~have~spelled~the~key~name~correctly.
1014 }
1015 \msg_new:nnnn { scontents } { for-key-value-unknown }
1016 { The~key~'#1=#2'~is~unknown~by~'\c_backslash_str foreachsc'~and~is~being~ignored. }
1017 {
1018   The~command~'\c_backslash_str foreachsc'~does~not~have~a~key~called~'#1'.\\
1019   Check~that~you~have~spelled~the~key~name~correctly.
1020 }
1021 \msg_new:nnnn { scontents } { type-key-unknown }
1022 { The~key~'#1'~is~unknown~and~is~being~ignored. }
1023 {
1024   This~command~does~not~have~a~key~called~'#1'.\\
1025   This~command~only~accepts~the~key~'width-tab'.
1026 }
1027 \msg_new:nnnn { scontents } { type-key-value-unknown }
1028 { The~key~'#1'~to~which~you~passed~'#2'~is~unknown~and~is~being~ignored. }
1029 {
1030   This~command~does~not~have~a~key~called~'#1'.\\
1031   This~command~only~accepts~the~key~'width-tab'.
1032 }
1033 \msg_new:nnn { scontents } { empty-environment }
1034 { scontents~environment~empty~\msg_line_context:. }
1035 \msg_new:nnnn { scontents } { verbatim-newline }
1036 { Verbatim~argument~of~#1~ended~by~end~of~line. }
1037 {
1038   The~verbatim~argument~of~the~#1~cannot~contain~more~than~one~line,~
1039   but~the~end~
1040   of~the~current~line~has~been~reached.~You~may~have~forgotten~the~
1041   closing~delimiter.
1042   \\ \\
1043   LaTeX~will~ignore~'#2'.
1044 }
1045 \msg_new:nnnn { scontents } { verbatim-tokenized }
1046 { The~verbatim~#1~cannot~be~used~inside~an~argument. }
1047 {
1048   The~#1~takes~a~verbatim~argument.~
1049   It~may~not~appear~within~the~argument~of~another~function.~
1050   It~received~an~illegal~token \tl_if_empty:nF {#3} { ~'#3' } .
1051   \\ \\
1052   LaTeX~will~ignore~'#2'.
1053 }

```

10.19 Finish package

Finish package implementation.

```

1054 </core>
1055 <plain | context>\ExplSyntaxOff

```

11 Index of Implementation

Symbols

\'	678
*	884, 889, 900
\,	678
\-	678
\<	678
\>	678
\\	15, 16, 17, 45, 744, 958, 988, 994, 1000, 1006, 1012, 1018, 1024, 1030, 1042, 1051
\'	678

B

\beginpgroup	70, 73
bool commands:	
\bool_if:NFTF	487, 551, 556, 582, 584, 632, 634, 637, 656, 702, 711, 725, 728, 760, 781, 783, 787
\bool_lazy_or:nnTF	386, 535
\bool_new:N	155, 157, 159, 161, 163, 165, 643
\bool_set_false:N	156, 160, 162, 164, 166, 188, 653
\bool_set_true:N	158, 184, 189, 213, 218, 226, 234, 664

C

\catcode	744
\char	31
char commands:	
\char_generate:nn	31, 873
\char_set_active_eq:NN	544, 868, 879
\char_set_active_eq:nN	836
\char_set_catcode:nn	361
\char_set_catcode_active:N	422, 447, 448, 449, 811, 867, 878, 900, 930
\char_set_catcode_active:n	701, 835, 884
\char_set_catcode_letter:N	356
\char_set_catcode_letter:n	496
\char_set_catcode_other:N	45, 47, 48, 495, 669
\char_set_lccode:nn	889
\l_char_special_seq	28, 495, 688
\char_value_catcode:n	355
\cleanseqsc	1, 6, 6, 33, 949
clist commands:	
\clist_map_function:NN	324
\countsc	1, 6, 6, 33, 943
cs commands:	
\cs_generate_variant:Nn	174, 177, 178, 179, 304, 373, 587, 629
\cs_if_exist:NFTF	34, 685
\cs_new:Npn	64, 176, 313, 338, 339, 341, 374, 384, 509, 563, 573, 579, 627, 687, 863, 865
\cs_new:Npx	175
\cs_new_protected:Npn	173, 248, 250, 256, 258, 264, 270, 272, 274, 291, 295, 297, 299, 305, 307, 311, 321, 326, 332, 345, 365, 392, 400, 408, 413, 420, 425, 436, 450, 452, 462, 464, 475, 484, 549, 580, 600, 619, 630, 644, 646, 677, 679, 681, 696, 706, 721, 738, 754, 777, 797, 812, 833, 843, 876, 885, 901, 921, 931
\cs_new_protected:Npx	588
\cs_replacement_spec:N	940
\cs_set:Npn	654
\cs_set:Npx	869
\cs_set_eq:NN	294, 301, 669, 672, 816, 817, 818, 830, 853
\cs_set_protected:Npn	235

\cs_set_protected:Npx	497, 529
\cs_to_str:N	294, 301, 302
\cs_undefine:N	302
\csname	72, 82

D

\DeclareOption	351
\def	2, 3, 5, 6, 71, 74, 75, 83, 96
\definetying	860
dim commands:	
\c_zero_dim	872
\do	669, 672, 678, 688
\dospecials	28, 670, 684

E

\else	93, 95, 344
else commands:	
\else:	317, 605
\end	51, 532, 845
\endcsname	72, 82
\endgroup	71, 74, 77, 90, 104, 744
\endinput	91, 105
\endscontents	405, 533
\endverbatim	642
\endverbatimsc	52, 846
\errhelp	78
\errmessage	79
exp commands:	
\exp_after:wN	316, 318, 335, 336, 357, 604, 606, 674, 773, 774
\exp_args:Nc	331
\exp_args:Nf	378, 567
\exp_args:NNNV	478
\exp_args:NV	249, 257, 271, 273, 559, 803, 934
\exp_not:N	499, 502, 509, 512, 532, 558, 591, 593, 594, 600, 823, 843, 845, 846, 847
\exp_not:n	403, 524, 571, 576, 577, 708, 723, 782, 788, 827
\expandafter	72, 82
\ExplSyntaxOff	21, 37, 358, 1055
\ExplSyntaxOn	21, 22, 26, 349, 360

F

\fi	81, 107, 108, 344
fi commands:	
\fi:	319, 607
file commands:	
\file_if_exist:nTF	489
\file_input:n	63, 359
\file_input_stop:	38
\foreachsc	1, 5, 5, 18, 20, 29, 748
\frenchspacing	645

G

\getstored	1, 4, 4, 29, 732, 786
group commands:	
\group_begin:	410, 446, 472, 486, 698, 740, 756, 799, 810, 852, 883, 888, 899, 923, 929
\group_end:	416, 479, 555, 628, 713, 730, 772, 808, 832, 857, 890, 896, 908, 927, 942
\group_insert_after:N	395, 396, 397, 398

I

if commands:

`\if_meaning:w` 315, 603
`\ifx` 72, 82, 94, 344
`\input` 25

int commands:

`\int_abs:n` 388
`\int_compare:nNnTF` 334, 595, 623, 977, 981
`\int_compare_p:nNn` 387, 388
`\int_decr:N` 620
`\int_incr:N` 612, 613
`\int_new:N` 151, 152, 153, 154
`\int_set:Nn` 227, 279, 355, 649, 651, 741, 762, 800, 819, 924
`\int_step_function:nnN` 496
`\int_step_function:nnnN` 765
`\int_to_roman:n` 20, 276
`\int_zero:N` 590
`\c_zero_int` 623
`\interlinepenalty` 661, 666

iow commands:

`\iow_char:N` 724, 935
`\iow_close:N` 552
`\iow_log:n` 29, 348
`\iow_new:N` 172
`\iow_now:Nn` 583
`\iow_open:Nn` 492
`\iow_term:n` 13

K

keys commands:

`\keys_define:nn` 110, 138, 181, 201, 210, 243
`\l_keys_key_tl` 19, 249, 257, 271, 273
`\keys_set:nn` 32, 442, 700, 757, 801, 912, 925

L

left commands:

`\c_left_brace_str` 59, 593, 601

M

`\meaningsc` 1, 6, 6, 16, 19, 20, 32, 915

mode commands:

`\mode_if_horizontal:TF` 665
`\mode_leave_vertical:` 658, 682, 871

msg commands:

`\msg_error:nn` 368
`\msg_error:nnn` 253, 261, 267, 284, 327, 815
`\msg_error:nnnn` ... 179, 254, 262, 268, 280, 285, 468
`\msg_expandable_error:nnn` 382
`\msg_expandable_error:nnnn` 389
`\msg_gset:nnn` 32
`\msg_line_context:` 33, 957, 962, 964, 970, 972, 1034
`\msg_new:nnn` . 328, 955, 961, 963, 965, 967, 969, 971, 973, 975, 1033
`\msg_new:nnnn` . 985, 991, 997, 1003, 1009, 1015, 1021, 1027, 1035, 1045
`\msg_warning:nn` 36, 596
`\msg_warning:nnn` 482, 490, 491, 540

N

`\NewDocumentCommand` 432, 692, 717, 734, 750, 793, 911, 917, 951

`\NewDocumentEnvironment` 407, 850

`\NewExpandableDocumentCommand` 945

`\next` 71, 74, 83, 96, 109

`\nobreak` 682
`\null` 659

O

`\obeylines` 670
`\outer` 31

P

`\PackageError` 75, 85, 98
`\par` 654
`\parfillskip` 650
`\parindent` 649
`\parskip` 648, 651

peek commands:

`\peek_charcode_ignore_spaces:NTF` 306
`\peek_charcode_remove_ignore_spaces:NTF` . 308

`\penalty` 661, 666

prg commands:

`\prg_generate_conditional_variant:Nnn` .. 180
`\prg_new_protected_conditional:Npnn` 621
`\prg_replicate:nn` 864
`\prg_return_false:` 625
`\prg_return_true:` 624

`\ProcessKeysOptions` 141

`\ProcessOptions` 353

`\ProvidesExplPackage` 9, 343

Q

quark commands:

`\q_mark` 315, 323, 335, 336, 339, 340, 341
`\quark_new:N` 170, 171

quark internal commands:

`\q__scontents_mark` 170, 593, 603
`\q__scontents_stop` 170, 502, 512, 594, 627

R

`\relax` 72, 82, 344

`\RequirePackage` 8, 309

right commands:

`\c_right_brace_str` 61, 593, 601

S

scan commands:

`\scan_stop:` 817, 891

`\Scontents` 1, 4, 4, 18, 19, 28, 691

`\scontents` 405

`scontents` 3, 405

scontents internal commands:

`__scontents_analyse_nesting:n` ... 26, 516, 580
`__scontents_analyse_nesting:w` 580
`__scontents_append_contents:nn` .. 22, 365, 403
`__scontents_atend_environment:` 428, 630
`__scontents_check_line_and_process:nn` 24, 446
`__scontents_compat_redefine:Npn` . 291, 304, 309, 330, 343, 351, 353

`__scontents_compat_restore:` 297, 362

`__scontents_compat_restore:N` 298, 299

`\l__scontents_compat_seq` 290, 293, 298

`__scontents_do_noligs:N` 31, 672, 818, 865

`\c__scontents_end_env_tl` 41, 500, 501, 510, 511, 524

`\g__scontents_end_verbatimsc_tl` .. 41, 828, 844

`\l__scontents_env_nesting_int` .. 16, 26, 151, 612, 620, 623

`\l__scontents_file_iow` 172, 492, 552, 583

`\l__scontents_file_tl` [16](#), [143](#), [494](#), [559](#), [585](#)
`__scontents_file_tl_write_start:n` [24](#), [473](#), [484](#),
[629](#)
`__scontents_file_write_stop:N` . . . [24](#), [477](#), [484](#)
`\l__scontents_fname_out_tl` . [16](#), [143](#), [185](#), [190](#), [473](#)
`\l__scontents_forced_eol_bool` . . . [126](#), [634](#), [725](#)
`__scontents_foreach_add_body:n` [769](#), [777](#)
`\l__scontents_foreach_after_bool` . [161](#), [162](#), [218](#),
[787](#)
`\l__scontents_foreach_after_`
 `bool` `\l__scontents_`
 `foreach_stop_bool` [159](#)
`\l__scontents_foreach_after_tl` [16](#), [143](#), [219](#), [788](#)
`\l__scontents_foreach_before_bool` [159](#), [213](#), [781](#)
`\l__scontents_foreach_before_tl` [16](#), [143](#), [214](#), [782](#)
`\l__scontents_foreach_name_seq_tl` . [16](#), [143](#), [758](#),
[786](#)
`\l__scontents_foreach_print_seq` [17](#), [167](#), [759](#), [771](#),
[779](#)
`\l__scontents_foreach_sep_tl` [238](#), [771](#)
`\l__scontents_foreach_start_int` [222](#), [766](#)
`\l__scontents_foreach_step_int` [230](#), [767](#)
`\l__scontents_foreach_stop_bool` [226](#), [760](#)
`\l__scontents_foreach_stop_int` [16](#), [151](#), [227](#), [762](#),
[768](#)
`__scontents_foreach_wrapper:n` [235](#), [784](#)
`\l__scontents_foreach_wrapper_bool` [159](#), [234](#), [783](#)
`__scontents_foreachsc_internal:nn` . . [751](#), [754](#)
`__scontents_format_case:nnn` [64](#), [531](#), [822](#)
`__scontents_getfrom_seq:nn` [22](#), [374](#), [745](#), [803](#), [934](#)
`__scontents_getfrom_seq:nnn` [374](#)
`__scontents_getstored_internal:nn` . . [735](#), [738](#)
`__scontents_grab_optional:n` [430](#)
`__scontents_grab_optional:w` . . . [23](#), [23](#), [430](#), [457](#)
`\c__scontents_hidden_space_str` [17](#), [168](#), [539](#), [635](#),
[709](#), [726](#), [804](#), [936](#)
`__scontents_if_nested:` [26](#)
`__scontents_if_nested:TF` [520](#), [580](#)
`__scontents_lastfrom_seq:n` [22](#), [392](#), [638](#), [712](#), [729](#)
`\l__scontents_macro_tmp_tl` . [16](#), [143](#), [477](#), [480](#), [481](#),
[635](#), [636](#)
`__scontents_make_control_chars_active:` . [471](#),
[504](#), [876](#)
`__scontents_meaningsc:n` [926](#), [931](#)
`__scontents_meaningsc_internal:nn` . . [918](#), [921](#)
`\l__scontents_name_seq_cmd_tl` [115](#), [710](#), [712](#), [727](#),
[729](#)
`\l__scontents_name_seq_env_tl` . . . [112](#), [636](#), [638](#)
`__scontents_nesting_decr:` [26](#), [522](#), [580](#)
`__scontents_nolig_list:` [673](#), [677](#)
`__scontents_norm_arg:n` [28](#), [691](#)
`__scontents_normalise_line_ends:N` [23](#), [441](#), [450](#)
`__scontents_optarg:nn` [305](#), [310](#), [312](#)
`__scontents_package_later_aux:Nn` . . . [331](#), [332](#)
`__scontents_par:` [175](#), [905](#)
`__scontents_parse_command_keys:n` . [19](#), [208](#), [256](#)
`__scontents_parse_command_keys:nn` [256](#)
`__scontents_parse_environment_keys:n` [19](#), [199](#),
[248](#)
`__scontents_parse_environment_keys:nn` . . [248](#)
`__scontents_parse_foreach_keys:n` . [20](#), [241](#), [264](#)
`__scontents_parse_foreach_keys:nn` [264](#)
`__scontents_parse_type_meaning_key:n` [246](#), [272](#)
`__scontents_parse_type_meaning_key:nn` . . [272](#)
`__scontents_parse_typemeaning_key:n` [20](#)
`__scontents_parse_version:w` [335](#), [336](#), [338](#)
`__scontents_parse_version_auxi:w` . . . [338](#), [339](#)
`__scontents_parse_version_auxii:w` . . [340](#), [341](#)
`__scontents_plain_disable_outer_par:` [885](#), [903](#)
`\l__scontents_print_cmd_bool` . . [22](#), [123](#), [711](#), [728](#)
`\l__scontents_print_env_bool` [22](#), [120](#), [637](#)
`__scontents_provides_aux:nn` [344](#), [345](#)
`__scontents_remove_leading_nl:n` [24](#), [484](#)
`__scontents_remove_leading_nl:nn` . . . [568](#), [573](#)
`__scontents_remove_leading_nl:w` [484](#)
`__scontents_require_auxi:wn` [310](#), [311](#)
`__scontents_require_auxii:wnw` [312](#), [321](#)
`__scontents_require_auxiii:n` [324](#), [326](#)
`__scontents_rescan_tokens:n` [30](#), [34](#), [173](#), [395](#), [541](#),
[742](#), [820](#)
`__scontents_ret:w` [24](#), [497](#), [505](#), [529](#), [544](#), [906](#)
`__scontents_Scontents_auxi:N` [691](#)
`__scontents_scontents_env_begin:` [405](#)
`__scontents_scontents_env_end:` [405](#)
`__scontents_Scontents_internal:nn` [691](#)
`\l__scontents_seq_item_int` . [16](#), [151](#), [279](#), [800](#), [803](#),
[924](#), [934](#)
`__scontents_set_active_eq:NN` [680](#), [876](#)
`__scontents_stararg:nn` [307](#), [352](#), [354](#)
`__scontents_start_after_option:w` . . [23](#), [24](#), [444](#),
[446](#)
`__scontents_start_environment:w` [423](#), [446](#)
`__scontents_stop_environment:` [427](#), [446](#)
`__scontents_store_to_seq:` [27](#)
`__scontents_store_to_seq:NN` [22](#), [400](#), [636](#), [710](#), [727](#)
`\l__scontents_storing_bool` . [17](#), [26](#), [155](#), [188](#), [556](#),
[584](#), [632](#)
`__scontents_tab:` [175](#), [904](#)
`\l__scontents_tab_width_int` [129](#), [864](#)
`__scontents_tabs_to_spaces:` . . [31](#), [836](#), [863](#), [939](#)
`\l__scontents_temp_bool` [643](#), [653](#), [656](#), [664](#)
`\g__scontents_temp_tl` . [16](#), [143](#), [394](#), [396](#), [398](#), [770](#),
[774](#), [775](#)
`\l__scontents_temp_tl` . [16](#), [143](#), [323](#), [324](#), [440](#), [441](#),
[442](#), [708](#), [709](#), [710](#), [723](#), [724](#), [726](#), [727](#), [802](#), [804](#), [805](#),
[806](#), [807](#), [933](#), [935](#), [936](#), [937](#), [939](#), [940](#)
`\l__scontents_tmpa_int` [151](#), [355](#), [361](#), [590](#), [595](#), [613](#)
`__scontents_tpestored_internal:nn` . [794](#), [797](#)
`__scontents_use_none_delimit_by_q_stop:w` [580](#)
`__scontents_verb_arg:w` [28](#), [691](#)
`__scontents_verb_arg_internal:n` [691](#)
`\l__scontents_verb_font_tl` [118](#), [671](#), [938](#)
`__scontents_verb_print:N` [30](#), [30](#), [791](#)
`__scontents_verb_print_EOL:` [816](#), [830](#)
`__scontents_verb_processor_iterate:w` . . [484](#)
`__scontents_verb_processor_output:n` . [26](#), [517](#),
[523](#), [528](#), [580](#)
`__scontents_verbatim_aux:` [645](#), [646](#)
`__scontents_vobeyspaces:` [645](#), [679](#)
`\l__scontents_writing_bool` . [17](#), [26](#), [155](#), [184](#), [189](#),
[487](#), [551](#), [582](#)
`__scontents_xobeysp:` [680](#), [681](#)
`__scontents_xverb:` [645](#), [833](#), [853](#)
`__scontents_xverb:w` [791](#)
`__scontents_zap_space:ww` [313](#), [318](#), [323](#)

<code>\Scontents*</code>	19
<code>\ScontentsCoreFileDate</code>	3, 94
<code>\ScontentsFileDate</code>	2, 10, 27, 94
<code>\ScontentsFileDescription</code>	6, 10, 28
<code>\ScontentsFileVersion</code>	5, 10, 23, 28
seq commands:	
<code>\seq_clear:N</code>	759
<code>\seq_clear_new:N</code>	952
<code>\seq_count:N</code>	379, 763, 946
<code>\seq_gput_right:Nn</code>	371
<code>\seq_if_exist:NTF</code>	369, 376
<code>\seq_item:Nn</code>	390, 394
<code>\seq_map_function:NN</code>	298, 495, 688
<code>\seq_new:N</code>	167, 290, 370
<code>\seq_put_right:Nn</code>	293, 779
<code>\seq_use:Nn</code>	771
<code>\setupsc</code>	2, 32, 909
skip commands:	
<code>\skip_set:Nn</code>	650
<code>\skip_vertical:N</code>	648
<code>\space</code>	27, 28, 344
<code>\startscontents</code>	405
<code>\startverbatimsc</code>	825
<code>\stopscontents</code>	405, 534
<code>\stopverbatimsc</code>	53, 847
str commands:	
<code>\c_backslash_str</code> ..	56, 469, 592, 601, 970, 972, 998, 1000, 1004, 1006, 1010, 1012, 1016, 1018
<code>\c_circumflex_str</code>	169
<code>\c_percent_str</code>	169, 537
<code>\str_const:Nn</code>	168
<code>\str_if_eq:nnTF</code>	327, 456, 539, 610
<code>\str_if_eq_p:nn</code>	537

T

\TeX and \LaTeX commands:

<code>\@</code>	355, 356, 361
<code>\@ifpackageloaded</code>	11
<code>\do@noligs</code>	31

tex commands:

<code>\tex_everypar:D</code>	674, 675
<code>\tex_kern:D</code>	872
<code>\tex_let:D</code>	891
<code>\tex_lowercase:D</code>	890
<code>\tex_newlinechar:D</code>	741, 819
<code>\tex_par:D</code>	652, 660, 666
<code>\tex_scantokens:D</code>	17, 173
<code>\tex_the:D</code>	675
<code>\tex_unpenalty:D</code>	675

tl commands:

<code>\c_space_tl</code>	175
<code>\tl_clear:N</code>	494
<code>\tl_const:Nn</code>	54
<code>\tl_gclear:N</code>	397, 773
<code>\tl_gset:Nn</code>	27, 347, 394, 770
<code>\tl_gset_rescan:Nnn</code>	42
<code>\tl_head:n</code>	456, 569
<code>\tl_if_blank:nTF</code> ..	252, 260, 266, 278, 283, 342, 367, 466, 514, 527, 814
<code>\tl_if_blank_p:n</code>	536
<code>\tl_if_empty:n</code>	180
<code>\tl_if_empty:NTF</code>	481, 806
<code>\tl_if_empty:nTF</code>	177, 276, 1050
<code>\tl_if_head_is_N_type:nTF</code>	454, 565
<code>\tl_if_novalue:nTF</code>	438, 699, 757, 801, 925
<code>\tl_log:N</code>	402, 805, 937
<code>\tl_new:N</code> ..	41, 143, 144, 145, 146, 147, 148, 149, 150
<code>\tl_put_right:Nn</code>	585, 635, 709, 726
<code>\tl_remove_once:Nn</code>	177, 177, 804, 936
<code>\tl_replace_all:Nnn</code> ...	177, 178, 451, 724, 935, 939
<code>\tl_rescan:nn</code>	17
<code>\tl_set:Nn</code> ..	185, 190, 214, 219, 323, 440, 480, 558, 708, 723, 758, 802, 933
<code>\tl_to_str:n</code>	16
<code>\tl_use:N</code>	671, 771, 786, 938

token commands:

<code>\token_if_eq_meaning:NNTF</code>	575
<code>\token_to_str:N</code>	868, 869
<code>\tt</code>	140
<code>\ttfamily</code>	139
<code>\typestored</code>	1, 5, 5, 16, 16, 19, 20, 30, 791

U

<code>\unprotect</code>	24
use commands:	

<code>\use:N</code>	29
<code>\use:n</code>	507, 553, 598, 785, 841
<code>\use_i:nn</code>	604
<code>\use_ii:nn</code>	606
<code>\use_none:n</code>	316, 352
<code>\use_none:nn</code>	352

V

<code>\verbatim</code>	642, 854
<code>\verbatimsc</code>	824
<code>verbatimsc</code>	5, 791

W

<code>\writestatus</code>	23
---------------------------------	----