

SCONTENTS

Stores L^AT_EX CONTENTS

V1.9 — 2020-01-21*

©2019–2020 by Pablo González†

CTAN: <https://www.ctan.org/pkg/scontents>

GitHub: <https://github.com/pablgonz/scontents>

Abstract

This package allows to store L^AT_EX code, including “*verbatim*”, in `\storedcontent` using the `\l3seq` module of `expl3`. The `\storedcontent` can be used as many times as desired in the document, additionally you can write to `\externalfiles` or show it in `\verb+verbatim style+`.

Contents

1 Motivation and Acknowledgments	1	5 Other commands provided	7
2 License and Requirements	1	5.1 The command \meaningsc	7
3 The scontents package	2	5.2 The command \countsc	7
3.1 Description of the package and load	2	5.3 The command \cleanseqsc	7
3.2 The TAB character	2	6 The scontents 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 scontents	3	7.3 From TeX-SX	9
4.2 The command \newenvsc	4	7.4 Customization of verbatimsc	11
4.3 The command \Scontents	4	8 Change history	14
4.4 The command \getstored	5	9 Index of Documentation	15
4.5 The command \foreachsc	5	10 References	15
4.6 The command \typestored	6	11 Implementation	16
4.7 The environment verbatimsc	6	12 Index of Implementation	41

1 Motivation and Acknowledgments

In L^AT_EX there is no direct way to record content for later use, although you can do this using `\macros`, recording `\verbatim content` 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 L^AT_EX³ 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 (lppl), version 1.3 or later (<http://www.latex-project.org/lppl.txt>). The software has the status “maintained”.

The **S**CONTENTS 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 v1.9, last revised 2020-01-21.

†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 L^AT_EX users

```
\usepackage{scontents}
```

or

```
\usepackage[⟨key = val⟩]{scontents}
```

The package options are not available for plain T_EX and ConTeXt, see 3.3.

For plain T_EX 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 L^AT_EX 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 T_EXLive 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 T_EX-TAB () 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. However, options set in the declaration of an environment (with `\newenvsc`) have precedence over options set with `\setupsc`.

Options in the optional arguments of environments and commands have the highest precedence, overriding both options in `\newenvsc`, and in `\setupsc`.

`verb-font = {⟨font family⟩}` default: `\ttfamily`

Sets the *font family* used to display the *stored content* for the `\typestored` and `\meaningsc` commands. 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`.

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.

<code>overwrite = {⟨true false⟩}</code>	default: <i>false</i>
Sets whether the <code>⟨files⟩</code> generated by <code>write-out</code> and <code>write-env</code> from the <code>scontents</code> environment will be rewritten. This key is available as a package option, for <code>\setupsc</code> , for <code>\Scontents*</code> and for the environment <code>scontents</code> .	
<code>print-all = {⟨true false⟩}</code>	default: <i>false</i>
It is a <code>⟨meta-key⟩</code> that sets the <code>print-env</code> key of the <code>scontents</code> environment and the <code>print-cmd</code> key of the <code>\Scontents</code> command. This key is only available as a package option or using <code>\setupsc</code> .	
<code>force-eol = {⟨true false⟩}</code>	default: <i>false</i>
Sets if the end of line for the <code>⟨stored content⟩</code> is hidden or not. This key is necessary only if the last line is the closing of some environment defined by the <code>fancyvrb</code> package as <code>\end{Verbatim}</code> or another environment that does not support a comments “%” after closing <code>\end{⟨env⟩}%</code> . This key is available for the <code>scontents</code> environment and the <code>\Scontents</code> command.	
<code>width-tab = {⟨integer⟩}</code>	default: 1
Sets the equivalence in <code>⟨spaces⟩</code> for the character TAB used when displaying stored content in <i>verbatim style</i> . The value must be a <code>⟨positive integer⟩</code> . This key is available for the <code>\typestored</code> and the <code>\meaningsc</code> commands.	

3.4 Options Overview

Summary of available options:

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

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.

Notes for plain T_EX and ConT_EXt users

In plain T_EX there is not environments as in L^AT_EX. Instead of using the environment `scontents`, one should use a *pseudo environment* delimited by `\scontents` and `\endscontents`.

```
\scontents
\endscontents
```

ConT_EXt users should use `\startscontents` and `\stopscontents`.

```
\startscontents[⟨keyval list⟩]
  ⟨env contents⟩
\stopscontents
```

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, relative or absolute paths are not supported. If *file.ext* does not exist, it will be created or overwritten if the `overwrite` key is used.

The characters TABs will be written in *file.ext* and the *contents* will be stored in the *sequence* established at that time. X_LT_EX 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, relative or absolute paths are not supported. If *file.ext* does not exist, it will be created or overwritten if the `overwrite` key is used.

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_LT_EX 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 \newenvsc

```
\newenvsc{⟨env name⟩}[⟨initial keys⟩]
```

The command `\newenvsc` allows you to create *new environments* based on the same characteristics of the `scontents` environment. The values entered in `[⟨initial keys⟩]` will be considered as the default values for this new environment and the valid *keys* are `store-env` and `print-env`. For example:

```
\newenvsc{myenvstore}[store-env=myseq,print-env=false]
```

created the `myenvstore` environment that stored the content in the `myseq` sequence and will not display the content when it is executed.

4.3 The command \Scontents

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

The `\Scontents` command reads the $\{\langle argument \rangle\}$ 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 `fverextra` 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 $\{\langle argument \rangle\}$ under *verbatim* category code regimen. If its first delimiter is a brace, it will be assumed that the $\{\langle argument \rangle\}$ is nested into braces. Otherwise it will be assumed that the ending of that $\langle argument \rangle$ 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. The starred argument (*) and $[\langle key = val \rangle]$ must not be separated by horizontal spaces between them and the command.

Both versions can be used anywhere in the document and cannot be used as an $\langle argument \rangle$ for other command.

Options for command

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

`store-cmd = {\langle seq name \rangle}` default: *contents*

Sets the name of the $\langle sequence \rangle$ in which the contents will be stored. If the sequence does not exist, it will be created globally.

`print-cmd = {\langle true | false \rangle}` default: *false*

Sets if the $\langle stored content \rangle$ is displayed or not at the time of running the command. The content is extracted from the $\langle sequence \rangle$ in which it is stored.

Options only for the starred version

`write-cmd = {\langle file.ext \rangle}` default: *not used*

Sets the name of the $\langle external file \rangle$ in which the $\langle contents \rangle$ of the $\{\langle argument \rangle\}$ will be written. The $\langle file.ext \rangle$ will be created in the working directory, relative or absolute paths are not supported. If $\langle file.ext \rangle$ does not exist, it will be created or overwritten if the `overwrite` key is used.

The characters TABs will be written in $\langle file.ext \rangle$ and the $\langle contents \rangle$ will be stored in the $\langle sequence \rangle$ established at that time. Xe^LT_EX users using the TAB character must add `-8bit` at the command line, otherwise you will get T_EX-TAB (`\^I`) in $\langle file.ext \rangle$.

`write-out = {\langle file.ext \rangle}` default: *not used*

Sets the name of the $\langle external file \rangle$ in which the $\langle contents \rangle$ of the $\{\langle argument \rangle\}$ will be written. The $\langle file.ext \rangle$ will be created in the working directory, relative or absolute paths are not supported. If $\langle file.ext \rangle$ does not exist, it will be created or overwritten if the `overwrite` key is used.

The characters TABs will be written in $\langle file.ext \rangle$, the rest of the $\langle keys \rangle$ will not be available and the $\langle contents \rangle$ will NOT be stored in any $\langle sequence \rangle$. Xe^LT_EX users using the TAB character must add `-8bit` at the command line, otherwise you will get T_EX-TAB (`\^I`) in $\langle file.ext \rangle$.

The key `overwrite` is available for this command.

4.4 The command `\getstored`

`\getstored` `\getstored[\langle index \rangle]{\langle seq name \rangle}`

The command `\getstored` gets the content stored in $\{\langle seq name \rangle\}$ according to the $\langle index \rangle$ in which it was stored. The command is robust and can be used as an $\langle argument \rangle$ for another command. If the optional argument is not passed, the default value is the “last element” stored in $\{\langle seq name \rangle\}$.

4.5 The command `\foreachsc`

`\foreachsc` `\foreachsc[\langle key = val \rangle]{\langle seq name \rangle}`

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

Options for command

`sep = {\langle code \rangle}` default: *empty*

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

`step = {\langle integer \rangle}` 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}`.

4.6 The command \typestored

`\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.

4.7 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 `verbatim` package is not compatible with the implementation of the `verbatimsc` environment.

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

Using the package `fancyvrb`:

```
\makeatletter
\let\verb@sc@undefined
\let\endverb@sc@undefined
\makeatother
\DefineVerbatimEnvironment{verbatimsc}{Verbatim}{numbers=left}
```

Using the package `minted`:

```
\makeatletter
\let\verb@sc@undefined
\let\endverb@sc@undefined
\makeatother
\usepackage{minted}
\newminted{tex}{linenos}
\newenvironment{verbatimsc}{\VerbatimEnvironment\begin{texcode}}{\end{texcode}}
```

Using the package `listings`:

```
\makeatletter
\let\verb@sc@undefined
\let\endverb@sc@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 `{⟨seq name⟩}`. 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.

5.2 The command \countsc

`\countsc{⟨seq name⟩}`

The command `\countsc` count a number of contents stored in `{⟨seq name⟩}`.

5.3 The command \cleanseqsc

`\cleanseqsc{⟨seq name⟩}`

The command `\cleanseqsc` remove all contents stored in `{⟨seq name⟩}`.

6 The SCONTENTS package in action

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

Abstract

This package allows to store L^AT_EX code, including “*verbatim*”, in `(sequences)` using the `l3seq` module of `expl3`. The `⟨stored content⟩` can be used as many times as desired in the document, additionally you can write to `⟨external files⟩` or show it in `(verbatim style)`.

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 Con^TE_Xt “*buffers*” by making use `(sequences)`.

I've only written:

```

\begin{abstract}
This package allows to store \holo{LaTeX} code, including \enquote{\emph{verbatim}},
in \mymeta{sequences} using the \mypkg{l3seq} module of \mypkg{expl3}. The \mymeta{stored
content} can be used as many times as desired in the document, additionally you can write
to \mymeta{external files} or show it in \mymeta{verbatim style}.
\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 `\holo{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 This package allows to store \holo{LaTeX} code, including \enquote{\emph{verbatim}},
4 in \mymeta{sequences} using the \mypkg{l3seq} module of \mypkg{expl3}. The \mymeta{stored
5 content} can be used as many times as desired in the document, additionally you can write
6 to \mymeta{external files} or show it in \mymeta{verbatim style}.
7 \end{abstract}
8 \end{scontents}

```

and

```

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

I stored the content in memory and then ran `\getstored` and `\typestored`. 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. The examples are attached to this documentation and can be extracted from your PDF viewer or from the command line by running:

```
$ pdfdetach -saveall scontents.pdf
```

and then you can use the excellent `arara`³ tool to compile them.

7.1 From answers package

Example 1

Adaptation of example 1 of the package `answers` .

```

1 % arara: pdflatex
2 % arara: clean: { extensions: [ aux, log ] }
3 \documentclass{article}
4 \usepackage[store-cmd=solutions]{scontents}
5 \newtheorem{ex}{Exercise}
6 \setlength{\parindent}{0pt}
7 \pagestyle{empty}
8 \begin{document}
9 \section{Problems}
10 \begin{ex}
11 First exercise
12 \Scontents{First solution.}
13 \end{ex}
14
15 \begin{ex}
16 Second exercise
17 \Scontents{Second solution.}
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 % arara: pdflatex
2 % arara: clean: { extensions: [ aux, log ] }
3 \documentclass{article}
4 \usepackage[store-env=defexercise,store-cmd=defexercise]{scontents}
5 \setlength{\parindent}{0pt}
6 \pagestyle{empty}
7 \begin{document}
8 % not starred
9 \Scontents{
10 Prove that  $x^n+y^n=z^n$  is not solvable in positive integers if  $n$  is at
11 most $-3$. \par
12 }
13 % starred
```

³The cool TeX automation tool: <https://www.ctan.org/pkg/arara>

```

14 \Scontents*|Refute the existence of black holes in less than $140$ characters.|%
15 % write environment to \jobname.txt
16 \begin{scontents}[write-env=\jobname.txt]
17 \def\NSA{\NSA}%
18 Prove that factorization is easily done via probabilistic algorithms and
19 advance evidence from knowledge of the names of its employees in the
20 seventies that the \NSA\ has known that for 40 years.\par
21 \end{scontents}
22 % see all stored
23 \begin{itemize}
24 \foreachsc[before={\item }]{\defexercise}
25 \end{itemize}
26 % \getstored are robust :)
27 \section{\getstored[2]{\defexercise}}
28 \end{document}

```

7.3 From TeX-SX

Example 3

Adapted from LaTeX equivalent of ConTeXt buffers .

```

1 % arara: pdflatex
2 % arara: clean: { extensions: [ aux, log ] }
3 \documentclass{article}
4 \usepackage[store-cmd=tikz]{scontents}
5 \usepackage{tikz}
6 \setlength{\parindent}{0pt}
7 \pagestyle{empty}
8 \Scontents*{\matrix{ \node (a) {$a$} ; & \node (b) {$b$} ; \\ } ;}
9 \Scontents*{\matrix[ampersand replacement=&]
10 { \node (a) {$a$} ; & \node (b) {$b$} ; \\ } ;}
11 \Scontents*{\matrix{\node (a) {$a$} ; & \node (b) {$b$} ; \\ } ;}
12 \begin{document}
13 \section{tikzpicture}
14 \begin{tikzpicture}
15 \getstored[1]{tikz}
16 \end{tikzpicture}
17
18 \begin{tikzpicture}
19 \getstored[2]{tikz}
20 \end{tikzpicture}
21
22 \begin{tikzpicture}
23 \getstored{tikz}
24 \end{tikzpicture}
25
26 \begin{scontents}[store-env=buffer]
27 Hello World!
28
29 This is a \verb+|fake poor man's buffer :)| .
30 \end{scontents}
31
32 \section{source tikz}
33 \typestored[1]{tikz}
34 \typestored[2]{tikz}
35 \typestored[3]{tikz}
36
37 \section{fake buffer}
38 \subsection{real content}
39 \getstored[1]{buffer}
40 \subsection{verbatim style}
41 \typestored[1]{buffer}
42 \subsection{meaning}
43 \meaningsc[1]{buffer}
44
45 \section{tikz again}
46 \foreachsc[before={\begin{tikzpicture}},after={\end{tikzpicture}},sep={\\[10pt]}]{tikz}
47 \end{document}

```

Example 4

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

```

1 % arara: pdflatex
2 % arara: clean: { extensions: [ aux, log] }
3 \documentclass{article}
4 \usepackage{scontents}
5 \setlength{\parindent}{0pt}
6 \pagestyle{empty}
7 \begin{document}
8 \begin{scontents}[store-env=main]
9 Something for main A.
10 \end{scontents}
11
12 \begin{scontents}[store-env=main]
13 Something for \verb|main B|.
14 \end{scontents}
15
16 \begin{scontents}[store-env=other]
17 Something for \verb|other|.
18 \end{scontents}
19
20 \textbf{Let's print them}
21
22 This is first stored in main: \getstored[1]{main}\par
23 This is second stored in main: \getstored{main}\par
24 This is stored in other: \getstored{other}
25
26 \textbf{Print all of stored in main}\par
27 \foreachsc[sep={\\[10pt]}]{main}
28 \end{document}
```

Example 5

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

```

1 % arara: pdflatex
2 % arara: clean: { extensions: [ aux, log] }
3 \documentclass{article}
4 \usepackage{scontents}
5 \setlength{\parindent}{0pt}
6 \pagestyle{empty}
7 \begin{document}
8 \section{Problem stated the first time}
9 \begin{scontents}[print-env=true,store-env=problem]
10 This is normal text.
11 \verb|This is from the verb command.|
12 \verb*|This is from the verb* command.|
13 This is normal text.
14 \begin{verbatim}
15 This is from the verbatim environment:
16 &%{}~
17 \end{verbatim}
18 \end{scontents}
19 \section{Problem restated}
20 \getstored[1]{problem}
21 \section{Problem restated once more}
22 \getstored[1]{problem}
23 \end{document}
```

Example 6

Adapted from [Environment hiding its content](#) .

```

1 % arara: pdflatex
2 % arara: clean: { extensions: [ aux, log] }
3 \documentclass[10pt]{article}
4 \usepackage{scontents}
5 \newenvsc{forshort}[store-env=forshort,print-env=false]
6 \setlength{\parindent}{0pt}
7 \pagestyle{empty}
```

```

8  \begin{document}
9
10 Something in the whole course.
11
12 \begin{forshort}
13     Just a summary...
14 \end{forshort}
15
16 \end{document}

```

7.4 Customization of `verbatimsc`

Example 7

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

```

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

```

Example 8

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

```

1 % arara: pdflatex
2

```

```

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

```

Example 9

Customization of `verbatimsc` using the `minted` package 

```

1 % arara: xelatex : {shell: true, options: [-8bit]}
2 % arara: clean: { extensions: [ aux, log] }
3 \documentclass{article}
4 \usepackage{scontents}
5 \makeatletter
6 \let\verbatimsc@\undefined
7 \let\endverbatimsc@\undefined
8 \makeatother
9 \usepackage{minted}
10 \newminted{tex}{linenos}
11 \newenvironment{verbatimsc}{\VerbatimEnvironment\begin{texcode}}{\end{texcode}}
12 \pagestyle{empty}
13 \setlength{\parindent}{0pt}
14 \begin{document}
15 \section{Test \texttt{\textbackslash textbackslash begin\{scontents\}} whit \texttt{\textbackslash texttt{minted}}}
16 Test \verb+\scontents+ + \par
17

```

```
18 \begin{scontents}[overwrite,write-env=\jobname.tsc,force-eol=true]
19 Using \verb+scontents+ env no \verb+[key=val]+, save in seq \verb+contents+
20 with index 1.\par
21
22 Prove new \Verb*{ new fextra whit braces } and environment \verb+Verbatim*+
23 \begin{Verbatim}[obeytabs, showtabs, tab=\rightarrowfill, tabcolor=red]
24 No tab
25     One real tab
26             Two real Tab plus      one tab
27 \end{Verbatim}
28 \end{scontents}
29
30 \section{See \Verb{\jobname.tsc}}
31 Read \Verb{\jobname.tsc} (shows TABs as red arrows):
32 \VerbatimInput[obeytabs, showtabs, tab=\rightarrowfill, tabcolor=red]{\jobname.tsc}
33
34 \section{Test \texttt{\textbackslash Scontents} whit \texttt{\minted}}
35
36 \Scontents{ We have coded \par this in \LaTeX: $E=mc^2$.}
37
38 \section{Test \texttt{\textbackslash getstored}}
39 \getstored[1]{contents}\par
40 \getstored{contents}
41
42 \section{Test \texttt{\textbackslash typestored}}
43 \typestored[1]{contents}
44 \typestored[2]{contents}
45 \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`.

- v1.9 (ctan), 2020-01-21**
 - Update and improvements in the internal code.
 - Updating the generic code for I/O verification.
 - Add `write-cmd` and `write-out` keys for `\Scontents*`.
 - Fix `sep` key in `\foreachsc`.
- v1.8 (ctan), 2019-11-18**
 - Add `\newenvsc` command.
 - Fix nested environment in plain TeX and ConTeXt.
 - Modified default value in `\getstored`.
 - Add `overwrite` key to reduce I/O operations.
 - Deleted an unnecessary group in the code.
- v1.7 (ctan), 2019-10-29**
 - The `verbatimsc` environment was rewritten.
 - Minor adjustments in documentation.
- 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.
 - Remove spurious warning.
- 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.
 - The environment can now nest.
- v1.3 (ctan), 2019-09-24**
 - 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

The italic numbers denote the pages where the corresponding entry is described.

C

Commands provide by `\Scontents`:

<code>\Scontents*</code>	3, 5
<code>\Scontents</code>	2, 3, 5
<code>\cleanseqsc</code>	7
<code>\countsc</code>	7
<code>\endscontents</code>	4
<code>\foreachsc</code>	5
<code>\getstored</code>	3–5
<code>\meaningsc</code>	2, 3, 7
<code>\newenvsc</code>	2, 4
<code>\scontents</code>	4
<code>\setupsc</code>	2–5
<code>\startscontents</code>	4
<code>\stopscontents</code>	4
<code>\typestored</code>	2, 3, 6

E

Environment provide by `\Scontents`:

<code>scontents</code>	2–4
<code>verbatimsc</code>	6, 11, 12

Environments

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

K

Keys

<code>after</code>	6
<code>before</code>	6
<code>force-eol</code>	3–5
<code>overwrite</code>	3–5
<code>print-all</code>	3
<code>print-cmd</code>	3, 5
<code>print-env</code>	3, 4
<code>sep</code>	5
<code>start</code>	6

<code>step</code>	5
<code>stop</code>	6
<code>store-all</code>	2, 3
<code>store-cmd</code>	2, 3, 5
<code>store-env</code>	2–4
<code>verb-font</code>	2, 3
<code>width-tab</code>	3, 6, 7
<code>wrapper</code>	6
<code>write-cmd</code>	3, 5
<code>write-env</code>	3, 4
<code>write-out</code>	3–5

L

<code>\lstinline</code>	5
-------------------------	---

M

<code>\meaning</code>	7
-----------------------	---

P

Packages

<code>answers</code>	8
<code>expl3</code>	1, 7, 14
<code>fancyvrb</code>	2, 3, 6, 11
<code>filecontentsdef</code>	1, 2, 7, 8, 14
<code>fvextra</code>	5
<code>l3keys2e</code>	1
<code>l3seq</code>	1, 7
<code>listings</code>	2, 5, 6, 11
<code>minted</code>	6, 12
<code>scontents</code>	1, 2, 7, 8, 14
<code>tcolorbox</code>	11
<code>verbatim</code>	6
<code>xparse</code>	1

V

<code>\Verb</code>	5
<code>\verb</code>	5

10 References

- [1] The L^AT_EX3 Project. “The `expl3` package”. Available from CTAN, <https://www.ctan.org/pkg/expl3>, 2020.
- [2] The L^AT_EX3 Project. “The `xparse` package”. Available from CTAN, <https://www.ctan.org/pkg/xparse>, 2020.
- [3] The L^AT_EX3 Project. “The `l3keys2e` package”. Available from CTAN, <https://www.ctan.org/pkg/l3keys2e>, 2020.
- [4] WRIGHT, JOSEPH. “Programming key-value in `expl3`”. Available from TUGBOAT, <https://www.tug.org/TUGboat/tb31-1/tb97wright-l3keys.pdf>, 2010.

11 Implementation

The most recent publicly released version of `scontents` is available at CTAN: <https://www.ctan.org/pkg/scontents>. Historical and developmental versions are available at <https://github.com/pablgonz/scontents>. While general feedback via email is welcomed, specific bugs or feature requests should be reported through the issue tracker: <https://github.com/pablgonz/scontents/issues>.

11.1 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\ScontentsFileDate{2020-01-21}%
3 <core>\def\ScontentsCoreFileDate{2020-01-21}%
4 (*loader)
5 \def\ScontentsFileVersion{1.9}%
6 \def\ScontentsFileDescription{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} {\ScontentsFileDate} {\ScontentsFileVersion} {\ScontentsFileDescription}
11 \ifpackageloaded { verbatim }
12 {
13     \msg_set:nnn { scontents } { unsupported-verbatim }
14     {
15         The~implementation~of~the~'verbatimsc'~environment~used~by~
16         \iow_char:N \typestored~is~not~compatible~with~package~'verbatim'.~
17         Review~the~documentation~and~redefine~the~'verbatimsc'~environment.
18     }
19     \msg_warning:nn { scontents } { unsupported-verbatim }
20 }
21 </latext>
```

The Plain and ConTeXt loaders are similar (probably because I don't know how to make a proper ConTeXt 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\ScontentsFileVersion}
24 <context>\unprotect
25 \input expl3-generic.tex
26 \ExplSyntaxOn
27 \tl_gset:c { ver @ scontents . sty } { \ScontentsFileDate\space
28   \ScontentsFileVersion\space \ScontentsFileDescription }
29 \iow_log:x { Package: ~ scontents ~ \use:c { ver @ scontents . sty } }
30 </!latext>
```

In Plain, check that the package isn't being loaded twice (`LATEX` and ConTeXt 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 
```

- 49 `/|`
50 `}`
- 51 `<`
52 `<`
53 `<`
54 `<`
55 `{`
56 `<`
57 `<`
58 `<`
59 `<`
60 `<`
61 `<`
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 
```

- 65 `<`
66 `<`
67 `<`

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

```

68 
```

- 68 `<`
69 `<`
70 `<`
71 `<`
72 `<`
73 `<`
74 `<`
75 `<`
76 `<`
77 `<`
78 `<`
79 `<`
80 `<`
81 `<`
82 `<`
83 `<`
84 `<`
85 `<`
86 `<`
87 `<`
88 `<`
89 `<`
90 `<`
91 `<`
92 `<`
93 `<`
94 `<`
95 `<`
96 `<`
97 `<`
98 `<`
99 `<`
100 `<`
101 `<`
102 `<`
103 `<`
104 `<`
105 `<`
106 `<`
107 `<`
108 `<`
109 `<`

```

110   \fi
111   \next

```

11.2 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.

```

112 \keys_define:nn { scontents }
113   {
114     store-env .tl_set:N      = \l__scontents_name_seq_env_tl,
115     store-env .initial:n    = contents,
116     store-env .value_required:n = true,
117     store-cmd .tl_set:N      = \l__scontents_name_seq_cmd_tl,
118     store-cmd .initial:n    = contents,
119     store-cmd .value_required:n = true,
120     verb-font .tl_set:N      = \l__scontents_verb_font_tl,
121     verb-font .value_required:n = true,
122     print-env .bool_set:N    = \l__scontents_print_env_bool,
123     print-env .initial:n    = false,
124     print-env .default:n     = true,
125     print-cmd .bool_set:N    = \l__scontents_print_cmd_bool,
126     print-cmd .initial:n    = false,
127     print-cmd .default:n     = true,
128     force-eol .bool_set:N    = \l__scontents_forced_eol_bool,
129     force-eol .initial:n    = false,
130     force-eol .default:n     = true,
131     overwrite .bool_set:N    = \l__scontents_overwrite_bool,
132     overwrite .initial:n    = false,
133     overwrite .default:n     = true,
134     width-tab .int_set:N     = \l__scontents_tab_width_int,
135     width-tab .initial:n    = 1,
136     width-tab .value_required:n = true,
137     print-all .meta:n        = { print-env = #1 , print-cmd = #1 },
138     print-all .default:n     = true,
139     store-all .meta:n        = { store-env = #1 , store-cmd = #1 },
140     store-all .value_required:n = true
141   }
142 /core
```

`\keys_define:nn { scontents }`
`\iflatex { verb-font .initial:n = \ttfamily }`
`\else { verb-font .initial:n = \tt }`

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

```

146 \iflatex \ProcessKeysOptions { scontents }
147 
```

11.3 Internal variables

Now we declare the internal variables we will use.

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

```

148 \tl_new:N \l__scontents_macro_tmp_tl
149 \tl_new:N \l__scontents_fname_out_tl
150 \tl_new:N \l__scontents_temp_tl
151 \tl_new:N \l__scontents_file_tl
152 \tl_new:N \g__scontents_temp_tl
153 \tl_new:N \l__scontents_FOREACH_name_seq_tl
154 \tl_new:N \l__scontents_FOREACH_before_tl
155 \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 or \meaningsc. \l__scontents_env_nesting_int stores the current nesting level of the scontents environment. \l__scontents_foreach_stop_int will save the value at which the \foreachsc loop will stop.

```

156 \int_new:N \l__scontents_foreach_stop_int
157 \int_new:N \l__scontents_seq_item_int
158 \int_new:N \l__scontents_env_nesting_int
159 \int_new:N \l__scontents_tmpa_int

```

(End definition for \l__scontents_seq_item_int and others.)

\l__scontents_writing_bool
\l__scontents_storing_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.

```

160 \bool_new:N \l__scontents_writing_bool
161 \bool_set_false:N \l__scontents_writing_bool
162 \bool_new:N \l__scontents_storing_bool
163 \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
\l__scontents_foreach_after_bool
\l__scontents_foreach_stop_bool
\l__scontents_foreach_wrapper_bool

Boolean variables used by the \foreachsc loop.

```

164 \bool_new:N \l__scontents_foreach_before_bool
165 \bool_set_false:N \l__scontents_foreach_before_bool
166 \bool_new:N \l__scontents_foreach_after_bool
167 \bool_set_false:N \l__scontents_foreach_after_bool
168 \bool_new:N \l__scontents_foreach_stop_bool
169 \bool_set_false:N \l__scontents_foreach_stop_bool
170 \bool_new:N \l__scontents_foreach_wrapper_bool
171 \bool_set_false:N \l__scontents_foreach_wrapper_bool
172 \bool_new:N \l__scontents_writable_bool

```

(End definition for \l__scontents_foreach_before_bool and others.)

\l__scontents_foreach_print_seq

The \l__scontents_foreach_print_seq is the sequence used by \foreachsc.

```
173 \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 TeX 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.

```

174 \str_const:Nx \c__scontents_hidden_space_str
175 { \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
\q__scontents_mark

Some quarks used along the code as macro delimiters.

```

176 \quark_new:N \q__scontents_stop
177 \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.

```
178 \iow_new:N \l__scontents_file_iow
```

(End definition for \l__scontents_file_iow.)

__scontents_rescan_tokens:n

\tl_rescan:nn 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.

```

179 \cs_new_protected:Npn \__scontents_rescan_tokens:n #1 { \tex_scantokens:D {#1} }
180 \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:
 181 \cs_new:Npx \__scontents_tab: { \c_space_tl }
 182 \cs_new:Npn \__scontents_par: { ^J ^J }
```

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

`\tl_remove_once:NV` Some nonstandard variants.

```
183 \cs_generate_variant:Nn \tl_remove_once:Nn { NV }
184 \cs_generate_variant:Nn \tl_replace_all:Nnn { Nx, Nxx, Nnx }
185 \cs_generate_variant:Nn \msg_error:nnnn { nnx }
186 \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`.)

11.4 Defining keys for the environment and commands

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

11.4.1 Keys for environment `scontents`

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

```
187 \keys_define:nn { scontents / scontents }
188 {
189   write-env .code:n      = {
190     \bool_set_true:N \l__scontents_writing_bool
191     \tl_set:Nn \l__scontents_fname_out_tl {\#1}
192   },
193   write-out .code:n     = {
194     \bool_set_false:N \l__scontents_storing_bool
195     \bool_set_true:N \l__scontents_writing_bool
196     \tl_set:Nn \l__scontents_fname_out_tl {\#1}
197   },
198   write-env .value_required:n = true,
199   write-out .value_required:n = true,
200   print-env .meta:nn       = { scontents } { print-env = #1 },
201   print-env .default:n    = true,
202   store-env .meta:nn       = { scontents } { store-env = #1 },
203   force-eol .meta:nn       = { scontents } { force-eol = #1 },
204   force-eol .default:n    = true,
205   overwrite .meta:nn       = { scontents } { overwrite = #1 },
206   overwrite .default:n    = true,
207   unknown .code:n         = { \__scontents_parse_environment_keys:n {\#1} }
208 }
```

11.4.2 Keys for command `\Scontents`

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

```
209 \keys_define:nn { scontents / Scontents }
210 {
211   write-cmd .code:n      = {
212     \bool_set_true:N \l__scontents_writing_bool
213     \tl_set:Nn \l__scontents_fname_out_tl {\#1}
214   },
215   write-out .code:n     = {
216     \bool_set_false:N \l__scontents_storing_bool
217     \bool_set_true:N \l__scontents_writing_bool
218     \tl_set:Nn \l__scontents_fname_out_tl {\#1}
219   },
220   write-env .value_required:n = true,
221   write-out .value_required:n = true,
222   print-cmd .meta:nn       = { scontents } { print-cmd = #1 },
223   print-cmd .default:n    = true,
224   store-cmd .meta:nn       = { scontents } { store-cmd = #1 },
225   force-eol .meta:nn       = { scontents } { force-eol = #1 },
226   force-eol .default:n    = true,
227   overwrite .meta:nn       = { scontents } { overwrite = #1 },
228   overwrite .default:n    = true,
```

```

229     unknown .code:n      = { \__scontents_parse_command_keys:n [#1] }
230   }

```

11.4.3 Keys for command \foreachsc

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

```

231 \keys_define:nn { scontents / foreachsc }
232 {
233   before .code:n      = {
234     \bool_set_true:N \l__scontents_foreach_before_bool
235     \tl_set:Nn \l__scontents_foreach_before_tl [#1]
236   },
237   before .value_required:n = true,
238   after .code:n       = {
239     \bool_set_true:N \l__scontents_foreach_after_bool
240     \tl_set:Nn \l__scontents_foreach_after_tl [#1]
241   },
242   after .value_required:n = true,
243   start .int_set:N     = \l__scontents_foreach_start_int,
244   start .value_required:n = true,
245   start .initial:n    = 1,
246   stop .code:n        = {
247     \bool_set_true:N \l__scontents_foreach_stop_bool
248     \int_set:Nn \l__scontents_foreach_stop_int [#1]
249   },
250   stop .value_required:n = true,
251   step .int_set:N     = \l__scontents_foreach_step_int,
252   step .value_required:n = true,
253   step .initial:n    = 1,
254   wrapper .code:n     = {
255     \bool_set_true:N \l__scontents_foreach_wrapper_bool
256     \cs_set_protected:Npn
257     \__scontents_foreach_wrapper:n ##1 [#1]
258   },
259   wrapper .value_required:n = true,
260   sep .tl_set:N        = \l__scontents_foreach_sep_tl,
261   sep .initial:n      = {},
262   sep .value_required:n = true,
263   unknown .code:n     = { \__scontents_parse_foreach_keys:n [#1] }
264 }

```

11.4.4 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>`.

```

265 \keys_define:nn { scontents / typemeaning }
266 {
267   width-tab .meta:nn = { scontents } { width-tab = #1 },
268   unknown .code:n  = { \__scontents_parse_type_meaning_key:n [#1] }
269 }

```

11.5 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>`.

11.5.1 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.

```

270 \cs_new_protected:Npn \__scontents_parse_environment_keys:n #1
271   { \exp_args:NV \__scontents_parse_environment_keys:nn \l_keys_key_tl [#1] }
272 \cs_new_protected:Npn \__scontents_parse_environment_keys:nn #1#2
273   {
274     \tl_if_blank:nTF [#2]
275       { \msg_error:nnn { scontents } { env-key-unknown } [#1] }
276       { \msg_error:nnnn { scontents } { env-key-value-unknown } [#1] { #2} }
277   }

```

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

11.5.2 Undefined keys for \scontents and \Scontents*

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

```

278 \cs_new_protected:Npn \_\_scontents_parse_command_keys:n #1
279   { \exp_args:NV \_\_scontents_parse_command_keys:nn \l_keys_key_tl {#1} }
280 \cs_new_protected:Npn \_\_scontents_parse_command_keys:nn #1#2
281   {
282     \tl_if_blank:nTF {#2}
283       { \msg_error:nnn { scontents } { cmd-key-unknown } {#1} }
284       { \msg_error:nnnn { scontents } { cmd-key-value-unknown } {#1} {#2} }
285   }

```

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

11.5.3 Undefined keys for \foreachsc

We check the $\langle keys \rangle$ 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.

```

286 \cs_new_protected:Npn \_\_scontents_parse_foreach_keys:nn #1#2
287   {
288     \tl_if_blank:nTF {#2}
289       { \msg_error:nnn { scontents } { for-key-unknown } {#1} }
290       { \msg_error:nnnn { scontents } { for-key-value-unknown } {#1} {#2} }
291   }
292 \cs_new_protected:Npn \_\_scontents_parse_foreach_keys:n #1
293   { \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`.)

11.5.4 Undefined keys for \typestored and \meaningsc

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[][(options)]{...}` syntax, we'll make the commands have a single optional argument which is processed by `\l_3keys`, and the unknown keys are brought here to `__scontents_parse_typemeaning_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.

```

294 \cs_new_protected:Npn \_\_scontents_parse_type_meaning_key:n #1
295   { \exp_args:NV \_\_scontents_parse_type_meaning_key:nn \l_keys_key_tl {#1} }
296 \cs_new_protected:Npn \_\_scontents_parse_type_meaning_key:nn #1#2
297   {
298     \tl_if_empty:fTF { \int_to_roman:n { -0 #1 } }
299     {
300       \tl_if_blank:nTF {#2}
301         { \int_set:Nn \l_scontents_seq_item_int {#1} }
302         { \msg_error:nnnn { scontents } { type-key-value-unknown } {#1} {#2} }
303     }
304     {
305       \tl_if_blank:nTF {#2}
306         { \msg_error:nnn { scontents } { type-key-unknown } {#1} }
307         { \msg_error:nnnn { scontents } { type-key-value-unknown } {#1} {#2} }
308     }
309   }

```

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

11.6 Compatibility layer with Plain

When loading the package outside of L^AT_EX we can't usually use xparse. However since xparse doesn't actually hold any dependency with L^AT_EX 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 L^AT_EX kernel (greatly simplified).

```

310 </core>
311 <*loader&!latex>
312 \seq_new:N \l_scontents_compat_seq

```

```

313 \cs_new_protected:Npn \__scontents_compat_redefine:Npn #1
314 {
315     \seq_put_right:Nn \l__scontents_compat_seq {#1}
316     \cs_set_eq:cN { __scontents_saved_\cs_to_str:N #1: } #1
317     \cs_new_protected:Npn #1
318 }
319 \cs_new_protected:Npn \__scontents_compat_restore:
320 {
321     \seq_map_function:NN \l__scontents_compat_seq \__scontents_compat_restore:N
322 }
323 \cs_set_eq:Nc #1 { __scontents_saved_\cs_to_str:N #1: }
324 \cs_undefine:c { __scontents_saved_\cs_to_str:N #1: }
325 }
326 \cs_generate_variant:Nn \__scontents_compat_redefine:Npn { c }
327 \cs_new_protected:Npn \__scontents_optarg:nn #1 #2
328 {
329     \peek_charcode_ignore_spaces:NTF [ {#1} {#1[#2]} ]
330     \cs_new_protected:Npn \__scontents_stararg:nn #1 #2
331     {
332         \peek_charcode_remove_ignore_spaces:NTF * {#1} {#2}
333     }
334     \__scontents_compat_redefine:Npn \RequirePackage
335     {
336         \__scontents_optarg:nn { \__scontents_require_auxi:wn } { }
337     }
338     \cs_new_protected:Npn \__scontents_require_auxi:wn [#1] #2
339     {
340         \__scontents_optarg:nn { \__scontents_require_auxii:wwn [##1][#2] } { }
341     }
342     \cs_new:Npn \__scontents_zap_space:ww #1~#2
343     {
344         #1 \if_meaning:w #2 \q_mark
345             \exp_after:wN \use_none:n
346         \else:
347             \exp_after:wN \__scontents_zap_space:ww
348         \fi: #2
349     }
350     \cs_new_protected:Npn \__scontents_require_auxii:wwn [#1] #2 [#3]
351     {
352         \tl_set:Nx \l__scontents_temp_tl { \__scontents_zap_space:ww #2 ~ \q_mark }
353         \clist_map_function:NN \l__scontents_temp_tl \__scontents_require_auxiii:n
354     }
355     \cs_new_protected:Npn \__scontents_require_auxiii:n #1
356     {
357         \str_if_eq:eeF {expl3} {#1}
358             { \msg_error:nnn { scontents } { invalid-package } {#1} }
359         }
360         \msg_new:nnn { scontents } { invalid-package }
361         {
362             Package~'#1'~invalid~in~scontents.~This~is~an~error~in~scontents.
363         }
364         \__scontents_compat_redefine:cpx { @ifpackagelater } #1
365         {
366             \exp_args:Nc \__scontents_package_later_aux:Nn { ver@#1.sty }
367         }
368         \cs_new_protected:Npn \__scontents_package_later_aux:Nn #1 #2
369         {
370             \int_compare:nNnTF
371                 { \exp_after:wN \__scontents_parse_version:w #1 //00 \q_mark } <
372                 { \exp_after:wN \__scontents_parse_version:w #2 //00 \q_mark }
373             }
374             \cs_new:Npn \__scontents_parse_version:w #1 { \__scontents_parse_version_auxi:w 0#1 }
375             \cs_new:Npn \__scontents_parse_version_auxi:w #1/#2/#3#4#5 \q_mark
376             {
377                 \__scontents_parse_version_auxii:w #1#2#3#4 \q_mark
378                 \cs_new:Npn \__scontents_parse_version_auxii:w #1#2#3#4#5 \q_mark
379                 {
380                     \tl_if_blank:nF {#2} {#1} #2 #3 #4
381                 }
382                 \__scontents_compat_redefine:Npn \ProvidesExplPackage #1 #2 #3 #4
383                 {
384                     \__scontents_provides_aux:nn {#1} { #2 \tl_if_empty:nF {#3} {#3~} #4 }
385                 }
386             }
387             \cs_new_protected:Npn \__scontents_provides_aux:nn #1 #2
388             {
389                 \tl_gset:cx { ver@#1.sty } {#2}
390                 \iow_log:n { Package~#1:~#2 }
391                 \ExplSyntaxOn
392             }
393             \__scontents_compat_redefine:Npn \DeclareOption
394                 { \__scontents_stararg:nn { \use_none:n } { \use_none:nn } }
395             \__scontents_compat_redefine:Npn \ProcessOptions
396                 { \__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

`\ExplSyntaxOn`. Next, a `\@ifpackagelater` test is done: we do that test too to ensure that `xparse` is compatible with the current running version of `\ExplSyntaxOn`. 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- \LaTeX formats, they are ignored. Okay, so load `xparse`:

```

380 \int_set:Nn \l__scontents_tmpa_int { \char_value_catcode:n { `@ } }
381 \char_set_catcode_letter:N @
382 \exp_after:wN
383 \ExplSyntaxOff
384 \file_input:n { xparse.sty }
385 \ExplSyntaxOn
386 \char_set_catcode:nn { `@ } { \l__scontents_tmpa_int }
387 \__scontents_compat_restore:
388 {/loader&!latex}
389 {*core}

```

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

11.7 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.

```

390 \cs_new_protected:Npn \__scontents_append_contents:nn #1#2
391 {
392     \tl_if_blank:nT {#1}
393     { \msg_error:nn { scontents } { empty-store-cmd } }
394     \seq_if_exist:cF { g__scontents_name_#1_seq }
395     { \seq_new:c { g__scontents_name_#1_seq } }
396     \seq_gput_right:cn { g__scontents_name_#1_seq } {#2}
397 }
398 \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.

```

399 \cs_new:Npn \__scontents_getfrom_seq:nn #1#2
400 {
401     \seq_if_exist:cTF { g__scontents_name_#2_seq }
402     {
403         \exp_args:Nf \__scontents_getfrom_seq:nnn
404         { \seq_count:c { g__scontents_name_#2_seq } }
405         {#1} {#2}
406     }
407     { \msg_expandable_error:nnn { scontents } { undefined-storage } {#2} }
408 }
409 \cs_new:Npn \__scontents_getfrom_seq:nnn #1#2#3
410 {
411     \bool_lazy_or:nnTF
412     { \int_compare_p:nNn {#2} = { 0 } }
413     { \int_compare_p:nNn { \int_abs:n {#2} } > {#1} }
414     { \msg_expandable_error:nnnn { scontents } { index-out-of-range } {#2} {#3} {#1} }
415     { \seq_item:cn { g__scontents_name_#3_seq } {#2} }
416 }

```

(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.

```

417 \cs_new_protected:Npn \__scontents_lastfrom_seq:n #1
418 {
419     \tl_gset:Nx \g__scontents_temp_tl { \seq_item:cn { g__scontents_name_#1_seq } {-1} }
420     \group_insert_after:N \__scontents_rescan_tokens:V
421     \group_insert_after:N \g__scontents_temp_tl
422     \group_insert_after:N \tl_gcarray:N

```

```

423     \group_insert_after:N \g__scontents_temp_tl
424   }
425 \cs_generate_variant:Nn \__scontents_lastfrom_seq:n { V }

(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.

```

426 \cs_new_protected:Npn \__scontents_store_to_seq:NN #1#2
427 {
428   \tl_log:N #1
429   \__scontents_append_contents:Vx #2 { \exp_not:V #1 }
430 }
```

(End definition for __scontents_store_to_seq:NN.)

11.8 Construction of environment scontents

In order to be able to define environments that behave similarly to scontents, we define a generic environment and make all other environment as wappers around that one.

11.8.1 The command \newenvsc

\newenvsc The \newenvsc command defines two functions __scontents_#1_env_begin: and __scontents_#1_env_end:, which set the current environment's default properties and then call the generic __scontents_env_generic_begin: and __scontents_env_generic_end:.

```

431 \tl_new:N \l__scontents_env_name_tl
432 \cs_new_protected:Npn \__scontents_scontents_setenv:nn #1 #2
433 {
434   \cs_new_protected:cpx { __scontents_#1_env_begin: }
435   {
436     \tl_set:Nn \l__scontents_env_name_tl {#1}
437     \keys_set:nn { scontents } {#2}
438     \__scontents_setup_verb_processor:
439     \__scontents_env_generic_begin:
440   }
441   \cs_new_protected:cpx { __scontents_#1_env_end: }
442   {
443     \__scontents_env_generic_end:
444     \exp_args:Nooo % http://ooooooooooooooo.com :) jeje
445     \__scontents_env_define:nnn { \tl_to_str:n {#1} }
446     {
447       \cs:w __scontents_#1_env_begin: \cs_end:
448       \cs:w __scontents_#1_env_end: \cs_end:
449     }
450   }
451   </core>
452   <*loader>
453   \NewDocumentCommand { \newenvsc } { m O{} }
454   {
455     <latex | plain> \cs_if_exist:cTF { #1 }
456     <context> \cs_if_exist:cTF { start #1 }
457     {
458       \msg_error:nnn { scontents } { env-already-defined } {#1}
459       \__scontents_scontents_setenv:nn {#1} {#2}
460     }
461   }
462   \cs_new_protected:Npn \__scontents_env_define:nnn #1 #2 #3
463   {
464     <latex | plain> \NewDocumentEnvironment {#1} { }
465     <context> \cs_new_protected:cpx { start #1 }
466     {
467       <!ilatex> \group_begin:
468       #2
469     }
470     <context> \cs_new_protected:cpx { stop #1 }
471     {
472       <!ilatex> \group_end:
473     }
474   }
475   </loader>
476   <*core>
```

(End definition for \newenvsc, \l__scontents_env_name_tl, and __scontents_scontents_setenv:nn. This function is documented on page 4.)

11.8.2 Generic definition of the environment

`_scontents_env_generic_begin:` Now we define the generic environment `_scontents_env_generic_begin`:

```

473 \cs_new_protected:Npn \_scontents_env_generic_begin:
474 {
475     \char_set_catcode_active:N \^^M
476     \_scontents_start_environment:w
477 }
478 \cs_new_protected:Npn \_scontents_env_generic_end:
479 {
480     \_scontents_stop_environment:
481     \_scontents_finish_storing:NNN \l__scontents_macro_tmp_tl
482     \l__scontents_name_seq_env_tl \l__scontents_print_env_bool
483 }
```

(End definition for `_scontents_env_generic_begin`.)

11.8.3 Definition of the environment scontents

`scontents` Now defining the `scontents` environment should be easy:

```

\contents
\contents
\endscontents
\startscontents
\stopscontents
```

(End definition for `scontents` and others. These functions are documented on page 4.)

11.8.4 key val for environment

Define a `[(key = val)]` for environment `scontents`

`_scontents_grab_optional:n` `_scontents_grab_optional:w` 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.

```

487 </core>
488 <*loader>
489 \NewDocumentCommand \_scontents_grab_optional:w { r[] }
490   { \_scontents_grab_optional:n {#1} }
491 </loader>
492 <*core>
493 \cs_new_protected:Npn \_scontents_grab_optional:n #1
494   {
495     \tl_if_novalue:nF {#1}
496     {
497       \tl_set:Nn \l__scontents_temp_tl {#1}
498       \_scontents_normalise_line_ends:N \l__scontents_temp_tl
499       \keys_set:nV { scontents / scontents } \l__scontents_temp_tl
500     }
501     \_scontents_start_after_option:w
502   }
```

(End definition for `_scontents_grab_optional:n` and `_scontents_grab_optional:w`.)

11.8.5 The environment itself

`_scontents_start_environment:w` 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 `TeX` doesn’t try to eliminate space-like characters.

`_scontents_check_line_process:xn` `_scontents_stop_environment:` 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_process:xn` checks that everything past `\begin{scontents}` is empty and then process the environment. `_scontents_check_line_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. This part of the code is inspired and adapted from the code of the package `xsimverb` by Clemens Niederberger.

```

503 \group_begin:
504   \char_set_catcode_active:N \^I
505   \char_set_catcode_active:N \^L
506   \char_set_catcode_active:N \^M
507   \cs_new_protected:Npn \__scontents_normalise_line_ends:N #1
508     { \tl_replace_all:Nnn #1 { ^M } { ~ } }
509   \cs_new_protected:Npn \__scontents_start_environment:w #1 ^M
510   {
511     \tl_if_head_is:N_type:nTF {#1}
512     {
513       \str_if_eq:eeTF { \tl_head:n {#1} } { [ }
514         { \__scontents_grab_optional:w #1 ^M }
515         { \__scontents_check_line_process:xn { } {#1} }
516       ]
517       { \__scontents_check_line_process:xn { } {#1} }
518     }
519   \cs_new_protected:Npn \__scontents_start_after_option:w #1 ^M
520     { \__scontents_check_line_process:xn { [...] } {#1} }
521 \cs_new_protected:Npn \__scontents_check_line_process:xn #1 #2
522   {
523     \tl_if_blank:nF {#2}
524     {
525       \msg_error:nnnx { scontents } { junk-after-begin }
526         { after~\c_backsplash_str begin { \l__scontents_env_name_tl } #1 } {#2}
527     }
528     \__scontents_make_control_chars_active:
529     \__scontents_file_tl_write_start:V \l__scontents_fname_out_tl
530   }
531 \cs_new_protected:Npn \__scontents_stop_environment:
532   {
533     \__scontents_file_write_stop:N \l__scontents_macro_tmp_tl
534     \bool_lazy_and:nnT
535       { \l__scontents_storing_bool }
536       { \tl_if_empty_p:N \l__scontents_macro_tmp_tl }
537     {
538       \msg_warning:nnx { scontents } { empty-environment }
539         { \l__scontents_env_name_tl }
540     }
541   }

```

(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_verb_processor_iterate:nnn`
`__scontents_setup_verb_processor:`
`__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 `\langle output file \rangle`, 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 `\langle output file \rangle` 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 TeX error is raised otherwise).

```

542 \cs_new_protected:Npn \__scontents_file_tl_write_start:n #1
543   {
544     \group_begin:
545       \__scontents_file_if_writable:nTF {#1}
546       {
547         \bool_set_true:N \l__scontents_writable_bool
548         \iow_open:Nn \l__scontents_file_iow {#1}
549       }
550       { \bool_set_false:N \l__scontents_writable_bool }
551       \tl_clear:N \l__scontents_file_tl
552       \seq_map_function:NN \l__char_special_seq \char_set_catcode_other:N
553       \int_step_function:nnN { 128 } { 1 } { 255 } \char_set_catcode_letter:n
554       \cs_set_protected:Npx \__scontents_ret:w ##1 ^M
555       {
556         \exp_not:N \__scontents_verb_processor_iterate:w
557         ##1 \c__scontents_end_env_tl
558         \c__scontents_end_env_tl

```

```

559           \exp_not:N \q_scontents_stop
560       }
561   \__scontents_make_control_chars_active:
562   \__scontents_ret:w
563 }
564 \cs_new:Npn \__scontents_setup_verb_processor:
565 {
566   \use:x
567   {
568     \cs_set:Npn \exp_not:N \__scontents_verb_processor_iterate:w
569     #####1 \c_scontents_end_env_tl
570     #####2 \c_scontents_end_env_tl
571     #####3 \exp_not:N \q_scontents_stop
572   } { \__scontents_verb_processor_iterate:nnn {##1} {##2} {##3} }
573 }
574 \cs_new:Npn \__scontents_verb_processor_iterate:nnn #1 #2 #3
575 {
576   \tl_if_blank:nTF {#3}
577   {
578     \__scontents_analyse_nesting:n {#1}
579     \__scontents_verb_processor_output:n {#1}
580   }
581   {
582     \__scontents_if_nested:TF
583     {
584       \__scontents_nesting_decr:
585       \__scontents_verb_processor_output:x
586       { \exp_not:n {#1} \c_scontents_end_env_tl \exp_not:n {#2} }
587     }
588   }
589   \tl_if_blank:nF {#1}
590   { \__scontents_verb_processor_output:n {#1} }
591 \cs_set_protected:Npx \__scontents_ret:w
592 {
593   \__scontents_env_end_function:
594   \bool_lazy_or:nnF
595   { \tl_if_blank_p:n {#2} }
596   { \str_if_eq_p:ee {#2} { \c_percent_str } }
597   {
598     \str_if_eq:VnF \c_scontents_hidden_space_str {#2}
599     {
600       \msg_warning:nnnn { scontents } { rescanning-text }
601       {#2} { \tl_use:N \l_scontents_env_name_tl }
602     }
603     \__scontents_rescan_tokens:n {#2}
604   }
605 }
606 \char_set_active_eq:NN ^M \__scontents_ret:w
607 }
608 }
609 ^M
610 }
611 \cs_new:Npn \__scontents_env_end_function:
612 {
613   \__scontents_format_case:nnn
614   { \exp_not:N \end { \if_false: } \fi: }
615   { \exp_after:wN \exp_not:N \cs:w end }
616   { \exp_after:wN \exp_not:N \cs:w stop }
617 \tl_use:N \l_scontents_env_name_tl
618 \__scontents_format_case:nnn
619   { \if_false: { \fi: } }
620   { \cs_end: }
621   { \cs_end: }
622 }
623 \cs_new_protected:Npn \__scontents_file_write_stop:N #1
624 {
625   \bool_if:NT \l_scontents_writable_bool
626   { \iow_close:N \l_scontents_file_iow }
627   \use:x
628   {
629     \group_end:

```

```

630          \bool_if:NT \l__scontents_storing_bool
631          {
632              \tl_set:Nn \exp_not:N #1
633              { \exp_args:NV \__scontents_remove_leading_nl:n \l__scontents_file_tl }
634          }
635      }
636  \cs_new:Npn \__scontents_remove_leading_nl:#1
637  {
638      \tl_if_head_is_N_type:nTF {#1}
639      {
640          \exp_args:Nf
641          \__scontents_remove_leading_nl:nn
642          { \tl_head:n {#1} } {#1}
643      }
644      { \exp_not:n {#1} }
645  }
646  \cs_new:Npn \__scontents_remove_leading_nl:nn #1 #2
647  {
648      \token_if_eq_meaning:NNTF ^^J #1
649      { \exp_not:o { \__scontents_remove_leading_nl:w #2 } }
650      { \exp_not:n {#2} }
651  }
652  \cs_new:Npn \__scontents_remove_leading_nl:w ^^J { }
653

```

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

`__scontents_verb_processor_output:n` does the output of each line read, to a token list and to a file, depending on the booleans `\l__scontents_writing_bool` and `\l__scontents_storing_bool`.

```

654  \cs_new_protected:Npn \__scontents_verb_processor_output:n #1
655  {
656      \bool_if:NT \l__scontents_writable_bool
657      { \iow_now:Nn \l__scontents_file_iow {#1} }
658      \bool_if:NT \l__scontents_storing_bool
659      { \tl_put_right:Nn \l__scontents_file_tl { ^^J #1 } }
660  }
661  \group_end:
662  \cs_generate_variant:Nn \__scontents_verb_processor_output:n { x }
663  \cs_generate_variant:Nn \__scontents_file_tl_write_start:n { v }

```

(End definition for `__scontents_verb_processor_output:n`.)

`__scontents_analyse_nesting:n` scans nested `\begin{scontents}` and steps a `\l__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.

Multiple `\end{scontents}` in the same line are not supported...

```

664  \cs_new_protected:Npn \__scontents_analyse_nesting:n #1
665  {
666      \int_zero:N \l__scontents_tmpa_int
667      \__scontents_analyse_nesting_format:n {#1}
668      \int_compare:nNnT { \l__scontents_tmpa_int } > { 1 }
669      { \msg_warning:nn { scontents } { multiple-begin } }
670  }
671  \cs_new_protected:Npn \__scontents_nesting_incr:
672  {
673      \int_incr:N \l__scontents_env_nesting_int
674      \int_incr:N \l__scontents_tmpa_int
675  }
676  \cs_new_protected:Npn \__scontents_nesting_decr:
677  { \int_decr:N \l__scontents_env_nesting_int }
678  \prg_new_protected_conditional:Npnn \__scontents_if_nested: { TF }
679  {
680      \int_compare:nNnTF { \l__scontents_env_nesting_int } > { \c_zero_int }
681      { \prg_return_true: }
682      { \prg_return_false: }
683  }
684  \cs_new:Npn \__scontents_use_none_delimit_by_q_stop:w #1 \q__scontents_stop { }

```

In \LaTeX , environments start with \begin{env} , so checking if a string contains $\begin{scontents}$ is straightforward. Since no } can appear inside «env», then just a macro delimited by } is enough.

```

685 \use:x
686 {
687   \cs_new_protected:Npn \exp_not:N \__scontents_analyse_nesting_latex:w ##1
688   \c_backslash_str begin \c_left_brace_str ##2 \c_right_brace_str
689 } {
690   \__scontents_tl_if_head_is_q_mark:nTF {#2}
691   { \__scontents_use_none_delimit_by_q_stop:w }
692   {
693     \str_if_eq:VnT \l__scontents_env_name_tl {#2}
694     { \__scontents_nesting_incr: }
695     \__scontents_analyse_nesting_latex:w
696   }
697 }
698 \cs_new_protected:Npx \__scontents_analyse_nesting_latex:n #1
699 {
700   \__scontents_analyse_nesting_latex:w #1
701   \c_backslash_str begin
702     \c_left_brace_str \exp_not:N \q__scontents_mark \c_right_brace_str
703   \exp_not:N \q__scontents_stop
704 }
```

In other formats, however, we don't have an "end anchor" to delimit the environment name, so a delimited macro won't help. We have to search for the entire environment command (usually \scontents and \startscontents).

```

705 \cs_new_protected:Npn \__scontents_analyse_nesting_generic_process:nn #1 #2
706 {
707   \tl_if_head_is_N_type:nTF {#2}
708   {
709     \__scontents_tl_if_head_is_q_mark:nF {#2}
710     {
711       \__scontents_nesting_incr:
712       \__scontents_analyse_nesting_generic:w #2 \q__scontents_stop
713     }
714   }
715   { \__scontents_analyse_nesting_generic:w #2 \q__scontents_stop }
716 }
717 \cs_new_protected:Npn \__scontents_analyse_nesting_generic:nn #1 #2
718 {
719   \__scontents_define_generic_nesting_function:n {#1}
720   \use:x
721   {
722     \exp_not:N \__scontents_analyse_nesting_generic:w #2
723     \c_backslash_str #1 \tl_use:N \l__scontents_env_name_tl
724     \exp_not:N \q__scontents_mark \exp_not:N \q__scontents_stop
725   }
726 }
727 \cs_new_protected:Npn \__scontents_define_generic_nesting_function:n #1
728 {
729   \use:x
730   {
731     \cs_set_protected:Npn \exp_not:N \__scontents_analyse_nesting_generic:w #####1
732     \c_backslash_str #1 \tl_use:N \l__scontents_env_name_tl
733     #####2 \exp_not:N \q__scontents_stop
734   } { \__scontents_analyse_nesting_generic_process:nn {##1} {##2} }
735 }
736 
737 
738 \cs_new_eq:NN \__scontents_analyse_nesting_format:n
739 \cs_new_protected:Npn \__scontents_analyse_nesting_latex:n
740 \cs_new_protected:Npn \__scontents_analyse_nesting_format:n
741 \__scontents_nesting_generic:nn { } }
742 \__scontents_nesting_generic:nn { start } }
743 
744 
```

(End definition for $\text{__scontents_analyse_nesting:n}$ and others.)

11.8.6 Recording of the content in the sequence

`_scontents_finish_storing:NNN` Finishes the environment by optionally calling `_scontents_store_to_seq:` and then clearing the temporary token list.

```

745 \cs_new_protected:Npn \_scontents_finish_storing:NNN #1 #2 #3
746 {
747     \bool_if:NT \l__scontents_storing_bool
748     {
749         \bool_if:NF \l__scontents_forced_eol_bool
750         { \tl_put_right:Nx #1 { \c__scontents_hidden_space_str } }
751         \_scontents_store_to_seq:NN #1 #2
752         \bool_if:NT #3 { \_scontents_lastfrom_seq:V #2 }
753     }
754 }
755 
```

(End definition for `_scontents_finish_storing:NNN.`)

`\verbatimsc` In Plain we emulate L^AT_EX's `\verbatim` environment.

`\endverbatimsc`

```

756 {*plain}
757 \bool_new:N \l__scontents_temp_bool
758 \cs_new_protected:Npn \verbatimsc
759 {
760     \group_begin:
761     \_scontents_verbatimsc_aux: \frenchspacing \_scontents_vobeyspaces:
762     \_scontents_xverb:
763 }
764 \cs_new_protected:Npn \endverbatimsc
765 { \group_end: }
766 \cs_new_protected:Npn \_scontents_verbatimsc_aux:
767 {
768     \skip_vertical:N \parskip
769     \int_set:Nn \parindent { 0pt }
770     \skip_set:Nn \parfillskip { 0pt plus 1fil }
771     \int_set:Nn \parskip { 0pt plus 0pt minus 0pt }
772     \tex_par:D
773     \bool_set_false:N \l__scontents_temp_bool
774     \cs_set:Npn \par
775     {
776         \bool_if:NTF \l__scontents_temp_bool
777         {
778             \mode_leave_vertical:
779             \null
780             \tex_par:D
781             \penalty \interlinepenalty
782         }
783     }
784     \bool_set_true:N \l__scontents_temp_bool
785     \mode_if_horizontal:T
786     { \tex_par:D \penalty \interlinepenalty }
787 }
788 }
789 \cs_set_eq:NN \do \char_set_catcode_other:N
790 \dospecials \obeylines
791 \tl_use:N \l__scontents_verb_font_tl
792 \cs_set_eq:NN \do \_scontents_do_noligs:N
793 \_scontents_nolig_list:
794 \tex_everypar:D \exp_after:wN
795 { \tex_the:D \tex_everypar:D \tex_unpenalty:D }
796 }
797 \cs_new_protected:Npn \_scontents_nolig_list:
798 { \do\`{ }\do\<\do\>{\do\, \do\'\do\-\ } }
799 \cs_new_protected:Npn \_scontents_vobeyspaces:
800 { \_scontents_set_active_eq:NN \_scontents_xobeysp: }
801 \cs_new_protected:Npn \_scontents_xobeysp:
802 { \mode_leave_vertical: \nobreak \ }
803 
```

(End definition for `\verbatimsc` and `\endverbatimsc`.)

\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.

```

804 <!*!latex>
805 \cs_if_exist:NF \dospecials
806 {
807     \cs_new:Npn \dospecials
808     { \seq_map_function:NN \l_char_special_seq \do }
809 }
810 </!*!latex>
```

(End definition for \dospecials.)

__scontents_bsphack: 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.

```

811 <!*core>
812 \int_new:N \l__scontents_save_sf_int
813 \dim_new:N \l__scontents_save_skip_dim
814 \cs_new_protected:Npn \_\_scontents_bsphack:
815 {
816     \scan_stop:
817     \mode_if_horizontal:T
818     {
819         \dim_set_eq:NN \l__scontents_save_skip_dim \tex_lastskip:D
820         \int_set_eq:NN \l__scontents_save_sf_int \tex_spacefactor:D
821     }
822 }
823 \cs_new_protected:Npn \_\_scontents_esphack:
824 {
825     \scan_stop:
826     \mode_if_horizontal:T
827     {
828         \int_set_eq:NN \tex_spacefactor:D \l__scontents_save_sf_int
829         \dim_compare:nNnT { \l__scontents_save_skip_dim } > { \c_zero_dim }
830         {
831             \dim_compare:nNnT { \tex_lastskip:D } = { \c_zero_dim }
832             {
833                 \nobreak
834                 \skip_horizontal:n { \c_zero_skip }
835             }
836             \tex_ignorespaces:D
837         }
838     }
839 }
840 </core>
841 <!*latex>
842 \cs_gset_eq:NN \_\_scontents_bsphack: \@bsphack
843 \cs_gset_eq:NN \_\_scontents_esphack: \@esphack
844 </!*latex>
```

(End definition for __scontents_bsphack: and __scontents_esphack:.)

11.9 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.

```

845 <!*loader>
846 \NewDocumentCommand { \Scontents }{ !s !o{} }
847 { \_\_scontents_Scontents_internal:nn {#1} {#2} }
848 </!*loader>
849 <!*core>
850 \cs_new_protected:Npn \_\_scontents_Scontents_internal:nn #1 #2
851 {
852     \_\_scontents_bsphack:
853     \group_begin:
```

```

854     \tl_if_novalue:nF {#2}
855     { \keys_set:nn { scontents / Scontents } {#2} }
856     \char_set_catcode_active:n { 9 }
857     \bool_if:NTF #1
858     { \__scontents_verb_arg:w }
859     { \__scontents_norm_arg:n }
860   }
861 \cs_new_protected:Npn \__scontents_norm_arg:n #1
862   {
863     \tl_set:Nn \l__scontents_temp_tl {#1}
864     \__scontents_Scontents_finish:
865   }
866 (/core)
867 (*loader)
868 \NewDocumentCommand { \__scontents_verb_arg:w } { +v }
869   { \__scontents_verb_arg_internal:n {#1} }
870 (/loader)
871 (*core)
872 \cs_new_protected:Npn \__scontents_verb_arg_internal:n #1
873   {
874     \tl_set:Nn \l__scontents_temp_tl {#1}
875     \tl_replace_all:Nnx \l__scontents_temp_tl { \iow_char:N \^M } { \iow_char:N \^J }
876     \__scontents_Scontents_finish:
877   }
878 \cs_new_protected:Npn \__scontents_Scontents_finish:
879   {
880     \__scontents_file_write_cmd:VV \l__scontents_fname_out_tl \l__scontents_temp_tl
881     \__scontents_finish_storing:NNN \l__scontents_temp_tl
882       \l__scontents_name_seq_cmd_tl \l__scontents_print_cmd_bool
883     \use:x
884     {
885       \group_end:
886       \bool_if:NTF \l__scontents_print_cmd_bool { \__scontents_esphack: }
887     }
888   }
889 \cs_new_protected:Npn \__scontents_file_write_cmd:nn #1#
890   {
891     \__scontents_file_if_writable:nT {#1}
892     {
893       \iow_open:Nn \l__scontents_file_iow {#1}
894       \iow_now:Nn \l__scontents_file_iow {#2}
895       \iow_close:N \l__scontents_file_iow
896     }
897   }
898 \prg_new_protected_conditional:Npnn \__scontents_file_if_writable:n #1 { T, F, TF }
899   {
900     \bool_if:NTF \l__scontents_writing_bool
901     {
902       \file_if_exist:nTF {#1}
903       {
904         \bool_if:NTF \l__scontents_overwrite_bool
905         {
906           \msg_warning:nnx { scontents } { overwrite-file } {#1}
907           \prg_return_true:
908         }
909         {
910           \msg_warning:nnx { scontents } { not-writing } {#1}
911           \prg_return_false:
912         }
913       }
914     }
915     \msg_warning:nnx { scontents } { writing-file } {#1}
916     \prg_return_true:
917   }
918 }
919 { \prg_return_false: }
920 }
921 \cs_generate_variant:Nn \__scontents_file_write_cmd:nn { VV }

```

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

11.10 The command \getstored

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

```

922  //</core>
923  /*<loader>
924  \NewDocumentCommand { \getstored } { O{-1} m }
925  { \__scontents_getstored_internal:nn {#1} {#2} }
926  /*</loader>
927  /*<core>
928  \cs_new_protected:Npn \__scontents_getstored_internal:nn #1 #2
929  {
930    \group_begin:
931      \int_set:Nn \tex_newlinechar:D { `^\^J }
932      \__scontents_rescan_tokens:x
933      {
934        \endgroup % This assumes \catcode`\\=0... Things might go off otherwise.
935        \__scontents_getfrom_seq:nn {#1} {#2}
936      }
937  }
```

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

11.11 The command \foreachsc

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

```

938  //</core>
939  /*<loader>
940  \NewDocumentCommand { \foreachsc } { o m }
941  { \__scontents_FOREACHSC_internal:nn {#1} {#2} }
942  /*</loader>
943  /*<core>
944  \cs_new_protected:Npn \__scontents_FOREACHSC_internal:nn #1 #2
945  {
946    \group_begin:
947      \tl_if_novalue:nF {#1} { \keys_set:nn { scontents / foreachsc } {#1} }
948      \tl_set:Nn \l__scontents_FOREACH_name_seq_tl {#2}
949      \seq_clear:N \l__scontents_FOREACH_print_seq
950      \bool_if:NF \l__scontents_FOREACH_stop_bool
951      {
952        \int_set:Nn \l__scontents_FOREACH_stop_int
953        { \seq_count:c { g__scontents_name_#2_seq } }
954      }
955      \int_step_function:nnN
956      { \l__scontents_FOREACH_start_int }
957      { \l__scontents_FOREACH_step_int }
958      { \l__scontents_FOREACH_stop_int }
959      \__scontents_FOREACH_add_body:n
960      \tl_gset:Nx \g__scontents_temp_tl
961      {
962        \exp_args:NNV \seq_use:Nn
963        \l__scontents_FOREACH_print_seq \l__scontents_FOREACH_sep_tl
964      }
965      \group_end:
966      \exp_after:wN \tl_gclear:N
967      \exp_after:wN \g__scontents_temp_tl
968      \g__scontents_temp_tl
969  }
970  \cs_new_protected:Npn \__scontents_FOREACH_add_body:n #1
971  {
972    \seq_put_right:Nx \l__scontents_FOREACH_print_seq
973    {
974      \bool_if:NT \l__scontents_FOREACH_before_bool
975      { \exp_not:V \l__scontents_FOREACH_before_tl }
976      \bool_if:NTF \l__scontents_FOREACH_wrapper_bool
977      { \__scontents_FOREACH_wrapper:n }
978      { \use:n }
979      { \getstored [#1] { \tl_use:N \l__scontents_FOREACH_name_seq_tl } }
980      \bool_if:NT \l__scontents_FOREACH_after_bool
981      { \exp_not:V \l__scontents_FOREACH_after_tl }
982    }
983  }
```

```
983 }
```

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

11.12 The command `\typestored`

```
\typestored
\__scontents_verb_print:N
\__scontents_xverb:w
\verbatimsc
 984 〈/core〉
 985 〈*loader〉
 986 〈NewDocumentCommand { \typestored } { o m }
 987  { \__scontents_typestored_internal:nn {#1} {#2} }
 988 〈/loader〉
 989 〈*core〉
 990 \cs_new_protected:Npn \__scontents_typestored_internal:nn #1 #2
 991  {
 992   \group_begin:
 993   \int_set:Nn \l__scontents_seq_item_int { 1 }
 994   \tl_if_no_value:nF {#1} { \keys_set:nn { scontents / typemeaning } {#1} }
 995   \tl_set:Nx \l__scontents_temp_tl
 996   { \exp_args:NV \__scontents_getfrom_seq:nn \l__scontents_seq_item_int {#2} }
 997   \tl_remove_once:NV \l__scontents_temp_tl \c__scontents_hidden_space_str
 998   \tl_log:N \l__scontents_temp_tl
 999   \tl_if_empty:NF \l__scontents_temp_tl
1000   { \__scontents_verb_print:N \l__scontents_temp_tl }
1001   \group_end:
1002 }
```

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`.

```
1003 \group_begin:
1004   \char_set_catcode_active:N \^M
1005   \cs_new_protected:Npn \__scontents_verb_print:N #1
1006   {
1007     \tl_if_blank:VT #1
1008     { \msg_error:nnn { scontents } { empty-variable } {#1} }
1009     \cs_set_eq:NN \__scontents_verb_print_EOL: ^M
1010     \cs_set_eq:NN ^M \scan_stop:
1011     \cs_set_eq:cN { do@noligs } \__scontents_do_noligs:N
1012     \int_set:Nn \tex_newlinechar:D { `\\^J }
1013     \__scontents_rescan_tokens:x
1014     {
1015       \__scontents_format_case:nnn
1016       { \exp_not:N \begin{verbatimsc} } % LaTeX
1017       { \verbatimsc } % Plain/Generic
1018       { \startverbatimsc } % ConTeXt
1019       ^M
1020       \exp_not:V #1 ^M
1021       \g__scontents_end_verbatimsc_tl
1022     }
1023     \cs_set_eq:NN ^M \__scontents_verb_print_EOL:
1024   }
1025 \group_end:
```

Finally, the `\verbatimsc` environment is defined.

```
1026 \cs_new_protected:Npn \__scontents_xverb:
1027   {
1028     \char_set_catcode_active:n { 9 }
1029     \char_set_active_eq:nN { 9 } \__scontents_tabs_to_spaces:
1030     \__scontents_xverb:w
1031   }
1032 〈/core〉
1033 〈*loader〉
1034 〈*!context〉
1035 \use:x
1036   {
1037     \cs_new_protected:Npn \exp_not:N \__scontents_xverb:w
1038     ##1 \g__scontents_end_verbatimsc_tl
```

```

1039 <|latex>      { ##1 \exp_not:N \end{verbatimsc} }
1040 <|plain>       { ##1 \exp_not:N \endverbatimsc }
1041 <|context>     { ##1 \exp_not:N \stopverbatimsc }
1042   }
1043 </|context>
1044 <*|latex>
1045 \NewDocumentEnvironment { verbatimsc } { }
1046   {
1047     \cs_set_eq:cN { @xverbatim } \__scontents_xverb:
1048     \verbatim
1049   }
1050   { }
1051 </|latex>
1052 <|context>\definetyping[verbatimsc]
1053 </|loader>
1054 <*|core>

```

(End definition for `\typestored` and others. These functions are documented on page 6.)

11.13 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.

```

1055 \cs_new:Npn \__scontents_tabs_to_spaces:
1056   { \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 $\text{\LaTeX}_2\text{\varepsilon}$'s `\do@noligs` which makes sure to not consume following space tokens. The $\text{\LaTeX}_2\text{\varepsilon}$ 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.

```

1057 \cs_new:Npn \__scontents_do_noligs:N #1
1058   {
1059     \char_set_catcode_active:N #1
1060     \char_set_active_eq:Nc #1 { __scontents_active_char_ \token_to_str:N #1 : }
1061     \cs_set:cpx { __scontents_active_char_ \token_to_str:N #1 : }
1062     {
1063       \mode_leave_vertical:
1064       \tex_kern:D \c_zero_dim
1065       \char_generate:nn { `#1 } { 12 }
1066     }
1067   }

```

(End definition for `__scontents_do_noligs:N`.)

`__scontents_tl_if_head_is_q_mark:nTF` Tests if the head of the token list is `\q__scontents_mark`.

```

1068 \prg_new_protected_conditional:Npnn \__scontents_tl_if_head_is_q_mark:n #1
1069   { T, F, TF }
1070   {
1071     \if_meaning:w \q__scontents_mark #1 \scan_stop:
1072     \prg_return_true:
1073   \else:
1074     \prg_return_false:
1075   \fi:
1076 }

```

(End definition for `__scontents_tl_if_head_is_q_mark:nTF`.)

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

```

1077 \cs_new_protected:Npn \__scontents_set_active_eq:NN #1
1078   {
1079     \char_set_catcode_active:N #1
1080     \char_set_active_eq:NN #1
1081   }
1082 </|core>
1083 <*|loader>

```

```

1084 \group_begin:
1085 <plain> \char_set_catcode_active:n { `* }
1086   \cs_new_protected:Npn \__scontents_plain_disable_outer_par:
1087   {*plain}
1088   {
1089     \group_begin:
1090       \char_set_lccode:nn { `* } { `^L }
1091       \tex_lowercase:D { \group_end:
1092         \tex_let:D * \scan_stop:
1093       }
1094   }
1095 //plain
1096 <latex | context> { }
1097 \group_end:
1098 //loader
1099 /*core*/
1100 \group_begin:
1101   \char_set_catcode_active:N *
1102   \cs_new_protected:Npn \__scontents_make_control_chars_active:
1103   {
1104     \__scontents_plain_disable_outer_par:
1105     \__scontents_set_active_eq:NN ^I \__scontents_tab:
1106     \__scontents_set_active_eq:NN ^L \__scontents_par:
1107     \__scontents_set_active_eq:NN ^M \__scontents_ret:w
1108   }
1109 \group_end:

```

(End definition for `__scontents_set_active_eq>NN` and `__scontents_make_control_chars_active:`)

11.14 The command `\setupsc`

User command `\setupsc` to setup module.

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

```

1110 //core
1111 /*loader*/
1112 \NewDocumentCommand { \setupsc } { +m }
1113   { \keys_set:nn { scontents } {#1} }
1114 //loader
1115 /*core*/

```

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

11.15 The command `\meaningsc`

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

```

1116 //core
1117 /*loader*/
1118 \NewDocumentCommand { \meaningsc } { o m }
1119   { \__scontents_meaningsc_internal:nn {#1} {#2} }
1120 //loader
1121 /*core*/
1122 \cs_new_protected:Npn \__scontents_meaningsc_internal:nn #1 #2
1123   {
1124     \group_begin:
1125       \int_set:Nn \l__scontents_seq_item_int { 1 }
1126       \tl_if_novalue:nF {#1} { \keys_set:nn { scontents / typemeaning } {#1} }
1127       \__scontents_meaningsc:n {#2}
1128     \group_end:
1129   }
1130 \group_begin:
1131   \char_set_catcode_active:N ^I
1132   \cs_new_protected:Npn \__scontents_meaningsc:n #1
1133   {
1134     \tl_set:Nx \l__scontents_temp_tl
1135     { \exp_args:NV \__scontents_getfrom_seq:nn \l__scontents_seq_item_int {#1} }
1136     \tl_replace_all:Nnx \l__scontents_temp_tl { \iow_char:N ^J } { ~ }
1137     \tl_remove_once:Nv \l__scontents_temp_tl \c__scontents_hidden_space_str
1138     \tl_log:N \l__scontents_temp_tl
1139     \tl_use:N \l__scontents_verb_font_tl

```

```

1140     \tl_replace_all:Nnx \l__scontents_temp_tl { ^^I } { \__scontents_tabs_to_spaces: }
1141     \cs_replacement_spec:N \l__scontents_temp_tl
1142   }
1143 \group_end:

```

(End definition for `\meaningsc`. This function is documented on page 7.)

11.16 The command `\countsc`

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

```

1144  </core>
1145  <*loader>
1146  \NewExpandableDocumentCommand { \countsc } { m }
1147    { \seq_count:c { g__scontents_name_#1_seq } }
1148  </loader>
1149  <*core>

```

(End definition for `\countsc`. This function is documented on page 7.)

11.17 The command `\cleanseqsc`

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

```

1150 </core>
1151 <*loader>
1152 \NewDocumentCommand { \cleanseqsc } { m }
1153   { \seq_clear_new:c { g__scontents_name_#1_seq } }
1154 </loader>
1155 <*core>

```

(End definition for `\cleanseqsc`. This function is documented on page 7.)

11.18 Warning and error messages

Warning and error messages used throughout the package.

```

1156 \msg_new:nnn { scontents } { junk-after-begin }
1157  {
1158    Junk~characters~#1~\msg_line_context: :
1159    \\ \\
1160    #2
1161  }
1162 \msg_new:nnnn { scontents } { env-already-defined }
1163  { Environment~'#1'~already~defined! }
1164  {
1165    You~have~used~\newenvsc
1166    with~an~environment~that~already~has~a~definition. \\ \\
1167    The~existing~definition~of~'#1'~will~not~be~altered.
1168  }
1169 \msg_new:nnn { scontents } { empty-stored-content }
1170  { Empty~value~for~key~'getstored'~\msg_line_context:.. }
1171 \msg_new:nnn { scontents } { empty-variable }
1172  { Variable~'#1'~empty~\msg_line_context:.. }
1173 \msg_new:nnn { scontents } { overwrite-file }
1174  { Overwriting~file~'#1'. }
1175 \msg_new:nnn { scontents } { writing-file }
1176  { Writing~file~'#1'. }
1177 \msg_new:nnn { scontents } { not-writing }
1178  { File~'#1'~already~exists.~Not~writing. }
1179 \msg_new:nnn { scontents } { rescanning-text }
1180  { Rescanning~text~'#1'~after~\c_backslash_str end{#2}~\msg_line_context:.. }
1181 \msg_new:nnn { scontents } { multiple-begin }
1182  { Multiple~\c_backslash_str begin{ \l__scontents_env_name_tl }~\msg_line_context:.. }
1183 \msg_new:nnn { scontents } { undefined-storage }
1184  { Storage~named~'#1'~is~not~defined. }
1185 \msg_new:nnn { scontents } { index-out-of-range }
1186  {
1187    \int_compare:nNnTF {#1} = { 0 }
1188    { Index~of~sequence~cannot~be~zero. }
1189    {
1190      Index~'#1'~out~of~range~for~'#2'..

```

```

1191         \int_compare:nNnTF {#1} > { 0 }
1192             { Max = } { Min = -} #3.
1193         }
1194     }
1195 \msg_new:nnnn { scontents } { env-key-unknown }
1196 {
1197     The~key~'#1'~is~unknown~by~environment~
1198     '\l__scontents_env_name_tl'~and~is~being~ignored.
1199 }
1200 {
1201     The~environment~'\l__scontents_env_name_tl'~does~not~have~a~key~called~'#1'.\\
1202     Check~that~you~have~spelled~the~key~name~correctly.
1203 }
1204 \msg_new:nnnn { scontents } { env-key-value-unknown }
1205 {
1206     The~key~'#1=#2'~is~unknown~by~environment~
1207     '\l__scontents_env_name_tl'~and~is~being~ignored.
1208 }
1209 {
1210     The~environment~'\l__scontents_env_name_tl'~does~not~have~a~key~called~'#1'.\\
1211     Check~that~you~have~spelled~the~key~name~correctly.
1212 }
1213 \msg_new:nnnn { scontents } { cmd-key-unknown }
1214 {
1215     The~key~'#1'~is~unknown~by~'\c_backslash_str Scontents'~and~is~being~ignored.}
1216 {
1217     The~command~'\c_backslash_str Scontents'~does~not~have~a~key~called~'#1'.\\
1218     Check~that~you~have~spelled~the~key~name~correctly.
1219 }
1220 \msg_new:nnnn { scontents } { cmd-key-value-unknown }
1221 {
1222     The~key~'#1=#2'~is~unknown~by~'\c_backslash_str Scontents'~and~is~being~ignored. }
1223 {
1224     The~command~'\c_backslash_str Scontents'~does~not~have~a~key~called~'#1'.\\
1225     Check~that~you~have~spelled~the~key~name~correctly.
1226 }
1227 \msg_new:nnnn { scontents } { for-key-unknown }
1228 {
1229     The~key~'#1'~is~unknown~by~'\c_backslash_str foreachsc'~and~is~being~ignored.}
1230 {
1231     The~command~'\c_backslash_str foreachsc'~does~not~have~a~key~called~'#1'.\\
1232     Check~that~you~have~spelled~the~key~name~correctly.
1233 }
1234 \msg_new:nnnn { scontents } { for-key-value-unknown }
1235 {
1236     The~command~'\c_backslash_str foreachsc'~does~not~have~a~key~called~'#1'.\\
1237     Check~that~you~have~spelled~the~key~name~correctly.
1238 }
1239 \msg_new:nnnn { scontents } { type-key-unknown }
1240 {
1241     This~command~does~not~have~a~key~called~'#1'.\\
1242     This~command~only~accepts~the~key~'width-tab'.
1243 }
1244 \msg_new:nnnn { scontents } { type-key-value-unknown }
1245 {
1246     This~command~does~not~have~a~key~called~'#1'.\\
1247     This~command~only~accepts~the~key~'width-tab'.
1248 }
1249 \msg_new:nnn { scontents } { empty-environment }
1250 {
1251     environment~'#1'~empty~\msg_line_context:.
1252 \msg_new:nnnn { scontents } { verbatim-newline }
1253 {
1254     Verbatim~argument~of~#1~ended~by~end~of~line. }
1255 {
1256     The~verbatim~argument~of~the~#1~cannot~contain~more~than~one~line,~
1257     but~the~end~
1258     of~the~current~line~has~been~reached.~You~may~have~forgotten~the~
1259     closing~delimiter.
1260 \\
1261     LaTeX~will~ignore~'#2'.
1262 }
1263 \msg_new:nnnn { scontents } { verbatim-tokenized }

```

```
1262 { The~verbatim~#1~cannot~be~used~inside~an~argument. }
1263 {
1264     The~#1~takes~a~verbatim~argument.~
1265     It~may~not~appear~within~the~argument~of~another~function.~
1266     It~received~an~illegal~token \tl_if_empty:nF {#3} { ~' #' } .
1267     \\ \\
1268     LaTeX~will~ignore~' #2'.
1269 }
```

11.19 Finish package

Finish package implementation.

```
1270 
```

```
1271 </core>
<plain | context>\ExplSyntaxOff
```

12 Index of Implementation

The italic numbers denote the pages where the corresponding entry is described, the numbers underlined and all others indicate the line on which they are implemented in the package code.

Symbols	
\'	798
*	<u>1085</u> , <u>1090</u> , <u>1101</u>
\,	798
\-	798
\<	798
\>	798
\\"	<u>16</u> , <u>45</u> , <u>934</u> , <u>1159</u> , <u>1166</u> , <u>1201</u> , <u>1210</u> , <u>1216</u> , <u>1222</u> , <u>1228</u> , <u>1234</u> , <u>1240</u> , <u>1246</u> , <u>1258</u> , <u>1267</u>
\`	798
B	
\begingroup	70, <u>75</u>
bool commands:	
\bool_if:NTF	<u>625</u> , <u>630</u> , <u>656</u> , <u>658</u> , <u>747</u> , <u>749</u> , <u>752</u> , <u>776</u> , <u>857</u> , <u>886</u> , <u>900</u> , <u>904</u> , <u>950</u> , <u>974</u> , <u>976</u> , <u>980</u>
\bool_lazy_and:nnTF	<u>534</u>
\bool_lazy_or:nnTF	<u>411</u> , <u>594</u>
\bool_new:N	<u>160</u> , <u>162</u> , <u>164</u> , <u>166</u> , <u>168</u> , <u>170</u> , <u>172</u> , <u>757</u>
\bool_set_false:N	<u>161</u> , <u>165</u> , <u>167</u> , <u>169</u> , <u>171</u> , <u>194</u> , <u>216</u> , <u>550</u> , <u>773</u>
\bool_set_true:N	<u>163</u> , <u>190</u> , <u>195</u> , <u>212</u> , <u>217</u> , <u>234</u> , <u>239</u> , <u>247</u> , <u>255</u> , <u>547</u> , <u>784</u>
C	
\catcode	<u>71</u> , <u>934</u>
char commands:	
\char_generate:nn	<u>36</u> , <u>1065</u>
\char_set_active_eq:NN	<u>606</u> , <u>1060</u> , <u>1080</u>
\char_set_active_eq:nN	<u>1029</u>
\char_set_catcode:nn	<u>386</u>
\char_set_catcode_active:N	<u>475</u> , <u>504</u> , <u>505</u> , <u>506</u> , <u>1004</u> , <u>1059</u> , <u>1079</u> , <u>1101</u> , <u>1131</u>
\char_set_catcode_active:n	<u>856</u> , <u>1028</u> , <u>1085</u>
\char_set_catcode_letter:N	<u>381</u>
\char_set_catcode_letter:n	<u>553</u>
\char_set_catcode_other:N	<u>45</u> , <u>47</u> , <u>48</u> , <u>552</u> , <u>789</u>
\char_set_lccode:nn	<u>1090</u>
\l_char_special_seq	<u>32</u> , <u>32</u> , <u>552</u> , <u>808</u>
\char_value_catcode:n	<u>380</u>
\cleanseqsc	<u>1</u> , <u>7</u> , <u>7</u> , <u>38</u> , <u>1150</u>
clist commands:	
\clist_map_function:NN	<u>346</u>
\countsc	<u>1</u> , <u>7</u> , <u>7</u> , <u>38</u> , <u>1144</u>
cs commands:	
\cs:w	<u>445</u> , <u>446</u> , <u>615</u> , <u>616</u>
\cs_end:	<u>445</u> , <u>446</u> , <u>620</u> , <u>621</u>
\cs_generate_variant:Nn	<u>180</u> , <u>183</u> , <u>184</u> , <u>185</u> , <u>326</u> , <u>398</u> , <u>425</u> , <u>662</u> , <u>663</u> , <u>921</u>
\cs_gset_eq:NN	<u>842</u> , <u>843</u>
\cs_if_exist:NTF	<u>34</u> , <u>452</u> , <u>453</u> , <u>805</u>
\cs_new:Npn	<u>64</u> , <u>182</u> , <u>335</u> , <u>363</u> , <u>364</u> , <u>366</u> , <u>399</u> , <u>409</u> , <u>564</u> , <u>574</u> , <u>611</u> , <u>637</u> , <u>647</u> , <u>653</u> , <u>684</u> , <u>807</u> , <u>1055</u> , <u>1057</u>
\cs_new:Npx	<u>181</u>
\cs_new_eq:NN	<u>738</u>
\cs_new_protected:Npn	<u>179</u> , <u>270</u> , <u>272</u> , <u>278</u> , <u>280</u> , <u>286</u> , <u>292</u> , <u>294</u> , <u>296</u> , <u>313</u> , <u>317</u> , <u>319</u> , <u>321</u> , <u>327</u> , <u>329</u> , <u>333</u> , <u>343</u> , <u>348</u> , <u>357</u> , <u>370</u> , <u>390</u> , <u>417</u> , <u>426</u> , <u>432</u> , <u>434</u> , <u>441</u> , <u>457</u> , <u>460</u> , <u>465</u> , <u>473</u> , <u>478</u> , <u>493</u> , <u>507</u> , <u>509</u> , <u>519</u> , <u>521</u> , <u>531</u> , <u>542</u> , <u>623</u> , <u>654</u> , <u>664</u> , <u>671</u> , <u>676</u> , <u>687</u> , <u>705</u> , <u>717</u> , <u>727</u> , <u>740</u> , <u>745</u> , <u>758</u>
D	
\DeclareOption	<u>376</u>
\def	<u>2</u> , <u>3</u> , <u>5</u> , <u>6</u> , <u>73</u> , <u>76</u> , <u>77</u> , <u>85</u> , <u>98</u>
\definetyping	<u>1052</u>
dim commands:	
\dim_compare:nNnTF	<u>829</u> , <u>831</u>
\dim_new:N	<u>813</u>
\dim_set_eq:NN	<u>819</u>
\c_zero_dim	<u>829</u> , <u>831</u> , <u>1064</u>
\do	<u>789</u> , <u>792</u> , <u>798</u> , <u>808</u>
\dospecials	<u>790</u> , <u>804</u>
E	
\else	<u>95</u> , <u>97</u>
else commands:	
\else:	<u>339</u> , <u>1073</u>
\end	<u>51</u> , <u>614</u> , <u>1039</u>
\endcsname	<u>74</u> , <u>84</u>
\endgroup	<u>73</u> , <u>76</u> , <u>79</u> , <u>92</u> , <u>106</u> , <u>934</u>
\endinput	<u>93</u> , <u>107</u>
\newlinechar	<u>72</u>
\endscontents	<u>4</u> , <u>484</u>
\endverbatimsc	<u>52</u> , <u>756</u> , <u>1040</u>
Environments	
\scontents	<u>25</u> , <u>26</u>
\errhelp	<u>80</u>
\errmessage	<u>81</u>
exp commands:	
\exp_after:wN	<u>338</u> , <u>340</u> , <u>360</u> , <u>361</u> , <u>382</u> , <u>615</u> , <u>616</u> , <u>794</u> , <u>966</u> , <u>967</u>
\exp_args:Nc	<u>356</u>
\exp_args:Nf	<u>403</u> , <u>641</u>
\exp_args:NNV	<u>962</u>
\exp_args:Nooo	<u>443</u>
\exp_args:NV	<u>271</u> , <u>279</u> , <u>293</u> , <u>295</u> , <u>633</u> , <u>996</u> , <u>1135</u>
\exp_not:N	<u>60</u> , <u>556</u> , <u>559</u> , <u>568</u> , <u>571</u> , <u>614</u> , <u>615</u> , <u>616</u> , <u>632</u> , <u>687</u> , <u>702</u> , <u>703</u> , <u>722</u> , <u>724</u> , <u>731</u> , <u>733</u> , <u>1016</u> , <u>1037</u> , <u>1039</u> , <u>1040</u> , <u>1041</u>
\exp_not:n	<u>429</u> , <u>586</u> , <u>645</u> , <u>650</u> , <u>651</u> , <u>975</u> , <u>981</u> , <u>1020</u>
\expandafter	<u>74</u> , <u>84</u>
\ExplSyntaxOff	<u>37</u> , <u>383</u> , <u>1271</u>
\ExplSyntaxOn	<u>26</u> , <u>374</u> , <u>385</u>
F	
\fi	<u>83</u> , <u>109</u> , <u>110</u>

fi commands:	
\fi:	341, 614, 619, 1075
file commands:	
\file_if_exist:nTF	902
\file_input:n	63, 384
\file_input_stop:	38
\foreachsc	1, 5, 5, 21, 22, 34, 938
\frenchspacing	761
G	
\getstored	1, 5, 5, 34, 922, 979
group commands:	
\group_begin: . .	462, 503, 544, 760, 853, 930, 946, 992, 1003, 1084, 1089, 1100, 1124, 1130
\group_end: . .	468, 629, 661, 765, 885, 965, 1001, 1025, 1091, 1097, 1109, 1128, 1143
\group_insert_after:N	420, 421, 422, 423
I	
if commands:	
\if_false:	614, 619
\if_meaning:w	337, 1071
\ifx	74, 84, 96
\input	25
int commands:	
\int_abs:n	413
\int_compare:nNnTF	359, 668, 680, 1187, 1191
\int_compare_p:nNn	412, 413
\int_decr:N	677
\int_incr:N	673, 674
\int_new:N	156, 157, 158, 159, 812
\int_set:Nn	248, 301, 380, 769, 771, 931, 952, 993, 1012, 1125
\int_set_eq:NN	820, 828
\int_step_function:nnnN	553, 955
\int_to_roman:n	22, 298
\int_zero:N	666
\c_zero_int	680
\interlinepenalty	781, 786
iow commands:	
\iow_char:N	16, 875, 1136
\iow_close:N	626, 895
\iow_log:n	29, 373
\iow_new:N	178
\iow_now:Nn	657, 894
\iow_open:Nn	548, 893
K	
keys commands:	
\keys_define:nn	112, 143, 187, 209, 231, 265
\l_keys_key_tl	21, 271, 279, 293, 295
\keys_set:nn	37, 437, 499, 855, 947, 994, 1113, 1126
L	
left commands:	
\c_left_brace_str	59, 688, 702
M	
\meaningsc	1, 7, 7, 19, 21, 22, 37, 1116
mode commands:	
\mode_if_horizontal:TF	785, 817, 826
\mode_leave_vertical:	778, 802, 1063
msg commands:	
\msg_error:nn	393
\msg_error:nnn	275, 283, 289, 306, 351, 454, 1008
\msg_error:nnnn	185, 276, 284, 290, 302, 307, 525
\msg_expandable_error:nnn	407
\msg_expandable_error:nnnn	414
\msg_gset:nnn	32
\msg_line_context:	33, 1158, 1170, 1172, 1180, 1182, 1250
\msg_new:nnn	353, 1156, 1169, 1171, 1173, 1175, 1177, 1179, 1181, 1183, 1185, 1249
\msg_new:nnnn	1162, 1195, 1204, 1213, 1219, 1225, 1231, 1237, 1243, 1251, 1261
\msg_set:nnn	13
\msg_warning:nn	19, 36, 669
\msg_warning:nnn	538, 906, 910, 915
\msg_warning:nnnn	600
N	
\NewDocumentCommand	450, 489, 846, 868, 924, 940, 986, 1112, 1118, 1152
\NewDocumentEnvironment	459, 1045
\newenvsc	1, 4, 4, 25, 25, 431, 485, 1165
\NewExpandableDocumentCommand	1146
\next	73, 76, 85, 98, 111
\nobreak	802, 833
\null	779
O	
\obeylines	790
P	
\PackageError	77, 87, 100
Packages	
l3keys2e	18
scontents	16, 17, 19
xparse	26
\par	774
\parfillskip	770
\parindent	769
\parskip	768, 771
peek commands:	
\peekCharCode_ignorespaces:NTF	328
\peekCharCode_removespaces:NTF	330
\penalty	781, 786
prg commands:	
\prg_generate_conditional_variant:Nnn	186
\prg_new_protected_conditional:Npnn	678, 898, 1068
\prg_replicate:nn	1056
\prg_return_false:	682, 911, 919, 1074
\prg_return_true:	681, 907, 916, 1072
\ProcessKeysOptions	146
\ProcessOptions	378
\ProvidesExplPackage	9, 368
Q	
quark commands:	
\q_mark	337, 345, 360, 361, 364, 365, 366
\quark_new:N	176, 177
quark internal commands:	
\q_scontents_mark	36, 176, 702, 724, 1071
\q_scontents_stop	176, 559, 571, 684, 703, 712, 715, 724, 733
R	
\relax	74, 84
\RequirePackage	8, 331
right commands:	
\c_right_brace_str	61, 688, 702

S

scan commands:

\scan_stop: 816, 825, 1010, 1071, 1092
 \Scontents 1, 4, 4, 20, 22, 32, 845
 \scontents 4, 484
 scontents 3, 484
 scontents internal commands:
 __scontents_analyse_nesting:n 29, 578, 664
 __scontents_analyse_nesting:w 664
 __scontents_analyse_nesting_format:n 667, 738, 740
 __scontents_analyse_nesting_generic:nn . 717, 741, 742
 __scontents_analyse_nesting_generic:w .. 712, 715, 722, 731
 __scontents_analyse_nesting_generic_- process:nn 705, 734
 __scontents_analyse_nesting_latex:n 698, 739
 __scontents_analyse_nesting_latex:w 687, 695, 700
 __scontents_append_contents:nn .. 24, 390, 429
 __scontents_esphack: 811, 852
 __scontents_check_line_process:nn ... 26, 503
 __scontents_compat_redefine:Npn . 313, 326, 331, 355, 368, 376, 378
 __scontents_compat_restore: 319, 387
 __scontents_compat_restore:N 320, 321
 \l__scontents_compat_seq 312, 315, 320
 __scontents_define_generic_nesting_- function:n 719, 727
 __scontents_do_noligs:N ... 36, 792, 1011, 1057
 \c__scontents_end_env_tl 41, 557, 558, 569, 570, 586
 \g__scontents_end_verbatimsc_tl 41, 1021, 1038
 __scontents_env_define:nnn 444, 457
 __scontents_env_end_function: 593, 611
 __scontents_env_generic_begin: 25, 26, 439, 473
 __scontents_env_generic_end: 25, 442, 478
 \l__scontents_env_name_tl .. 60, 431, 526, 539, 601, 617, 693, 723, 732, 1182, 1198, 1201, 1207, 1210
 \l__scontents_env_nesting_int .. 19, 29, 156, 673, 677, 680
 __scontents_esphack: 811, 886
 __scontents_file_if_writable:n 898
 __scontents_file_if_writable:nTF ... 545, 891
 \l__scontents_file_iow 178, 548, 626, 657, 893, 894, 895
 \l__scontents_file_tl 18, 148, 551, 633, 659
 __scontents_file_tl_write_start:n 26, 529, 542, 663
 __scontents_file_write_cmd:nn .. 880, 889, 921
 __scontents_file_write_stop:N ... 27, 533, 542
 __scontents_finish_storing:NNN .. 481, 745, 881
 \l__scontents_fname_out_tl . 18, 148, 191, 196, 213, 218, 529, 880
 \l__scontents_forced_eol_bool 128, 749
 __scontents_FOREACH_ADD_BODY:n 959, 970
 \l__scontents_FOREACH_AFTER_BOOL .. 164, 239, 980
 \l__scontents_FOREACH_AFTER_TL 18, 148, 240, 981
 \l__scontents_FOREACH_BEFORE_BOOL 164, 234, 974
 \l__scontents_FOREACH_BEFORE_TL 18, 148, 235, 975
 \l__scontents_FOREACH_NAME_SEQ_TL . 18, 148, 948, 979

\l__scontents_FOREACH_PRINT_SEQ 19, 173, 949, 963, 972
 \l__scontents_FOREACH_SEP_TL 260, 963
 \l__scontents_FOREACH_START_INT 243, 956
 \l__scontents_FOREACH_STEP_INT 251, 957
 \l__scontents_FOREACH_STOP_BOOL .. 164, 247, 950
 \l__scontents_FOREACH_STOP_INT 19, 156, 248, 952, 958
 __scontents_FOREACH_WRAPPER:n 257, 977
 \l__scontents_FOREACH_WRAPPER_BOOL 164, 255, 976
 __scontents_FOREACHSC_INTERNAL:NN .. 941, 944
 __scontents_FORMAT_CASE:NNN . 64, 613, 618, 1015
 __scontents_getfrom_seq:nn 24, 399, 935, 996, 1135
 __scontents_getfrom_seq:nnn 399
 __scontents_getstored_internal:nn .. 925, 928
 __scontents_grab_optional:n 487
 __scontents_grab_optional:w ... 26, 26, 487, 514
 \c__scontents_hidden_space_str 19, 174, 598, 750, 997, 1137
 __scontents_if_nested: 29
 __scontents_if_nested:TF 582, 664
 __scontents_lastfrom_seq:n 24, 417, 752
 \l__scontents_MACRO_TMP_TL . 18, 148, 481, 533, 536
 __scontents_make_control_chars_active: . 528, 561, 1077
 __scontents_meaningsc:n 1127, 1132
 __scontents_meaningsc_internal:nn .. 1119, 1122
 \l__scontents_name_seq_cmd_tl 117, 882
 \l__scontents_name_seq_env_tl 114, 482
 __scontents_nesting_decr: 29, 584, 664
 __scontents_nesting_incr: 671, 694, 711
 __scontents_nolig_list: 793, 797
 __scontents_norm_arg:n 32, 845
 __scontents_normalise_line_ends:N 26, 498, 507
 __scontents_optarg:nn 327, 332, 334
 \l__scontents_overwrite_bool 131, 904
 __scontents_package_later_aux:Nn .. 356, 357
 __scontents_par: 181, 1106
 __scontents_parse_command_keys:n . 22, 229, 278
 __scontents_parse_command_keys:nn 278
 __scontents_parse_environment_keys:n 21, 207, 270
 __scontents_parse_environment_keys:nn .. 270
 __scontents_parse_foreach_keys:n . 22, 263, 286
 __scontents_parse_foreach_keys:nn 286
 __scontents_parse_type_meaning_key:n 268, 294
 __scontents_parse_type_meaning_key:nn .. 294
 __scontents_parse_typemeaning_key:n 22
 __scontents_parse_version:w 360, 361, 363
 __scontents_parse_version_auxi:w ... 363, 364
 __scontents_parse_version_auxii:w .. 365, 366
 __scontents_plain_disable_outer_par: . 1086, 1104
 \l__scontents_print_cmd_bool .. 24, 125, 882, 886
 \l__scontents_print_env_bool 24, 122, 482
 __scontents_provides_aux:nn 369, 370
 __scontents_remove_leading_nl:n 27, 542
 __scontents_remove_leading_nl:nn ... 642, 647
 __scontents_remove_leading_nl:w 542
 __scontents_require_auxi:wn 332, 333
 __scontents_require_auxii:wnw 334, 343
 __scontents_require_auxiii:n 346, 348
 __scontents_rescan_tokens:n 34, 35, 179, 420, 603, 932, 1013

__scontents_ret:w	27, 554, 562, 591, 606, 1107	\seq_new:N	173, 312, 395
\l__scontents_save_sf_int	812, 820, 828	\seq_put_right:Nn	315, 972
\l__scontents_save_skip_dim	813, 819, 829	\seq_use:Nn	962
__scontents_Scontents_auxi:N	845	\setupsc	2, 37, 1110
__scontents_Scontents_finish:	864, 876, 878	skip commands:	
__scontents_Scontents_internal:nn	845	\skip_horizontal:n	834
__scontents_scontents_setenv:nn	431	\skip_set:Nn	770
\l__scontents_seq_item_int	19, 156, 301, 993, 996, 1125, 1135	\skip_vertical:N	768
__scontents_set_active_eq:NN	800, 1077	\c_zero_skip	834
__scontents_setup_verb_processor:	438, 542	\space	27, 28
__scontents_stararg:nn	329, 377, 379	\startscontents	4, 484
__scontents_start_after_option:w	26, 501, 503	\startverbatimsc	1018
__scontents_start_environment:w	476, 503	\stopscontents	4, 484
__scontents_stop_environment:	480, 503	\stopverbatimsc	53, 1041
__scontents_store_to_seq:	31	str commands:	
__scontents_store_to_seq:NN	25, 426, 751	\c_backslash_str	56, 526, 688, 701, 723, 732, 1180, 1182, 1214, 1216, 1220, 1222, 1226, 1228, 1232, 1234
\l__scontents_storing_bool	19, 29, 160, 194, 216, 535, 630, 658, 747	\c_circumflex_str	175
__scontents_tab:	181, 1105	\c_percent_str	175, 596
\l__scontents_tab_width_int	134, 1056	\str_const:Nn	174
__scontents_tabs_to_spaces:	36, 1029, 1055, 1140	\str_if_eq:nnTF	350, 513, 598, 693
\l__scontents_temp_bool	757, 773, 776, 784	\str_if_eq_p:nn	596
\g__scontents_temp_tl	18, 148, 419, 421, 423, 426, 960, 967, 968		T
\l__scontents_temp_tl	18, 148, 345, 346, 497, 498, 499, 863, 874, 875, 880, 881, 995, 997, 998, 999, 1000, 1134, 1136, 1137, 1138, 1140, 1141	TeX and L ^A T _E X 2 _ε commands:	
__scontents_tl_if_head_is_q_mark:nTF	690, 709, 1068	\@	380, 381, 386
\l__scontents_tmpr_int	156, 380, 386, 666, 668, 674	\@bsphack	842
__scontents_typestored_internal:nn	987, 990	\@esphack	843
__scontents_use_none_delimit_by_q_stop:w	664	\@ifpackageloaded	11
__scontents_verb_arg:w	32, 845	tex commands:	
__scontents_verb_arg_internal:n	845	\tex_everypar:D	794, 795
\l__scontents_verb_font_tl	120, 791, 1139	\tex_ignorespaces:D	836
__scontents_verb_print:N	35, 35, 984	\tex_kern:D	1064
__scontents_verb_print_EOL:	1009, 1023	\tex_lastskip:D	819, 831
__scontents_verb_processor_iterate:nnn	542	\tex_let:D	1092
__scontents_verb_processor_iterate:w	542	\tex_lowercase:D	1091
__scontents_verb_processor_output:n	29, 579, 585, 590, 654	\tex_newlinechar:D	931, 1012
__scontents_verbatimsc_aux:	761, 766	\tex_par:D	772, 780, 786
__scontents_vobeyspaces:	761, 799	\tex_scantokens:D	19, 179
\l__scontents_writable_bool	172, 547, 550, 625, 656	\tex_spacefactor:D	820, 828
\l__scontents_writing_bool	19, 29, 160, 190, 195, 212, 217, 900	\tex_the:D	795
__scontents_xobeysp:	800, 801	\tex_unpenalty:D	795
__scontents_xverb:	762, 1026, 1047		tl commands:
__scontents_xverb:w	984	\c_space_tl	181
__scontents_zap_space:ww	335, 340, 345	\tl_clear:N	551
\Scontents*	22	\tl_const:Nn	54
\ScontentsCoreFileDate	3, 96	\tl_gclear:N	422, 966
\ScontentsFileDialog	2, 10, 27, 96	\tl_gset:Nn	27, 372, 419, 960
\ScontentsFileDescription	6, 10, 28	\tl_gset_rescan:Nnn	42
\ScontentsFileVersion	5, 10, 23, 28	\tl_head:n	513, 643
seq commands:		\tl_if_blank:nTF	274, 282, 288, 300, 305, 367, 392, 523, 576, 589, 1007
\seq_clear:N	949	\tl_if_blank_p:n	595
\seq_clear_new:N	1153	\tl_if_empty:n	186
\seq_count:N	404, 953, 1147	\tl_if_empty:NTF	999
\seq_gput_right:Nn	396	\tl_if_empty:nTF	183, 298, 369, 1266
\seq_if_exist:NTF	394, 401	\tl_if_empty_p:N	536
\seq_item:Nn	415, 419	\tl_if_head_is_N_type:nTF	511, 639, 707
\seq_map_function:NN	320, 552, 808	\tl_if_novalue:nTF	495, 854, 947, 994, 1126
		\tl_log:N	428, 998, 1138
		\tl_new:N	41, 148, 149, 150, 151, 152, 153, 154, 155, 431
		\tl_put_right:Nn	659, 750
		\tl_remove_once:Nn	183, 183, 997, 1137
		\tl_replace_all:Nnn	183, 184, 508, 875, 1136, 1140
		\tl_rescan:nn	19

\tl_set:Nn	191, 196, 213, 218, 235, 240, 345, 436, 497,	use commands:	
	632, 863, 874, 948, 995, 1134	\use:N	29
\tl_to_str:n	\use:n	566, 627, 685, 720, 729, 883, 978, 1035
\tl_use:N	\use_none:n	338, 377
token commands:		\use_none:nn	377
\token_if_eq_meaning:NNTF		
\token_to_str:N		
\tt	V	
\ttfamily	\verbatim	1048
\typestored	\verbatimsc	<u>756</u> , 1017
	1, 6, 6, 19, 21, 22, 35, <u>984</u>	verbatimsc	6, <u>984</u>
U		W	
\unprotect	\writestatus	23