

SCONTENTS

STORES L^AT_EX CONTENTS

V2.8 2026-11-01^{*}

©2019–2026 by Pablo González L[†]

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

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

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*.

Contents

1	Description of the package	1	6	Other commands provided	8
2	Motivation and Acknowledgments	1	6.1	The command <code>\countsc</code>	8
3	License and Requirements	2	6.2	The command <code>\cleanseqsc</code>	9
4	The scontents package	2	7	Programming interface	9
4.1	Installation	2	8	The scontents package in action	9
4.2	Loading and options	2	9	Examples	10
4.3	The TAB character	2	9.1	From answers package	10
4.4	Configuration of the options	3	9.2	From filecontentsdef package	10
4.5	Options Overview	4	9.3	From TeX-SX	11
5	User interface	4	9.4	Customization of <code>verbatimsc</code>	13
5.1	The environment <code>scontents</code>	4	9.5	The command <code>\mergesc</code> in action	15
5.2	The command <code>\newenvsc</code>	5	9.6	Stored contents before storing them	16
5.3	The command <code>\Scontents</code>	5	9.7	The tagged PDF example	17
5.4	The command <code>\getstored</code>	6	10	References	17
5.5	The command <code>\foreachsc</code>	6	11	Change history	18
5.6	The command <code>\tpestored</code>	7	12	Index of Documentation	20
5.7	The command <code>\meaningsc</code>	7	13	Implementation	21
5.8	The command <code>\mergesc</code>	7	14	Index of Implementation	62
5.9	The environment <code>verbatimsc</code>	8			

1 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 ConT_EXt “*buffers*” by making use of *sequences*.

2 Motivation and Acknowledgments

In L^AT_EX there is no direct way to record content for later use, although you can do this using `\macro`, 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 ConT_EXt which allows you to save content in memory, including *verbatim*, to be used later. The `filecontentsdef`[1] package solves this problem and since `expl3`[2] 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 to `filecontentsdef` package. Also a special thanks to Phelype Oleinik who has collaborated and adapted a large part of the code and all L^AT_EX 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 ConT_EXt buffers
3. Storing an array of strings in a command
4. Collecting contents of environment and store them for later retrieval

^{*}This file describes a documentation for v2.8, last revised 2026-11-01.

[†]E-mail: `<pablgonz@educarchile.cl>`.

5. Collect contents of an environment (that contains verbatim content)

- Starting with version 2.3 the `SCONTENTS` package is fully compatible with *tagged* PDF[3] and will have $\text{\LaTeX}$ release 2026-11-01 ($\text{\TeX}$ Live 2026) and update `expl3` release as a minimum requirement.

3 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 (<https://www.latex-project.org/lppl.txt>). The software has the status “maintained”.

The package `SCONTENTS` requires an updated version of $\text{\LaTeX}$ to work (minimum required to $\text{\LaTeX}$ release 2026-11-01 and `expl3` update). This package can be used with plain, context, xelatex, lualatex, pdflatex and the classical workflow `latex»dvips»ps2pdf`.

4 The scontents package

4.1 Installation

The package `SCONTENTS` is present in $\text{\TeX}$ Live and MiK $\text{\TeX}$, use the package manager to install. For manual installation, download `scontents.zip` and unzip it, run `luatex scontents.ins` and move all files to appropriate locations, then run `mktexlsr`. To produce the documentation with source code run `luatex scontents.ins` and `lualatex scontents.dtx` three times.

<code>scontents.tex</code>	»	TDS:tex/generic/scontents/
<code>scontents-code.tex</code>	»	TDS:tex/generic/scontents/
<code>scontents.sty</code>	»	TDS:tex/latex/scontents/
<code>t-scontents.mkiv</code>	»	TDS:tex/context/third/scontents/
<code>scontents.pdf</code>	»	TDS:doc/latex/scontents/
<code>README.md</code>	»	TDS:doc/latex/scontents/
<code>scontents.dtx</code>	»	TDS:source/latex/scontents/
<code>scontents.ins</code>	»	TDS:source/latex/scontents/

4.2 Loading and options

The package is loaded in the usual way:

For $\text{\LaTeX}$ users

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

For plain $\text{\TeX}$ users

```
\input scontents.tex
```

For Con $\text{\TeX}$ t users

```
\usemodule{scontents}
```

- The package options are not available for plain $\text{\TeX}$ and Con $\text{\TeX}$ t and must be passed using `\setupsc` (§4.4). Con $\text{\TeX}$ t users should use `-luatex`, the implementation does not support LuaMeta $\text{\TeX}$.

4.3 The TAB character

Some users use the horizontal TAB ‘`␣`’ character from keyboard to indent the source code of the document and depending on the text editor used, some will use real TAB (*hard tabs*), others with *soft tabs* (`␣` or `␣␣␣`) or both.

At first glance it may seem the same, but the way in which TAB character (*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 $\text{\LaTeX}$ document, the character TAB ‘`␣`’ are treated as explicit spaces (in most contexts) and is the behavior when *stored content*, but when *writing files* these are preserved.

With a $\text{\TeX}$ Live distribution, the TAB character is *printable* for `latex`, `pdflatex` and `lualatex`, but if you use `xelatex` you must add the `-8bit` option on the command line, otherwise you will get $\text{\TeX}$ character TAB ‘`^^I`’ in the *output file*.

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](#).

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 packages `fancyvrb`[4], `fvextra`[5] or `lstlisting` of the package `listings`[6] or when you want to generate a `MakeFile` file.

4.4 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 `<key>` will return an error. In this section are described some of the options, a summary of all options is shown in §4.5.

`\setupsc` `\setupsc{<key> = <val>}`

The command `\setupsc` takes the mandatory argument `{<key> = <val>}` and sets the `<keys>` for the rest of the document, or until the end of the current group, 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`.

`debug-var` = `{<true> | <false>}` default: *true*

Determines whether the `<argument>` passed to commands or the `<body>` passed to provided environments is written to the `.log` file. This `<key>` is only available as a package option or using `\setupsc`.

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

Sets the `<font family>` used to display the `<stored content>` for `\typestored`, `\mergesc` 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`.

`remember` = `{<seq names>}` default: *not used*

Sets the *sequences*, a comma separated list of `{<seq names>}`, whose `<stored contents>` can be used before they are stored, as `\label` and `\ref` do with the references. At the end of the document the `<stored contents>` of these *sequences* are written to the file `\jobname.sct` and in the next compilation `\getstored`, `\foreachsc`, `\typestored`, `\meaningsc`, `\mergesc` and `\countsc` take the `<stored contents>` from it, so they are available before they are stored. As with the references, two compilations are needed: the first one prints nothing where a `<stored content>` is not available yet and a “warning” at the end of the document asks to compile again. If a `<stored content>` is still not available, for example with an `<index>` greater than the number of `<stored contents>`, a “warning” says that there were undefined `<stored contents>`. This `<key>` is only available in $\TeX$, in the preamble, as a package option or using `\setupsc`. For example:

```
\setupsc{remember=objectives} % in the preamble
\section{Objectives}
\getstored[1]{objectives}
\section{Methods}
\Scontents[store-cmd=objectives]{Measure the effect of the method.}
```

A complete example is in §9.6. The file `\jobname.sct` is an auxiliary file, as the `.aux` file, and it can be removed at any time. To remove it with `latexmk -c` add this line to the `latexmkrc` file:

```
push @generated_exts, 'sct';
```

`overwrite` = `{<true> | <false>}` default: *false*

Sets whether the `<files>` generated by `write-out` and `write-env` from the `scontents` environment will be rewritten. This `<key>` is available as a package option, for `\setupsc`, for `\Scontents` and `\Scontents*`, for the base environment `scontents` and all environments created using `\newenvsc`.

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

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

`force-eol` = `{<true> | <false>}` default: *false*

Sets if the *last end of line* for the `<stored content>` is hidden or not. This `<key>` is necessary only if the *last line* is the closing of some environment defined by `fancyvrb`[4] or `fvextra`[5] packages as `\end{Verbatim}` or another environment that does not support a comments ‘%’ after closing `\end{«env»}%`. This `<key>` is available for the base environment `scontents`, the `\Scontents` and `\Scontents*` commands and all environments created using `\newenvsc`.

`width-tab` = `{<integer>}` default: *1*

Sets the equivalence in `<spaces>` for the character TAB ‘`\tab`’ used when displaying `<stored content>` in *verbatim style*. The value must be a `<positive integer>`. This `<key>` is available for `\typestored`, `\mergesc` and `\meaningsc` commands.

4.5 Options Overview

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

5 User interface

The user interface consists in `scontents` environment, `\Scontents` and `\Scontents*` commands to `<stored content>`, `\getstored` command to get the `<stored content>`, `\typestored` and `\mergesc` commands to print *verbatim style* the `<stored content>` along with other utilities described in this documentation.

5.1 The environment scontents

scontents

```
\begin{scontents}[<key=val>]
  <body env>
\end{scontents}
```

The `scontents` environment takes the optional argument `[<key = val>]` and the `{<body env>}`, which it “stores” in a `<sequence>` or “writes” to an `<external file>` if desired, 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 spaces* ‘`␣`’ of the start of the environment.

Comments ‘`%`’ or *any character* after `\begin{scontents}` or `[<key = val>]` 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, the `<stored content>` in the *sequence* `{<inner>}` is *only available* after `<stored content>` in the *sequence* `{<outer>}` one has been retrieved, either by using the key `print-env` or `\getstored` command.

- It is advisable to store the `<content>` in *sequences* with different names, so as not to get lost in the order (position) in which it is stored.

Notes for plain T_EX and ConT_EXt users

In plain T_EX there are no 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

```
\scontents[<key=val>]
  <body env>
\endscontents
```

The *pseudo environment* takes the same optional argument `[<key = val>]` as the `scontents` environment, on the same line as `\scontents`, and the `{<body env>}` up to `\endscontents`.

ConT_EXt users should use `\startscntents` and `\stopscntents`.

```
\startscntents \startscntents[⟨key=val⟩]
\stopscntents  ⟨body env⟩
\stopscntents
```

The *pseudo environment* takes the same optional argument [`⟨key = val⟩`] as the `scontents` environment, on the same line as `\startscntents`, and the `⟨body env⟩` up to `\stopscntents`.

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 `⟨body env⟩` 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 `⟨stored content⟩` is extracted from the *sequence* `⟨seq name⟩` set by the key `store-env` at the time it is executed.

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

Sets the *name* of the `⟨external file⟩` in which the `⟨body env⟩` of the environment will be written. The `⟨file.ext⟩` will be created in the working directory and the `⟨body env⟩` will be stored in the *sequence* set by the key `store-env` at the time it is executed. If `⟨file.ext⟩` does not exist, it will be created or overwritten if the `overwrite` key is used.

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

Sets the *name* of the `⟨external file⟩` in which the `⟨body env⟩` of the environment will be written. The `⟨file.ext⟩` will be created in the working directory and the `⟨body env⟩` will NOT be stored in the *sequence* set by the key `store-env` at the time it is executed. If `⟨file.ext⟩` does not exist, it will be created or overwritten if the `overwrite` key is used.

- 🔗 In the keys `write-env` and `write-out` the character TAB will be written in `⟨file.ext⟩`, relative or absolute paths are not supported. X₃L_AT_EX users using character TAB must add `-8bit` at the command line, otherwise you will get T_EX character TAB ‘`^^I`’ in `⟨file.ext⟩`.

5.2 The command `\newenvsc`

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

The command `\newenvsc` takes the mandatory argument `⟨env name⟩` and the optional argument [`⟨initial keys⟩`], and creates the `⟨new environment⟩` `⟨env name⟩` with 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`. If `⟨env name⟩` already exists, an “error” is returned and its definition is not altered.

In plain T_EX and ConT_EXt the `⟨new environment⟩` is a *pseudo environment* as the one of `scontents` (§5.1): for an environment named `myenv` it is `\myenv... \endmyenv` in plain T_EX and `\startmyenv... \stopmyenv` in ConT_EXt. For example:

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

created the environment `myenvstore` that `⟨stored content⟩` in the *sequence* `⟨myseq⟩` and will NOT display the `⟨stored content⟩` when the environment it is executed.

5.3 The command `\Scontents`

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

The command `\Scontents` takes an optional *starred argument* ‘`*`’, the optional argument [`⟨key = val⟩`] and the mandatory `⟨argument⟩`, between *braces* or, in the starred version, between two `⟨del⟩`, and stores the `⟨argument⟩` in the *sequence* set by the key `store-cmd`.

The command `\Scontents` reads the `⟨argument⟩` in standard mode and “stores” it in the *sequence* set by the key `store-cmd` at the time it is executed. It is not possible to pass *verbatim things*, but it is possible to use the implementation of `\Verb` delimited by *braces* ‘`{ }`’ provided by the `fvextra`[5] package for *verbatim one line* and `\lstinline` from `listings`[6] package, but it is preferable to use the *starred argument* ‘`*`’.

The `\Scontents*` command reads the `⟨argument⟩` under *verbatim catcode* regimen and “stores” it in the *sequence* set by the key `store-cmd` at the time it is executed. If its “first” delimiter is a *brace* ‘`{`’,

it will be assumed that the $\{\langle argument \rangle\}$ is nested within *braces*. Otherwise it will be assumed 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 $\{\langle argument \rangle\}$ is used with *braces* and character TAB typed from the keyboard are converted into spaces.

The *starred argument* `'*` and $[\langle key = val \rangle]$ must NOT be separated by *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 $[\langle key = val \rangle]$.

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

Sets the *name* of the *sequence* in which the $\{\langle argument \rangle\}$ is 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 $\langle stored content \rangle$ is extracted from the *sequence* $\{\langle seq name \rangle\}$ set by the key `store-cmd` at the time it is executed.

Options for writing files

The following $\langle keys \rangle$ and the `force-eol` and `overwrite` keys are available for both versions of the command. The `\Scontents*` command writes the $\{\langle argument \rangle\}$ to $\langle file.ext \rangle$ exactly as it was typed. The `\Scontents` command reads the $\{\langle argument \rangle\}$ in standard mode, so it is written as a list of tokens, in the same way as `\detokenize` does: a space is added after each control word, the character `#` is doubled, the line breaks become spaces and the blank lines become `\par`. To write the $\{\langle argument \rangle\}$ literally, use the *starred argument* `'*`.

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

Sets the *name* of the $\langle external file \rangle$ in which the $\{\langle argument \rangle\}$ will be written. The $\langle file.ext \rangle$ will be created in the working directory and the $\{\langle argument \rangle\}$ will be *stored* in the *sequence* set by the key `store-cmd` at the time it is executed. If $\langle file.ext \rangle$ does not exist, it will be created or overwritten if the `overwrite` key is used.

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

Sets the *name* of the $\langle external file \rangle$ in which the $\{\langle argument \rangle\}$ will be written. The $\langle file.ext \rangle$ will be created in the working directory and the $\{\langle argument \rangle\}$ will NOT be *stored* in any *sequence* set by the key `store-cmd` at the time it is executed. If $\langle file.ext \rangle$ does not exist, it will be created or overwritten if the `overwrite` key is used.

- In the keys `write-cmd` and `write-out` the character TAB will be written in $\langle file.ext \rangle$, relative or absolute paths are not supported. X₃LaTeX users using character TAB must add `-8bit` at the command line, otherwise you will get T_EX character TAB `'^A^I'` in $\langle file.ext \rangle$.

5.4 The command `\getstored`

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

The command `\getstored` takes the optional argument $[\langle index \rangle]$ and the mandatory argument $\{\langle seq name \rangle\}$, and retrieves the $\langle stored content \rangle$ whose *position* in the *sequence* $\{\langle seq name \rangle\}$ is the *integer value* set in $\langle index \rangle$. A *negative* $\langle index \rangle$ counts from the last $\langle stored content \rangle$ and, if the *optional argument* is not passed, the “*last*” $\langle stored content \rangle$ is retrieved.

The command is robust and can be used as an $\{\langle argument \rangle\}$ for another command.

5.5 The command `\foreachsc`

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

The command `\foreachsc` takes the optional argument $[\langle key = val \rangle]$ and the mandatory argument $\{\langle seq name \rangle\}$, and executes the command `\getstored` on each $\langle stored content \rangle$ in the *sequence* $\{\langle seq name \rangle\}$, or `\tystored` or `\meaningsc` with the `tystored` or `meaningsc` key. If the *optional argument* is not passed, `\getstored` is executed on all the $\langle stored content \rangle$ in the *sequence* $\{\langle seq name \rangle\}$.

Options for command

`getstored` $\langle \text{without value} \rangle$ default: *active*

Executes `\getstored` on each $\langle stored content \rangle$, as without $\langle keys \rangle$.

`tystored` $\langle \text{without value} \rangle$ default: *not active*

Executes `\tystored` on each $\langle stored content \rangle$, which is shown in *verbatim style* (§5.6).

<code>meaningsc</code>	<i><without value></i>	default: <i>not active</i>
	Executes <code>\meaningsc</code> on each <i><stored content></i> (§5.7).	
<code>sep = { <code> }</code>		default: <i>empty</i>
	Establishes the <i>separation</i> between each <i><stored content></i> in the <i>sequence</i> <i>{<seq name>}</i> . For example, you can use <code>sep={\[\[10pt]}</code> for vertical separation of <i><stored contents></i> .	
<code>step = { <integer> }</code>		default: <i>1</i>
	Sets the <i>increment</i> (<i><step></i>) applied to the value set by key <code>start</code> for each <i><stored content></i> in the <i>sequence</i> <i>{<seq name>}</i> . The value must be a <i><positive integer></i> .	
<code>start = { <integer> }</code>		default: <i>1</i>
	Set the <i>position</i> of the <i><stored content></i> within the <i>sequence</i> <i>{<seq name>}</i> from which to <i>start</i> executing. The value must be a <i><positive integer></i> .	
<code>stop = { <integer> }</code>		default: <i>total</i>
	Set the <i>position</i> of the <i><stored content></i> within the <i>sequence</i> <i>{<seq name>}</i> at which execution ends. The value must be a <i><positive integer></i> .	
<code>before = { <code> }</code>		default: <i>empty</i>
	Sets the <i>{<code>}</i> that will be executed <i>before</i> each <i><stored content></i> within the <i>sequence</i> <i>{<seq name>}</i> . The <i>{<code>}</i> must be passed <i>between braces</i> ‘{ }’.	
<code>after = { <code> }</code>		default: <i>empty</i>
	Sets the <i>{<code>}</i> that will be executed <i>after</i> each <i><stored content></i> within the <i>sequence</i> <i>{<seq name>}</i> . The <i>{<code>}</i> must be passed <i>between braces</i> ‘{ }’.	
<code>wrapper = { <code> {#1} more code } }</code>		default: <i>empty</i>
	Wraps the <i><stored content></i> within the <i>sequence</i> <i>{<seq name>}</i> referenced by <i>{#1}</i> . The <i>{<code>}</i> must be passed <i>between braces</i> ‘{ }’. For example <code>\foreachsc[wrapper={\makebox[1em][l]{#1}}]{contents}</code> .	

5.6 The command `\typestored`

`\typestored` `\typestored[<index, start-stop, 1-end, keys>]{<seq name>}`

The command `\typestored` takes the optional argument with the *<index>*, the *<start-stop>*, the *<1-end>* and the *<keys>*, and the mandatory argument *{<seq name>}*, and prints the *<stored content>* in the *sequence* *{<seq name>}* in the internal `verbatimsc` environment (§5.9). If the *optional argument* is not passed, the *first <stored content>* is printed.

The *integer value* set in *<index>* is the *position* of the *<stored content>* in the *sequence* *{<seq name>}*, a *negative <index>* counts from the last *<stored content>* as in `\getstored`. The *integer values* set in *<start-stop>* define a *range* of positions, where *end* is the last *<stored content>* and a *<start>* greater than *<stop>* prints the *range* in reverse order, and *<1-end>* prints “all” the *<stored content>*. Several of them can be given separated by commas. The rest of the accepted *<keys>* are `print-cmd` with default value `true`, `write-out`, `width-tab` and `overwrite`.

- In the key `write-out` the character TAB will be written in *<file.ext>*, relative or absolute paths are not supported. X_YLaTeX users using character TAB must add `-8bit` at the command line, otherwise you will get T_EX character TAB ‘`^A^I`’ in *<file.ext>*.

5.7 The command `\meaningsc`

`\meaningsc` `\meaningsc[<index, start-stop, 1-end, keys>]{<seq name>}`

The command `\meaningsc` takes the same optional argument as `\typestored` (§5.6) and the mandatory argument *{<seq name>}*, and prints the *meaning* of the *<stored content>* in the *sequence* *{<seq name>}*, as `\meaning` does, on a single line. The *<index>*, *<start-stop>*, *<1-end>* and *<keys>* have the same behavior as in `\typestored` and, if the *optional argument* is not passed, the *first <stored content>* is printed.

5.8 The command `\mergesc`

`\mergesc` `\mergesc[<typestored | meaningsc | getstored, keys>]{<seq A>[<index>], <seq B>[<start - stop>], <seq C>[<1-end>]}`

The command `\mergesc` takes the optional argument with the *<keys>*, one of them `typestored`, `meaningsc` or `getstored`, and the mandatory argument with a *comma separated list of sequences*, and assembles the *<stored content>* in them, as *{<seq A>[1], <seq B>[2-5] and <seq C>[1-end]}* in the syntax, and then executes `\typestored` (§5.6) if the `typestored` key is active, `\meaningsc` (§5.7) if the `meaningsc` key is active, or rescans it as `\getstored` (§5.4) does if the `getstored` key is active.

The *{<argument>}* taken by this command is a *comma separated list* of the form *{<seq name>}* followed by either *[<index>]*, *[<start-stop>]* or *[<1-end>]*, a *{<seq name>}* alone assembles all its *<stored content>*. The use of the keys `typestored`, `meaningsc` or `getstored` are “mandatory” and disjoint from each other,

the rest of the accepted *keys* are `print-cmd`, `write-out`, `width-tab` and `overwrite`, which are not accepted with the `getstored` key.

The use of the `write-out` key with this command follows the same rules already described, the main advantage is that allows joining *stored content* without rewriting the file over and over again, by design T_EX does not have an *append mode* for writing files, this effectively allows you to write chunks of code and then merge them into a single file.

5.9 The environment `verbatimsc`

`verbatimsc` The environment used by `\typestored` and `\mergesc` to display the *stored content* in *verbatim style*. The environment is compatible with *tagged* PDF and can be customized in the following ways after loading the `SCONTENTS` package:

Using the packages `fvextra`[5] or `fancyvrb`[4]:

```
\ExplSyntaxOn
\cs_undefine:N \verbatimsc
\cs_undefine:N \endverbatimsc
\ExplSyntaxOff
\usepackage{fancyvrb}
\DefineVerbatimEnvironment{verbatimsc}{Verbatim}{numbers=left}
```

Using the package `minted`[7]:

```
\ExplSyntaxOn
\cs_undefine:N \verbatimsc
\cs_undefine:N \endverbatimsc
\ExplSyntaxOff
\usepackage{minted}
\newminted{tex}{linenos}
\newenvironment{verbatimsc}{\VerbatimEnvironment\begin{texcode}}{\end{texcode}}
```

Using the package `listings`[6]:

```
\ExplSyntaxOn
\cs_undefine:N \verbatimsc
\cs_undefine:N \endverbatimsc
\ExplSyntaxOff
\usepackage{listings}
\lstnewenvironment{verbatimsc}
{
  \lstset{
    basicstyle=\small\ttfamily,
    columns=fullflexible,
    language=[LaTeX]TeX,
    numbers=left,
    numberstyle=\tiny\color{gray},
    keywordstyle=\color{red}
  }
}{}
\end{verbatimsc}
```

`\startverbatimsc` In plain T_EX the environment is `\verbatimsc... \endverbatimsc` and in ConT_EXt it is `\startverbatimsc... \stopverbatimsc`, defined with `\definetyping`, so there it can be customized with `\setuptyping[verbatimsc][...]`.

🔗 At the moment, the `fvextra`[5] and `fancyvrb`[4] packages partially support *tagged* PDF.

6 Other commands provided

6.1 The command `\countsc`

`\countsc` `\countsc{<seq name>}`

The command `\countsc` takes the mandatory argument `{<seq name>}` and returns the number of *stored content* in the *sequence* `{<seq name>}`. The command is expandable, so it can be used in an *integer expression*, and if the *sequence* does not exist an “error” is returned along with 0.

6.2 The command `\cleanseqsc`

```
\cleanseqsc <\cleanseqsc{<seq name>}
```

The command `\cleanseqsc` takes the mandatory argument `{<seq name>}` and removes all `<stored content>` in the *sequence* `{<seq name>}`. If the *sequence* does not exist, it is created empty.

7 Programming interface

For other packages and for the code written with `expl3`, the `<stored contents>` can be read without printing them with the following functions. They work in $\text{\LaTeX}$, plain $\text{\TeX}$ and $\text{\ConTeXt}$ and also read the *sequences* set by the `remember` key.

The `<stored contents>` are given as they were stored, without the mark that `SCONTENTS` adds to hide the last end of line: the content of an environment is a string whose lines are separated by the end of line character `^^J`, the content of `\Scontents*` is also a string and the content of `\Scontents` keeps its tokens.

```
\scontents_if_exist_p:n <\scontents_if_exist_p:n {<seq name>}
\scontents_if_exist:nTF <\scontents_if_exist:nTF {<seq name>} {<true code>} {<false code>}
```

Tests if the *sequence* `{<seq name>}` exists.

```
\scontents_count:n <\scontents_count:n {<seq name>}
```

Leaves the number of `<stored contents>` in the *sequence* `{<seq name>}` in the input stream, or `0` if it does not exist, without an “error”. It can be used in an *integer expression*.

```
\scontents_get:nnN <\scontents_get:nnN {<seq name>} {<index>} {<tl var>}
```

Sets the `<tl var>`, locally, to the `<stored content>` at the position `{<index>}` of the *sequence* `{<seq name>}`. The `{<index>}` is an *integer expression*, a negative one counts from the end. If the *sequence* does not exist or the `{<index>}` is out of range, the `<tl var>` is empty and an “error” is returned.

```
\scontents_map_inline:nn <\scontents_map_inline:nn {<seq name>} {<inline function>}
```

Applies the `{<inline function>}` to each `<stored content>` of the *sequence* `{<seq name>}`, in order, the `<stored content>` is `#1`. The `{<inline function>}` can use the other functions and `\scontents_map_inline:nn` again. If the *sequence* does not exist an “error” is returned. For example, this shows in the `.log` file each `<stored content>` of the *sequence* data converted to uppercase:

```
\ExplSyntaxOn
\scontents_map_inline:nn { data }
{ \iow_log:e { \text_uppercase:n {#1} } }
\ExplSyntaxOff
```

8 The `SCONTENTS` package in action

Remember the abstract on the first page? This is it:

Abstract

This package allows to store $\text{\LaTeX}$ 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 $\text{\ConTeXt}$ “*buffers*” by making use of *sequences*.

I’ve only written:

```
\begin{abstract}
This package allows to store \hologo{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*[no-index]{scontents}` package allows to `\mymeta{store contents}` in `\mymeta{sequences}` or `\mymeta{external files}`. In some ways it is similar to the `\mypkg{filecontentsdef}` package, with the difference in which the `\mymeta{content}` is stored. The idea behind this package is to get an approach to `\hologo{ConTeXt}` `\emph{\enquote{buffers}}` by making use of `\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 \hologo{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*[no-index]{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} \emph{\enquote{buffers}} by
6 making use of \mymeta{sequences}.
7 \end{scontents}

```

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

9 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:

```
$ pdftdetach -saveall scontents.pdf
```

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

9.1 From answers package

Example 1

Adaptation of example 1 of the package **answers**[8]. Source file .

```

1 % arara: lualatex: { branch: developer }
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}

```

9.2 From filecontentsdef package

Example 2

Adaptation of example from package **filecontentsdef**[1]. Source file .

```

1 % arara: lualatex: { branch: developer }
2 % arara: clean: { extensions: [ aux, log ] }

```

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

```

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
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}

```

9.3 From TeX-SX

Example 3

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

```

1 % arara: lualatex: { branch: developer }
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](#). Source file [\[E\]](#).

```

1 % arara: lualatex: { branch: developer }
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 \(that contains verbatim content\)](#). Source file [\[E\]](#).

```

1 % arara: lualatex: { branch: developer }
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](#). Source file [📄](#).

```

1 % arara: lualatex: { branch: developer }
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}

```

9.4 Customization of verbatimsc

Example 7

Customization of `verbatimsc` using the `fvextra`[5] and `tcolorbox`[9] packages. Source file [📄](#).

```

1 % arara: lualatex: { branch: developer }
2 % arara: clean: { extensions: [ aux, log ] }
3 \documentclass{article}
4 \usepackage{scontents}
5 \ExplSyntaxOn
6 \cs_undefine:N \verbatimsc
7 \cs_undefine:N \endverbatimsc
8 \ExplSyntaxOff
9 \usepackage{fvextra}
10 \usepackage{xcolor}
11 \definecolor{mygray}{gray}{0.9}
12 \usepackage{tcolorbox}
13 \newenvironment{verbatimsc}%
14 {\VerbatimEnvironment
15 \begin{tcolorbox}[colback=mygray, boxsep=0pt, arc=0pt, boxrule=0pt]
16 \begin{Verbatim}[fontsize=\scriptsize, breaklines, breakafter=*, breaksymbolsep=0.5em,
17 breakaftersymbolpre={\,\tiny\ensuremath{\lfloor}}]}%
18 {\end{Verbatim}%
19 \end{tcolorbox}}
20 \setlength{\parindent}{0pt}
21 \pagestyle{empty}
22 \begin{document}
23 \section[Test \texttt{\textbackslash begin\{scontents\}} with \texttt{fancyvrb}]
24 Test \verb+{scontents}+ \par
25
26 \begin{scontents}
27 Using \verb+scontents+ env no \verb+[key=val]+, save in seq \verb+contents+
28 with index 1.
29
30 Prove new \Verb*{ fancyvrb with braces } and environment \verb+Verbatim*+
31 \begin{verbatim}
32 verbatim environment
33 \end{verbatim}
34 \end{scontents}
35
36 \section[Test \texttt{\textbackslash Scontents} with \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\textbackslashash typestored}}
48 \typestored[1]{contents}
49 \typestored[2]{contents}
50 \end{document}

```

Example 8

Customization of `verbatimsc` using the `listings`[6] package. Source file .

```

1 % arara: lualatex: { branch: developer }
2 % arara: clean: { extensions: [ aux, log ] }
3 \documentclass{article}
4 \usepackage{scontents}
5 \ExplSyntaxOn
6 \cs_undefine:N \verbatimsc
7 \cs_undefine:N \endverbatimsc
8 \ExplSyntaxOff
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\textbackslashash begin\{scontents\}} with \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 | and environment \verb+Verbatim*+
34 \begin{verbatim}
35   verbatim environment
36 \end{verbatim}
37 \end{scontents}
38
39 \section{Test \texttt{\textbackslash\textbackslashash Scontents*} with \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\textbackslashash getstored}}
45 \getstored{contents}\par
46 \getstored[1]{contents}
47
48 \section{Test \texttt{\textbackslash\textbackslashash typestored}}
49 \typestored[1]{contents}
50 \typestored[2]{contents}
51 \end{document}

```

Example 9

Customization of `verbatimsc` using the `minted`[7] package. Source file .

```

1 % arara: lualatex: { branch: developer }
2 % arara: clean: { extensions: [ aux, log ] }
3 \documentclass{article}
4 \usepackage{scontents}
5 \ExplSyntaxOn
6 \cs_undefine:N \verbatimsc

```

```

7 \cs_undefine:N \endverbatimsc
8 \ExplSyntaxOff
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 begin\{scontents\}} with \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 fvextra with 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} with \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 tpestored}}
43 \tpestored[1]{contents}
44 \tpestored[2]{contents}
45 \end{document}

```

9.5 The command \mergesc in action

The command `\mergesc` in action, adapted from Denis Bitouzé request at <https://github.com/pablgonz/scontents/issues/2>. Source file .

```

1 % arara: lualatex: { branch: developer }
2 % arara: clean: { extensions: [ aux, log ] }
3 \documentclass{article}
4 \usepackage{scontents}
5 % Fix part of a MCE that should go before babel's loading
6 \begin{scontents}[store-env=mce]
7 \documentclass[french]{article}
8 \usepackage[T1]{fontenc}
9 \usepackage[utf8]{inputenc}
10 \usepackage{lmodern}
11 \usepackage[a4paper]{geometry}
12 \end{scontents}
13 % Fix part of a MCE that should go after (>=) babel's loading
14 \begin{scontents}[store-env=mce]
15 \usepackage{babel}
16 \begin{document}
17 \end{scontents}
18 % Fix part of a MCE that should go after its body
19 \begin{scontents}[store-env=mce]
20 \end{document}
21 \end{scontents}
22 \begin{document}
23 \section{First answer}
24 % Variable part of a MCE that should added to the fixed preamble, before babel's loading
25 \begin{scontents}[store-env=mce-1]
26 \usepackage{amsmath}
27 \end{scontents}

```

```

28 % Variable part of a MCE being the code snippet
29 \begin{scontents}[store-env=mce-1]
30 \begin{align}
31   0 & \& \neq 1 \\
32   1 & \& \neq 0
33 \end{align}
34 \end{scontents}
35 \begin{description}
36 \item[Preamble's addition]\leavevmode
37   \typestored[1]{mce-1}
38 \item[Code snippet]\leavevmode
39   \typestored[2]{mce-1}
40 \item[MCE]\leavevmode
41   \mergesc[typestored, print-cmd=true]
42   {
43     {mce}[1], {mce-1}[1], {mce}[2], {mce-1}[2], {mce}[3]
44   }
45 \end{description}
46 \section{Second answer}
47 % Variable part of a MCE that should added to the fixed preamble, before babel's loading
48 \begin{scontents}[store-env=mce-2]
49 \usepackage{amsmath}
50 \end{scontents}
51 % Variable part of a MCE being the code snippet
52 \begin{scontents}[store-env=mce-2]
53 \begin{flalign}
54   0 & \& \neq 1 \\
55   1 & \& \neq 0
56 \end{flalign}
57 \end{scontents}
58
59 \begin{description}
60 \item[Preamble's addition]\leavevmode
61   \typestored[1]{mce-2}
62 \item[Code snippet]\leavevmode
63   \typestored[2]{mce-2}
64 \item[MCE]\leavevmode
65   \mergesc[typestored, print-cmd=true, write-out=mce.txt, overwrite=true]
66   {
67     {mce}[1], {mce-2}[1], {mce}[2], {mce-2}[2], {mce}[3]
68   }
69 \end{description}
70 \end{document}

```

9.6 Stored contents before storing them

The `remember` key in action: the *stored contents* of the *sequence* objectives are used in the section “Objectives” before they are stored in the section “Methods”, as requested at <https://github.com/pablonz/scontents/issues/1>. It needs two compilations. Source file .

```

1 % arara: lualatex: { branch: developer }
2 % arara: lualatex: { branch: developer }
3 % arara: clean: { extensions: [ aux, log, sct ] }
4 \documentclass{article}
5 \usepackage[remember=objectives]{scontents}
6 \begin{document}
7 \section{Objectives}
8 The project has \countsc{objectives} objectives, developed in the
9 methods: \foreachsc[sep={; }]{objectives}.
10
11 The first one is the key of the project: \getstored[1]{objectives}.
12
13 \section{Methods}
14 \subsection{First objective}
15 \Scontents[store-cmd=objectives, print-cmd]{measure the effect of the method on small samples}
16
17 The samples are taken from the previous studies.
18
19 \subsection{Second objective}
20 \begin{scontents}[store-env=objectives, print-env]
21 compare the results with the ones of the previous studies
22 \end{scontents}

```

23 `\end{document}`

9.7 The tagged PDF example

This example is just to show the compatibility of `scontents` with *tagged* PDF using `lualatex`. The attached files are only for testing. Source file  and *tagged* PDF .

```

1 % arara: lualatex: { branch: developer }
2 % arara: clean: { extensions: [ aux, log ] }
3 \DocumentMetadata{tagging=on, lang=en-US, pdfversion=2.0, pdfstandard=ua-2}
4 \documentclass{article}
5 \usepackage{scontents,unicode-math,hyperref}
6 \hypersetup{pdftitle = {Test scontents package},}
7 \begin{document}
8   Some
9
10  \begin{scontents}[print-env=true]
11    First code \verb|\foo|
12
13    And more code \verb|\bar|
14  \end{scontents}
15
16  Text
17
18  \begin{scontents}[print-env=true]
19    Second code \verb|\foo|
20
21    And more code \verb|\bar|
22  \end{scontents}
23
24  Text
25
26  \Scontents*{code \verb|\baz|}
27
28  % \typestored
29  \typestored[1]{contents}
30
31  % \mergesc
32  \mergesc[typestored]{ {contents}[1-end] }
33
34  % \getstored
35  \getstored[2]{contents}
36  \end{document}

```

 This example has been checked using `veraPDF` together with `ngpdf`.

10 References

- [1] BURNOL, JEAN FRANÇOIS. “The filecontentsdef package”. Available from CTAN, <https://ctan.org/pkg/filecontentsdef>, 2019.
- [2] The L^AT_EX Project. “The expl3 package”. Available from CTAN, <https://www.ctan.org/pkg/expl3>, 2026.
- [3] FISCHER, ULRIKE. “tagpdf – L^AT_EX kernel code for PDF tagging”. Available from CTAN, <https://www.ctan.org/pkg/tagpdf>, 2026.
- [4] VAN ZANDT, TIMOTHY. “The fancyvrb package - Fancy Verbatims in L^AT_EX”. Available from CTAN, <https://www.ctan.org/pkg/fancyvrb>, 2026.
- [5] POORE, GEOFFREY M. “The fvextra package - Extensions and patches for fancyvrb”. Available from CTAN, <https://www.ctan.org/pkg/fvextra>, 2026.
- [6] HOFFMANN, JOBST. “The listings package”. Available from CTAN, <https://www.ctan.org/pkg/listings>, 2025.
- [7] POORE, GEOFFREY M. “The minted package - Highlighted source code in L^AT_EX”. Available from CTAN, <https://www.ctan.org/pkg/minted>, 2026.
- [8] WRIGHT, JOSEPH. “answers – Setting questions (or exercises) and answers”. Available from CTAN, <https://ctan.org/pkg/answers>, 2014.

- [9] F. STURM, THOMAS. “tcolorbox – Coloured boxes, for L^AT_EX examples and theorems, etc”. Available from CTAN, <https://ctan.org/pkg/tcolorbox>, 2026.
- [10] The L^AT_EX Project. “The xparse package”. Available from CTAN, <https://www.ctan.org/pkg/xparse>, 2024.
- [11] The L^AT_EX Project. “The L^AT_EX 2_ε sources”. Available from CTAN, <https://ctan.org/tex-archive/macros/latex/base>, 2026.
- [12] The L^AT_EX Project. “L^AT_EX for authors current version”. Available from CTAN, <https://ctan.org/pkg/latex-base>, 2026.
- [13] NIEDERBERGER, CLEMENS. “xsim – eXercise Sheets IMproved”. Available from CTAN, <https://www.ctan.org/pkg/xsim>, 2023.
- [14] The L^AT_EX Project. “latex-lab – L^AT_EX laboratory”. Available from CTAN, <https://www.ctan.org/pkg/latex-lab>, 2026.
- [15] The L^AT_EX Project. “verbatim – Reimplementation of and extensions to L^AT_EX verbatim”. Available from CTAN, <https://www.ctan.org/pkg/verbatim>, 2023.
- [16] The L^AT_EX Project. “The L^AT_EX 3 Interfaces”. Available from CTAN, <https://www.ctan.org/pkg/l3kernel>, 2026.
- [17] WRIGHT, JOSEPH. “Programming key-value in expl3”. Available from TUGBOAT, <https://www.tug.org/TUGboat/tb31-1/tb97wright-l3keys.pdf>, 2010.

11 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`[1] package to the current version with only `expl3`[2].

- v2.8 (ctan), 2026-11-01**
 - Update requirement to L^AT_EX release 2026-11-01.
 - Fix warning for empty argument in `\Scontents` and `\Scontents*`.
 - Add error for empty sequence names and undefined sequences.
 - Check positive integers in `start`, `step`, `stop` and `width-tab` keys.
 - The messages for writing and overwriting files only go to the `.log` file.
 - Fix `wrapper` key and allow `\par` in the optional argument of `\foreachsc`.
 - Fix negative `<index>` in `\typestored`, `\meaningsc` and `\mergesc`.
 - Fix the check of existing `<output files>` in ConT_EXt for files written in the same run.
 - Fix loading in ConT_EXt when `expl3` was already loaded by another module.
 - The messages of the environments show their start and end as written in plain T_EX and ConT_EXt.
 - The “error” `index-out-of-range` shows the limit of the sequence and the “errors” `index-out-of-range` and `empty-variable` have a help text.
 - Fix `{<seq name>}` without `<index>`, undefined sequences and `<index>` out of range in `\mergesc`.
 - Fix TAB at the beginning of a line in `\typestored` and `\mergesc`.
 - Fix empty line at the beginning of `\typestored` and `\mergesc` in plain T_EX.
 - Unknown `<keys>` in the package options, `\setupsc` and the `<initial keys>` of `\newenvsc` return an “error” that names the command.
 - Added regression tests for L^AT_EX, plain T_EX and ConT_EXt, run with `l3build` check.
 - New `remember` key to use the `<stored contents>` before they are stored, only in L^AT_EX (GitHub issue #1).
 - New `getstored` key for `\mergesc` and `getstored`, `typestored` and `meaningsc` keys for `\foreachsc` (GitHub issue #10).
 - New programming interface for `expl3`: `\scontents_if_exist:nTF`, `\scontents_count:n`, `\scontents_get:nnN` and `\scontents_map_inline:nn` (GitHub issue #10).
- v2.7 (ctan), 2026-06-01**
 - Internal environment `verbatimsc` update for *tagged* PDF.
 - Update requirement to L^AT_EX release 2026-06-01.
 - Add `debug-var` key for writing variables to the `.log` file.
 - Update and improvements documentation and examples.
- v2.6 (ctan), 2025-11-20**
 - Internal environment `verbatimsc` update for *tagged* PDF.
 - Update requirement to L^AT_EX release 2025-11-01.
- v2.5 (ctan), 2025-11-01**
 - Replacing `\scantokens` (primitive) with `\tl_retokenize:n`.

	<ul style="list-style-type: none"> – Cleanup warnings and details returned by <code>expltools</code>. – Update minimum required of <code>expl3</code> to 2025-07-08.
v2.4 (ctan), 2025-05-15	<ul style="list-style-type: none"> – Optimization of expansion code from ‘x’ to ‘e’. – Restructuring code for documentation and implementation. – Add new keys for <code>\typestored</code> and <code>\meaningsc</code>.
v2.3 (ctan), 2025-04-23	<ul style="list-style-type: none"> – Check the version of <code>expl3</code> in plain $\TeX$ and Con$\TeX$t. – Adapting the <code>verbatimsc</code> environment for <i>tagged</i> PDF. – Update minimum required to $\LaTeX$ release 2024-11-01. – Safer code for replacement <code>\obeyedline</code>.
v2.2 (ctan), 2025-03-26	<ul style="list-style-type: none"> – Fix internal definition for some functions. – Replace <code>\peek_charcode_ignore_spaces:NTF</code> by <code>\peek_charcode:NTF</code>. – Set correct code for <code>\obeyedline</code> implement in $\LaTeX$ release 2024-06-01.
v2.1 (ctan), 2024-06-14	<ul style="list-style-type: none"> – Fix <code>\cleanseqsc</code> command. – Add <code>\mergesc</code> command. – Fix internal definition for seq var. – Fix internal code for <code>\typestored</code>. – Replace <code>\cs_argument_spec:N</code> by <code>\cs_parameter_spec:N</code>. – Detect <code>l3keys2e</code> package (obsolete in june 2022 $\LaTeX$ release). – Minor adjustments in the documentation.
v2.0 (ctan), 2022-04-04	<ul style="list-style-type: none"> – Adapting the <code>verbatimsc</code> environment (compatibility <code>verbatim</code> package). – Removed compatibility layer for older $\LaTeX$ releases. – Fix loader in plain $\TeX$ and Con$\TeX$t. – Minor adjustments in the documentation.
v1.9 (ctan), 2020-01-21	<ul style="list-style-type: none"> – Update and improvements in the internal code. – Updating the generic code for I/O verification. – Add <code>write-cmd</code> and <code>write-out</code> keys for <code>\Scontents*</code>. – Fix <code>sep</code> key in <code>\foreachsc</code>.
v1.8 (ctan), 2019-11-18	<ul style="list-style-type: none"> – Add <code>\newenvsc</code> command. – Fix nested environment in plain $\TeX$ and Con$\TeX$t. – Modified default value in <code>\getstored</code>. – Add <code>overwrite</code> key to reduce I/O operations. – Deleted an unnecessary group in the code.
v1.7 (ctan), 2019-10-29	<ul style="list-style-type: none"> – The <code>verbatimsc</code> environment was rewritten. – Minor adjustments in documentation.
v1.6 (ctan), 2019-10-26	<ul style="list-style-type: none"> – The internal behavior of <code>\getstored</code> has been modified. – The internal behavior of <code>\foreachsc</code> has been modified. – Corrected file extension for Con$\TeX$t. – Remove spurious warning.
v1.5 (ctan), 2019-10-24	<ul style="list-style-type: none"> – Add support for plain $\TeX$ and Con$\TeX$t. – Split internal code for optimization. – Add support for vertical spaces in <code>[⟨key = val⟩]</code>. – Add <code>\foreachsc</code> command. – Check if <code>verbatim</code> package is loaded.
v1.4 (ctan), 2019-10-03	<ul style="list-style-type: none"> – Add <code>store-all</code> key. – Messages and keys were separated. – Restructuring of documentation. – Now the version of <code>expl3</code> is checked instead of <code>xparse</code>. – The internal behavior of <code>force-eol</code> has been modified.
v1.3 (ctan), 2019-09-24	<ul style="list-style-type: none"> – The environment <code>scontents</code> can now nest. – Added <code>force-eol</code>, <code>verb-font</code> and <code>width-tab</code> keys. – The extra space has been removed when you run <code>\getstored</code>. – Internal code has been rewritten more efficiently. – Remove <i>starred argument</i> ‘*’ for <code>\typestored</code>. – Remove <code>filecontentsdef</code> dependency. – Changing <code>\regex_replace_all:</code> for <code>\tl_replace_all:</code>.
v1.2 (ctan), 2019-08-28	<ul style="list-style-type: none"> – Restructuring of documentation. – Added copy of <code>\tex_scantokens:</code>.
v1.1 (ctan), 2019-08-12	<ul style="list-style-type: none"> – Extension of documentation. – Replace <code>\tex_endinput:D</code> by <code>\file_input_stop:</code>.
v1.0 (ctan), 2019-07-30	<ul style="list-style-type: none"> – First public release.

12 Index of Documentation

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

C		Keys for \Scontents provided by SCONTENTS:	
Commands provided by SCONTENTS:		print-cmd	6
\Scontents	5	store-cmd	6
\cleanseqsc	9	write-cmd	6
\countsc	8	write-out	6
\foreachsc	6	Keys for \setupsc provided by SCONTENTS:	
\getstored	6	debug-var	3
\meaningsc	7	force-eol	3
\mergesc	7	overwrite	3
\newenvsc	5	print-all	3
\setupsc	3	remember	3
\typestored	7	store-all	3
		verb-font	3
		width-tab	3
E		Keys for environments provided by SCONTENTS:	
Environments :		print-env	5
filecontentsdefmacro	1	store-env	5
lstlisting	3	write-env	5
Verbatim	3	write-out	5
Environments provided by SCONTENTS:			
scontents	4		
K		P	
Keys for \foreachsc provided by SCONTENTS:		Packages :	
after	7	answers	10
before	7	expl3	1, 2, 9, 18, 19
getstored	6	fancyvrb	3, 8
meaningsc	7	filecontentsdef	1, 9, 10, 18, 19
sep	7	fvextra	3, 5, 8, 13
start	7	l3keys2e	19
step	7	l3seq	1, 9
stop	7	listings	3, 5, 8, 14
typestored	6	minted	8, 14
wrapper	7	tcolorbox	13
		verbatim	19
		xparse	19

13 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>.

- All variables and functions defined in this package are private and are NOT intended to work or be used by another package or module, except the functions of the programming interface (§7).

13.1 General conventions

The package is split into a *loader* and a *core*. The loader is different for each format and generates `scontents.sty` for $\text{\LaTeX}$, `scontents.tex` for plain $\text{\TeX}$ and `t-scontents.mkiv` for Con $\text{\TeX}$ t, the core is shared by the three formats and generates `scontents-code.tex`, which is loaded by each loader.

Since the core runs in plain $\text{\TeX}$ and Con $\text{\TeX}$ t, where the names of some primitives may be redefined by the format, the core uses the `\tex_...:D` names for the primitives that have no `expl3` interface, such as `\tex_kern:D`, `\tex_lastskip:D` or `\tex_spacefactor:D`.

Each command and environment is implemented in the same order: its variables, its messages, its *keys*, its internal functions and finally the public command or environment.

13.2 Declaration of the package

First we set up the module name for `docstrip` and the `l3doc` class:

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

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

```
2 <*loader>
3 \def\ScontentsFileDate{2026-11-01}%
4 \def\ScontentsFileVersion{2.8}%
5 \def\ScontentsFileDescription{Stores LaTeX contents in memory or files}%
```

The $\text{\LaTeX}$ loader is quite simple, we just need to make sure of the minimum version for correct operation and then set interfaces up. The choice of $\text{\LaTeX}$ release 2026-11-01 is the latest available in $\text{\TeX}$ Live 2026 and is necessary to be able to implement the package's full compatibility with *tagged* PDF.

```
6 <*latex>
7 \NeedsTeXFormat{LaTeX2e}[2026-11-01]
8 \ProvidesExplPackage
9   {scontents} {\ScontentsFileDate} {\ScontentsFileVersion}
10  {\ScontentsFileDescription}
11 </latex>
```

13.2.1 Load `expl3-generic` and `xparse-generic` in plain $\text{\TeX}$ and Con $\text{\TeX}$ t

The plain $\text{\TeX}$ and Con $\text{\TeX}$ t loaders are similar. We define the constant string `\c__scontents_version_str` with the version info and add `\ExplSyntaxOn` to be able to load the frozen `xparse`[10]. In Con $\text{\TeX}$ t the module is loaded after `\unprotect`, which makes `'!` a letter, but `expl3` and `xparse-generic.tex` use `'!` in boolean expressions and `expl3-generic.tex` does not set its category code, so `'!` is set to *other* before loading it and is a letter again at the end (§13.27).

```
12 <!*latex>
13 <context>\writestatus{loading}{User Module scontents v\ScontentsFileVersion}
14 <context>\unprotect
15 <context>\catcode`!=12\relax
16 \input expl3-generic.tex
17 \ExplSyntaxOn
18 \str_const:ce { c__scontents_version_str } { \ScontentsFileDate\space
19   v\ScontentsFileVersion\space \ScontentsFileDescription }
20 \iow_log:e { Package: ~ scontents ~ \str_use:c { c__scontents_version_str } }
```

Outside of $\text{\LaTeX}$ we can't use `\lcmd`[11, 12], part of the $\text{\LaTeX}$ kernel, but the old `xparse`[10] provides `xparse-generic.tex`, which is loadable in any format. The integer variable `\l__scontents_char_value_int` saves the category code of `'@` while it is loaded.

```
21 \int_new:N \l__scontents_char_value_int
22 \int_set:Nn \l__scontents_char_value_int { \char_value_catcode:n { \@ } }
23 \char_set_catcode_letter:N \@
24 \file_input:n { xparse-generic.tex }
25 \char_set_catcode:nn { \@ } { \l__scontents_char_value_int }
26 </!*latex>
```

(End of definition for `\c__scontents_version_str` and `\l__scontents_char_value_int`.)

13.2.2 Checking expl3 version

For plain T_EX, L^AT_EX and ConT_EXt we must check the minimum requirement, in this case `\tl_retokenize:n` which was added in release 2025-07-08 of [expl3](#) included in T_EX Live 2025.

```

27 \cs_if_exist:NF \tl_retokenize:n
28 {
29   \msg_new:nnn { scontents } { expl-too-old }
30   {
31     Please~install~an~up~to~date~TeX~distribution~or~update~using~
32     your~TeX~package~manager~or~from~CTAN.  \l
33     See~documentation.~Loading~scontents~will~abort!
34   }
35   \msg_fatal:nn { scontents } { expl-too-old }
36   \ExplSyntaxOff
37   \file_input_stop:
38 }

```

13.2.3 Preventing double loading in plain T_EX

In plain T_EX, check that the package isn't being loaded twice, L^AT_EX and ConT_EXt already defend against that:

```

39 (*plain)
40 \tl_if_exist:NT \l__scontents_cmd_name_tl
41 {
42   \msg_new:nnn { scontents } { already-loaded }
43   {
44     The~'scontents'~package~is~already~loaded.~
45     Aborting~input~\msg_line_context:.
46   }
47   \msg_warning:nn { scontents } { already-loaded }
48   \ExplSyntaxOff
49   \file_input_stop:
50 }
51 (/plain)
52 (/loader)

```

13.2.4 Checking proper loader

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

```

53 (*core)
54 \def\ScontentsCoreFileDate{2026-11-01}%
55 \begingroup
56   \catcode32=10
57   \endlinechar=32
58   \def\next{\endgroup}%
59   \expandafter\ifx\csname PackageError\endcsname\relax
60     \begingroup
61       \def\next{\endgroup\endgroup}%
62       \def\PackageError#1#2#3%
63         {%
64           \endgroup
65           \errhelp{#3}%
66           \errmessage{#1 Error: #2!}%
67         }%
68     \fi
69     \expandafter\ifx\csname ScontentsFileDate\endcsname\relax
70       \def\next
71         {%
72           \PackageError{scontents}{No scontents loader detected}
73             {%
74               You have attempted to use the scontents code directly rather
75               than using the correct loader. Loading of scontents will abort.
76             }%
77           \endgroup
78           \endinput
79         }%
80     \else
81       \ifx\ScontentsFileDate\ScontentsCoreFileDate
82       \else
83         \def\next

```

```

84     {%
85     \PackageError{scontents}{Mismatched scontents files detected}
86     {%
87     You have attempted to load scontents with mismatched files:
88     probably you have one or more files 'locally installed' which
89     are in conflict. Loading of scontents will abort.
90     }%
91     \endgroup
92     \endinput
93     }%
94     \fi
95     \fi
96     \next
97     </core>

```

13.3 Variables and functions by format

We define and set the variables and the function that must be handled separately in order to work properly with plain $\TeX$, $\LaTeX$ and Con $\TeX$ t.

`\_scontents_format_case:nnn` The function `\_scontents_format_case:nnn` leaves the argument for the current format: `#1` for $\LaTeX$, `#2` for plain $\TeX$ and `#3` for Con $\TeX$ t.

```

98 (*loader)
99 \cs_new:Npn \_scontents_format_case:nnn #1 #2 #3
100 <latex> {#1} % LaTeX
101 <plain> {#2} % Plain/Generic
102 <context> {#3} % ConTeXt

```

(End of definition for `\_scontents_format_case:nnn`.)

`\_scontents_file_if_exist:nTF` The conditional `\_scontents_file_if_exist:nTF` tests if the *<output file>* passed in `#1` exists in the working directory. In $\LaTeX$ and plain $\TeX$ it is `\file_if_exist:nTF`. In Con $\TeX$ t the file search keeps the result found at the start of the run, so a file written by the package in the same run is not found; there the test is done with `lfs.isfile` of Lua $\TeX$, the *<output file>* is always in the working directory.

```

103 <!context> \cs_new_eq:NN \_scontents_file_if_exist:nTF \file_if_exist:nTF
104 <*context>
105 \prg_new_conditional:Npnn \_scontents_file_if_exist:n #1 { TF }
106 {
107     \str_if_eq:eeTF
108     {
109         \lua_now:e
110         { tex.write(lfs.isfile("\lua_escape:e {#1}")~and~"1"~or~"0") }
111     }
112     { 1 }
113     { \prg_return_true: }
114     { \prg_return_false: }
115 }
116 </context>

```

(End of definition for `\_scontents_file_if_exist:nTF`.)

`\c_scontents_newline_tl` The constant token list `\c_scontents_newline_tl` stores the character ‘`^^J`’ used as end of line in the *<stored content>*.

```

117 <*!context>
118 \tl_const:Nc \c_scontents_newline_tl { \iow_char:N ^^J }
119 </!context>

```

In Con $\TeX$ t the package does not work under LMTX, so loading is aborted there. The test with `\contextlmtxmode` is adapted from the file `t-lua-widow-control.mkx1` of Max Chernoff’s `lua-widow-control` package, `\contextlmtxmode` is described in <https://source.contextgarden.net/tex/context/base/mkx1/context.mkx1>.

```

120 <*context>
121 \bool_if:NTF \contextlmtxmode
122 {
123     \msg_new:nnn { scontents } { luametateX }
124     {
125         The '~!scontents!~package~doesn't~work~under~LMTX.
126     }
127     \msg_fatal:nn { scontents } { luametateX }

```

```

128   }
129   {
130     \tl_const:Ne \c__scontents_newline_tl { \iow_char:N ^^J }
131   }
132 </context>

```

(End of definition for `\c__scontents_newline_tl`.)

```

\g__scontents_end_verbatimsc_tl
\c__scontents_end_env_tl
\l__scontents_env_name_tl

```

The global token list `\g__scontents_end_verbatimsc_tl` matches the end of the `verbatimsc` environment (§13.15) in each format.

```

133 \tl_new:N \g__scontents_end_verbatimsc_tl
134 \tl_gset_rescan:Nnn \g__scontents_end_verbatimsc_tl
135   {
136     \char_set_catcode_other:N \
137 < *latex>
138     \char_set_catcode_other:N \{
139     \char_set_catcode_other:N \}
140 < /latex>
141   }
142 < latex> { \end{verbatimsc} }
143 < plain> { \endverbatimsc }
144 < context> { \stopverbatimsc }

```

The constant token list `\c__scontents_end_env_tl` matches the end of the environments defined by `\newenvsc` and the token list `\l__scontents_env_name_tl` stores the *name* of the environment being processed (§13.13).

```

145 \tl_new:N \l__scontents_env_name_tl
146 \tl_const:Ne \c__scontents_end_env_tl
147   {
148     \c_backslash_str
149 < latex|plain> end
150 < context> stop
151 < latex> \c_left_brace_str
152     \exp_not:N \l__scontents_env_name_tl
153 < latex> \c_right_brace_str
154   }

```

(End of definition for `\g__scontents_end_verbatimsc_tl`, `\c__scontents_end_env_tl`, and `\l__scontents_env_name_tl`.)

13.4 Loading the package core

Now we load the core `SCONTENTS` code:

```

155 \file_input:n { scontents-code.tex }
156 </loader>

```

13.5 Some kernel variants

```

\tl_replace_all:NeV
\tl_retokenize:e

```

Some nonstandard kernel variants.

```

157 < *core>
158 \cs_generate_variant:Nn \tl_replace_all:Nnn { NeV }
159 \cs_generate_variant:Nn \tl_retokenize:n { e }

```

(End of definition for `\tl_replace_all:NeV` and `\tl_retokenize:e`.)

13.6 General variables

```

\q__scontents_stop
\q__scontents_mark
\s__scontents_stop
\s__scontents_mark

```

Quarks and scan marks used along the code as delimiters.

```

160 \quark_new:N \q__scontents_stop
161 \quark_new:N \q__scontents_mark
162 \scan_new:N \s__scontents_stop
163 \scan_new:N \s__scontents_mark

```

(End of definition for `\q__scontents_stop` and others.)

```

\l__scontents_cmd_name_tl

```

The token list `\l__scontents_cmd_name_tl` stores the *name* of the command that receives the *<keys>*: `\Scontents`, `\foreachsc`, `\typestored`, `\meaningsc`, `\mergesc`, `\setupsc`, `\newenvsc` or `\usepackage` for the options of the package. It is used in the messages for unknown *<keys>* (§13.9).

```

164 \tl_new:N \l__scontents_cmd_name_tl

```

(End of definition for `\l__scontents_cmd_name_tl`.)

```
\l__scontents_file_name_tl
\l__scontents_file_write_iow
\l__scontents_writing_bool
\l__scontents_storing_bool
\l__scontents_writable_bool
```

The token list `\l__scontents_file_name_tl` stores the name of the *⟨output file⟩* set by the `write-env`, `write-out` and `write-cmd` keys and `\l__scontents_file_write_iow` is the *output stream* used to write it.

The boolean variables `\l__scontents_writing_bool` and `\l__scontents_storing_bool`, set by the same keys, determine whether the content is written to the *⟨output file⟩* and whether it is stored, `\l__scontents_storing_bool` is true by default. The boolean variable `\l__scontents_writable_bool` keeps track of whether the *⟨output file⟩* was opened by `\l__scontents_file_if_writable:nTF` (§13.10).

```
165 \tl_new:N \l__scontents_file_name_tl
166 \iow_new:N \l__scontents_file_write_iow
167 \bool_new:N \l__scontents_writing_bool
168 \bool_new:N \l__scontents_storing_bool
169 \bool_new:N \l__scontents_writable_bool
170 \bool_set_true:N \l__scontents_storing_bool
```

(End of definition for `\l__scontents_file_name_tl` and others.)

```
\l__scontents_seq_item_seq
\g__scontents_name_sc!internal_seq
```

The sequence `\l__scontents_seq_item_seq` stores the *⟨indexes⟩* of the *⟨stored content⟩* requested to `\typestored`, `\meaningsc` or `\mergesc` and the sequence `\g__scontents_name_sc!internal_seq` is the one assembled by `\mergesc` (§13.22).

```
171 \seq_new:N \l__scontents_seq_item_seq
172 \seq_new:c { \g__scontents_name_sc!internal_seq }
```

(End of definition for `\l__scontents_seq_item_seq` and `\g__scontents_name_sc!internal_seq`.)

```
\g__scontents_last_stored_tl
```

The token list `\g__scontents_last_stored_tl` is used by the function `\__scontents_lastfrom_seq:n` (§13.11) to execute the last *⟨stored content⟩* outside the group.

```
173 \tl_new:N \g__scontents_last_stored_tl
```

(End of definition for `\g__scontents_last_stored_tl`.)

```
\c__scontents_hidden_space_str
```

The constant string `\c__scontents_hidden_space_str` hides the *⟨forced space⟩* added by T_EX when recording content in a macro. It contains the *reserved phrase* ‘`%^^Ascheol%`’, which is added to the end of the *⟨stored content⟩* when the key `force-eol` is false.

```
174 \str_const:Ne \c__scontents_hidden_space_str
175 { \c_percent_str \c_circumflex_str \c_circumflex_str A scheol \c_percent_str }
```

(End of definition for `\c__scontents_hidden_space_str`.)

```
\l__scontents_save_sf_int
\l__scontents_save_skip
```

Variables used by the functions `\__scontents_bsphack:` and `\__scontents_esphack:` (§13.10.2).

```
176 \int_new:N \l__scontents_save_sf_int
177 \skip_new:N \l__scontents_save_skip
```

(End of definition for `\l__scontents_save_sf_int` and `\l__scontents_save_skip`.)

13.7 General messages

13.7.1 Definition of error messages

```
undefined-storage
index-out-of-range
```

The “error” **undefined-storage** is raised when the *sequence* passed to `\getstored`, `\foreachsc`, `\typestored`, `\meaningsc`, `\mergesc`, `\countsc` or to the functions of the programming interface does not exist (§13.11, §13.18, §13.22, §13.24, §13.26) and the “error” **index-out-of-range** is raised when the *⟨index⟩* is zero or its absolute value is greater than the number of *⟨stored contents⟩* (§13.11, §13.22, §13.26).

```
178 \msg_new:nnnn { scontents } { undefined-storage }
179 { Storage~named~'#1'~is~not~defined. }
180 { No~content~has~been~stored~in~a~sequence~named~'#1'. }
181 \msg_new:nnnn { scontents } { index-out-of-range }
182 {
183   \int_compare:nNnTF {#1} = { 0 }
184   { Index~of~sequence~'#2'~cannot~be~zero. }
185   {
186     Index~'#1'~out~of~range~for~'#2',~the~
187     \int_compare:nNnTF {#1} > { 0 }
188     { maximum~is~#3. }
189   }
```

```

189         { minimum~is~-#3. }
190     }
191 }
192 {
193     The~sequence~'#2'~has~#3~stored~contents,~use~an~index~from~
194     1~to~#3~or~from~#3~to~-1.
195 }

```

(End of definition for *undefined-storage* and *index-out-of-range*.)

positive-integer The “error” **positive-integer** is raised when the keys **width-tab**, **start**, **step** or **stop** receive a value that is not a *positive integer* (§13.8, §13.18.2) and the “error” **empty-store-name** is raised when the name of the *sequence* is blank (§13.11).

```

196 \msg_new:nnnn { scontents } { positive-integer }
197 { The~key~'#1'~needs~a~positive-integer,~'#2'~given~\msg_line_context:. }
198 { The~value~'#2'~is~ignored~and~the~key~'#1'~keeps~its~previous~value. }
199 \msg_new:nnnn { scontents } { empty-store-name }
200 { Empty~sequence~name~for~storing~content~\msg_line_context:. }
201 {
202     The~keys~'store-env',~'store-cmd'~and~'store-all'~need~a~non-empty~
203     sequence~name.~The~content~will~not~be~stored.
204 }

```

(End of definition for *positive-integer* and *empty-store-name*.)

13.7.2 Definition of messages for writing files

writing-file The “info” messages **writing-file** and **overwrite-file** are written to the .log file when an *output file* is written or overwritten and the “warning” **not-writing** is issued when the *output file* already exists and the key **overwrite** is not active (§13.10).

```

205 \msg_new:nnn { scontents } { writing-file }
206 { Writing~file~'#1'. }
207 \msg_new:nnn { scontents } { overwrite-file }
208 { Overwriting~file~'#1'. }
209 \msg_new:nnn { scontents } { not-writing }
210 { File~'#1'~already~exists.~Not~writing~\msg_line_context:. }
211 </core>

```

(End of definition for *writing-file*, *overwrite-file*, and *not-writing*.)

13.8 Keys for the package

\l__scontents_tab_width_int The integer variable `\l__scontents_tab_width_int` stores the value of the **width-tab** key. The function `\__scontents_set_positive_int:Nnn` sets the integer variable passed in **#1** to the value passed in **#3** only if it is a *positive integer*, otherwise it raises the “error” **positive-integer** naming the *key* passed in **#2** and the value is ignored. It is used by the **width-tab** key and by the **start**, **step** and **stop** keys of `\foreachsc` (§13.18.2). The core code does not use ‘!’ in boolean expressions, so it does not depend on the category code of ‘!’ in ConT_EXt (§13.2.1).

```

212 <*core>
213 \int_new:N \l__scontents_tab_width_int
214 \cs_new_protected:Npn \__scontents_set_positive_int:Nnn #1 #2 #3
215 {
216     \bool_lazy_or:nnTF
217     { \tl_if_blank_p:n {#3} }
218     { \int_compare_p:nNn {#3} < { 1 } }
219     { \msg_error:nnnn { scontents } { positive-integer } {#2} {#3} }
220     { \int_set:Nn #1 {#3} }
221 }

```

(End of definition for `\l__scontents_tab_width_int` and `\__scontents_set_positive_int:Nnn`.)

store-env We define the common *keys* used by the options passed to the package, by `\setupsc` and by the environments and commands. The unknown *keys* are passed to `\__scontents_unknown_keys_cmd:n` (§13.9), which names the command from `\l__scontents_cmd_name_tl`: `\usepackage` for the options of the package, `\setupsc` or `\newenvsc` for the *initial keys* of an environment.

```

222 \keys_define:nn { scontents }
223 {
224     store-env .tl_set:N      = \l__scontents_name_seq_env_tl,
225     store-env .initial:n    = contents,

```

```

226   store-env .value_required:n = true,
227   store-cmd .tl_set:N         = \l__scontents_name_seq_cmd_tl,
228   store-cmd .initial:n        = contents,
229   store-cmd .value_required:n = true,
230   verb-font .tl_set:N         = \l__scontents_verb_font_tl,
231   verb-font .value_required:n = true,
232   print-env .bool_set:N       = \l__scontents_print_env_bool,
233   print-env .initial:n        = false,
234   print-env .default:n        = true,
235   print-cmd .bool_set:N       = \l__scontents_print_cmd_bool,
236   print-cmd .initial:n        = false,
237   print-cmd .default:n        = true,
238   force-eol .bool_set:N       = \l__scontents_forced_eol_bool,
239   force-eol .initial:n        = false,
240   force-eol .default:n        = true,
241   overwrite .bool_set:N       = \l__scontents_overwrite_bool,
242   overwrite .initial:n        = false,
243   overwrite .default:n        = true,
244   width-tab .code:n           =
245     { \l__scontents_set_positive_int:Nnn \l__scontents_tab_width_int { width-tab } {#1} },
246   width-tab .initial:n        = 1,
247   width-tab .value_required:n = true,
248   print-all .meta:n          = { print-env = #1, print-cmd = #1 },
249   print-all .default:n        = true,
250   store-all .meta:n          = { store-env = #1, store-cmd = #1 },
251   store-all .value_required:n = true,
252   debug-var .bool_set:N       = \l__scontents_debug_var_bool,
253   debug-var .initial:n        = true,
254   debug-var .default:n        = true,
255   remember .code:n           = { \l__scontents_remember:n {#1} },
256   remember .value_required:n = true,
257   remember .usage:n           = preamble,
258   unknown .code:n             = { \l__scontents_unknown_keys_cmd:n {#1} },
259 }
260 </core>

```

The default value of the `verb-font` key depends on the format.

```

261 <loader>\keys_define:nn { scontents }
262 <latex> { verb-font .initial:n = \ttfamily }
263 <plain|context> { verb-font .initial:n = \tt }

```

In L^AT_EX the `<keys>` passed as options to the package are processed and an unknown `<key>` returns an “error”.

```

264 <*latex>
265 \tl_set:Nn \l__scontents_cmd_name_tl { usepackage }
266 \ProcessKeyOptions [ scontents ]
267 </latex>

```

(End of definition for `store-env` and others.)

13.9 Handling unknown keys

The `<keys>` that are not known by the environments and the commands are passed to the functions defined here. The name of the `<key>` is taken from `\l_keys_key_str` and the value, if any, is passed as argument.

13.9.1 Definition of error messages for unknown keys

The “errors” `env-key-unknown` and `env-key-value-unknown` are raised when an unknown `<key>`, without or with value, is passed to an environment defined by `\newenvsc` (§13.13.4).

```

268 <*core>
269 \msg_new:nnnn { scontents } { env-key-unknown }
270 {
271   The~key~'#1'~is~unknown~by~environment~
272   '\l__scontents_env_name_tl'~and~is~being~ignored.
273 }
274 {
275   The~environment~'\l__scontents_env_name_tl'~does~not~have~a~key~called~'#1'.~\\
276   Check~that~you~have~spelled~the~key~name~correctly.
277 }
278 \msg_new:nnnn { scontents } { env-key-value-unknown }
279 {

```

```

280   The~key~'#1=#2'~is~unknown~by~environment~
281   '\__scontents_env_name_tl'~and~is~being~ignored.
282 }
283 {
284   The~environment~'\__scontents_env_name_tl'~does~not~have~a~key~called~'#1'.\\
285   Check~that~you~have~spelled~the~key~name~correctly.
286 }

```

(End of definition for `env-key-unknown` and `env-key-value-unknown`.)

`cmd-key-unknown` The “errors” `cmd-key-unknown` and `cmd-key-value-unknown` are raised when an unknown *(key)*, with-
`cmd-key-value-unknown` out or with value, is passed to the commands `\Scontents`, `\foreachsc`, `\typestored`, `\meaningsc`,
`\mergesc` or `\setupsc`, to the *(initial keys)* of `\newenvsc` or as an option of the package (§13.8, §13.16.3,
§13.18.2, §13.19.2).

```

287 \msg_new:nnnn { scontents } { cmd-key-unknown }
288 {
289   The~key~'#1'~is~unknown~by~command~
290   \c_backslash_str \__scontents_cmd_name_tl \c_space_tl
291   and~is~being~ignored~\msg_line_context:.
292 }
293 {
294   The~command~\c_backslash_str \__scontents_cmd_name_tl \c_space_tl
295   does~not~have~a~key~called~'#1'.\\
296   Check~that~you~have~spelled~the~key~name~correctly.
297 }
298 \msg_new:nnnn { scontents } { cmd-key-value-unknown }
299 {
300   The~key~'#1=#2'~is~unknown~by~command~
301   \c_backslash_str \__scontents_cmd_name_tl \c_space_tl
302   and~is~being~ignored~\msg_line_context:.
303 }
304 {
305   The~command~\c_backslash_str \__scontents_cmd_name_tl \c_space_tl
306   does~not~have~a~key~called~'#1'.\\
307   Check~that~you~have~spelled~the~key~name~correctly.
308 }

```

(End of definition for `cmd-key-unknown` and `cmd-key-value-unknown`.)

13.9.2 Internal functions for unknown keys

`\__scontents_unknown_keys_env:n` The function `\__scontents_unknown_keys_env:n` raises the “error” for an unknown *(key)* passed to
`\__scontents_unknown_keys_env:nn` an environment defined by `\newenvsc`.

```

309 \cs_new_protected:Npn \__scontents_unknown_keys_env:n #1
310 { \exp_args:NV \__scontents_unknown_keys_env:nn \l_keys_key_str {#1} }
311 \cs_new_protected:Npn \__scontents_unknown_keys_env:nn #1 #2
312 {
313   \tl_if_blank:nTF {#2}
314   { \msg_error:nnn { scontents } { env-key-unknown } {#1} }
315   { \msg_error:nnnn { scontents } { env-key-value-unknown } {#1} {#2} }
316 }

```

(End of definition for `\__scontents_unknown_keys_env:n` and `\__scontents_unknown_keys_env:nn`.)

`\__scontents_unknown_keys_cmd:n` The function `\__scontents_unknown_keys_cmd:n` raises the “error” for an unknown *(key)* passed to
`\__scontents_unknown_keys_cmd:nn` the commands `\Scontents`, `\foreachsc`, `\mergesc` and `\setupsc`, to the *(initial keys)* of `\newenvsc`
and to the package.

```

317 \cs_new_protected:Npn \__scontents_unknown_keys_cmd:n #1
318 { \exp_args:NV \__scontents_unknown_keys_cmd:nn \l_keys_key_str {#1} }
319 \cs_new_protected:Npn \__scontents_unknown_keys_cmd:nn #1 #2
320 {
321   \tl_if_blank:nTF {#2}
322   { \msg_error:nnn { scontents } { cmd-key-unknown } {#1} }
323   { \msg_error:nnnn { scontents } { cmd-key-value-unknown } {#1} {#2} }
324 }

```

(End of definition for `\__scontents_unknown_keys_cmd:n` and `\__scontents_unknown_keys_cmd:nn`.)

13.10 Utility functions

Functions used in the implementation of `\newenvsc` (§13.13.5) and `\mergesc` (§13.22).

```

\__scontents_use_delimit_by_s_stop:nw
\__scontents_use_i_delimit_by_s_stop:nw
\__scontents_use_none_delimit_by_s_stop:w
\__scontents_use_none_delimit_by_q_stop:w

```

```

325 \cs_new:Npn \__scontents_use_delimit_by_s_stop:nw #1 \s__scontents_stop {#1}
326 \cs_new:Npn \__scontents_use_i_delimit_by_s_stop:nw #1 #2 \s__scontents_stop {#1}
327 \cs_new:Npn \__scontents_use_none_delimit_by_s_stop:w #1 \s__scontents_stop { }
328 \cs_new:Npn \__scontents_use_none_delimit_by_q_stop:w #1 \q__scontents_stop { }

```

(End of definition for `\__scontents_use_delimit_by_s_stop:nw` and others.)

```
\__scontents_tl_if_head_is_q_mark:nTF
```

The conditional `\__scontents_tl_if_head_is_q_mark:nTF` tests if the head of the token list is `\q__scontents_mark`.

```

329 \prg_new_protected_conditional:Npnn \__scontents_tl_if_head_is_q_mark:n #1
330 { F, TF }
331 {
332   \exp_after:w \if_meaning:w
333   \exp_after:w
334   \q__scontents_mark \__scontents_use_i_delimit_by_s_stop:nw #1 ? \s__scontents_stop
335   \prg_return_true:
336   \else:
337   \prg_return_false:
338   \fi:
339 }

```

(End of definition for `\__scontents_tl_if_head_is_q_mark:nTF`.)

```

\__scontents_remove_trailing_eol:N
\__scontents_remove_trailing_eol:w

```

The function `\__scontents_remove_trailing_eol:N` removes the string `\c__scontents_hidden_space_str` from the end of the token list passed in `#1`. It is used by `\typestored` (§13.20) and `\mergesc` (§13.22).

```

340 \cs_new_protected:Npe \__scontents_remove_trailing_eol:N #1
341 {
342   \exp_not:N \exp_after:w \exp_not:N \__scontents_remove_trailing_eol:w
343   #1 \s__scontents_stop \c__scontents_hidden_space_str \s__scontents_stop \s__scontents_mark
344 }
345 \use:e
346 {
347   \cs_new_protected:Npn \exp_not:N \__scontents_remove_trailing_eol:w #1
348   \c__scontents_hidden_space_str \s__scontents_stop #2 \s__scontents_mark #3
349 } {
350   \tl_set:Ne #3
351   {
352     \tl_if_empty:nTF {#2}
353     { \exp_not:o { \__scontents_use_delimit_by_s_stop:nw #1 } }
354     { \exp_not:n {#1} }
355   }
356 }

```

(End of definition for `\__scontents_remove_trailing_eol:N` and `\__scontents_remove_trailing_eol:w`.)

```
\__scontents_file_if_writable:nTF
```

The conditional `\__scontents_file_if_writable:nTF` tests if the *output file* passed in `#1` can be written, according to the `write-env`, `write-cmd`, `write-out` and `overwrite` keys, and issues the messages for writing files (§13.7). The *output file* is searched with `\__scontents_file_if_exist:nTF`.

```

357 \prg_new_protected_conditional:Npnn \__scontents_file_if_writable:n #1 { T, TF }
358 {
359   \bool_if:NTF \l__scontents_writing_bool
360   {
361     \__scontents_file_if_exist:nTF {#1}
362     {
363       \bool_if:NTF \l__scontents_overwrite_bool
364       {
365         \msg_info:nne { scontents } { overwrite-file } {#1}
366         \prg_return_true:
367       }
368       {
369         \msg_warning:nne { scontents } { not-writing } {#1}
370         \prg_return_false:
371       }
372     }
373     {
374       \msg_info:nne { scontents } { writing-file } {#1}

```

```

375         \prg_return_true:
376     }
377 }
378 { \prg_return_false: }
379 }

```

(End of definition for `\__scontents_file_if_writable:nTF`.)

```

\__scontents_file_write_cmd:nn
\__scontents_file_write_cmd:VV

```

The function `\__scontents_file_write_cmd:nn` writes the content passed in #2 to the *⟨output file⟩* passed in #1, it is used by the commands.

```

380 \cs_new_protected:Npn \__scontents_file_write_cmd:nn #1 #2
381 {
382     \__scontents_file_if_writable:nT {#1}
383     {
384         \iow_open:Nn \__scontents_file_write_iow {#1}
385         \iow_now:Nn \__scontents_file_write_iow {#2}
386         \iow_close:N \__scontents_file_write_iow
387     }
388 }
389 \cs_generate_variant:Nn \__scontents_file_write_cmd:nn { VV }

```

(End of definition for `\__scontents_file_write_cmd:nn`.)

```

\__scontents_set_write_out:n
\__scontents_set_write_store:n

```

The function `\__scontents_set_write_out:n` sets the name of the *⟨output file⟩* passed in #1 and only writes the content, it is used by the `write-out` key. The function `\__scontents_set_write_store:n` also stores the content, it is used by the `write-env` and `write-cmd` keys (§13.13.4, §13.16.3, §13.19.2).

```

390 \cs_new_protected:Npn \__scontents_set_write_out:n #1
391 {
392     \bool_set_false:N \__scontents_storing_bool
393     \bool_set_true:N \__scontents_writing_bool
394     \tl_set:Nn \__scontents_file_name_tl {#1}
395 }
396 \cs_new_protected:Npn \__scontents_set_write_store:n #1
397 {
398     \__scontents_set_write_out:n {#1}
399     \bool_set_true:N \__scontents_storing_bool
400 }

```

(End of definition for `\__scontents_set_write_out:n` and `\__scontents_set_write_store:n`.)

13.10.1 Functions for TAB and verbatimsc

```

\__scontents_tab:
\__scontents_par:

```

Control sequences to replace the tab ‘`^^I`’ and form feed ‘`^^L`’ characters.

```

401 \cs_new:Nc \__scontents_tab: { \c_space_tl }
402 \cs_new:Nn \__scontents_par: { ^^J ^^J }

```

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

```

\__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. The spaces are *active* characters, the same as the spaces of the *verbatim* context: a space token at the beginning of a line would be dropped in vertical mode and the TAB that indent a line would be lost. The *active* space is made with `\tex_lowercase:D` from the *active* character ‘`^^@`’, since `\char_generate:nn` would expand it while defining the function.

```

403 \group_begin:
404 \char_set_catcode_active:N ^^@
405 \char_set_lccode:nn { 0 } { 32 }
406 \tex_lowercase:D
407 {
408     \group_end:
409     \cs_new:Npn \__scontents_tabs_to_spaces:
410         { \prg_replicate:nn { \__scontents_tab_width_int } { ^^@ } }
411 }

```

(End of definition for `\__scontents_tabs_to_spaces:`.)

```

\__scontents_do_noligs:N

```

The function `\__scontents_do_noligs:N` is an alternative definition for $\text{\TeX 2}_{\epsilon}$ ’s `\do@noligs` which makes sure to not consume the following space tokens. The $\text{\TeX 2}_{\epsilon}$ version ends with `\char`#1`, which leaves $\text{\TeX}$ still looking for an *⟨optional space⟩*.

```

412 \cs_new_protected:Npn \__scontents_do_noligs:N #1

```

```

413 {
414   \char_set_catcode_active:N #1
415   \cs_set:cpe { __scontents_active_char_ \token_to_str:N #1 : }
416   {
417     \mode_leave_vertical:
418     \tex_kern:D \c_zero_dim
419     \tex_char:D \exp_not:N #1
420   }
421   \char_set_active_eq:Nc #1 { __scontents_active_char_ \token_to_str:N #1 : }
422 }

```

(End of definition for `\__scontents_do_noligs:N`)

Shortcut definitions for common catcode changes. The ‘`^^L`’ needs a special treatment in plain T_EX because it is an `\outer` token there.

```

\__scontents_set_active_eq:NN
\__scontents_plain_disable_outer_par:
\__scontents_make_control_chars_active:
423 \cs_new_protected:Npn \__scontents_set_active_eq:NN #1
424 {
425   \char_set_catcode_active:N #1
426   \char_set_active_eq:NN #1
427 }
428 </core>
429 <*loader>
430 \group_begin:
431 <plain> \char_set_catcode_active:n { \* }
432 \cs_new_protected:Nn \__scontents_plain_disable_outer_par:
433 <*plain>
434 {
435   \group_begin:
436   \char_set_lccode:nn { \* } { \^^L }
437   \tex_lowercase:D { \group_end:
438   \tex_let:D * \scan_stop:
439   }
440 }
441 </plain>
442 <latex|context> { }
443 \group_end:
444 </loader>
445 <*core>
446 \cs_new_protected:Nn \__scontents_make_control_chars_active:
447 {
448   \__scontents_plain_disable_outer_par:
449   \__scontents_set_active_eq:NN \^^I \__scontents_tab:
450   \__scontents_set_active_eq:NN \^^L \__scontents_par:
451   \__scontents_set_active_eq:NN \^^M \__scontents_ret:w
452 }

```

(End of definition for `\__scontents_set_active_eq:NN`, `\__scontents_plain_disable_outer_par:`, and `\__scontents_make_control_chars_active:`)

13.10.2 Functions `\@bsphack` and `\@esphack`

`\__scontents_bsphack:` We emulate `\@bsphack` and `\@esphack` for plain T_EX and ConT_EXt, this is necessary to prevent *unwanted spaces* when the `print-cmd` key is false. In L_T_EX the kernel definitions are used.

```

453 \cs_new_protected:Nn \__scontents_bsphack:
454 {
455   \scan_stop:
456   \mode_if_horizontal:T
457   {
458     \skip_set_eq:NN \l__scontents_save_skip \tex_lastskip:D
459     \int_set_eq:NN \l__scontents_save_sf_int \tex_spacefactor:D
460   }
461 }
462 \cs_new_protected:Nn \__scontents_esphack:
463 {
464   \scan_stop:
465   \mode_if_horizontal:T
466   {
467     \int_set_eq:NN \tex_spacefactor:D \l__scontents_save_sf_int
468     \dim_compare:nNnT { \l__scontents_save_skip } > { \c_zero_skip }
469     {
470       \skip_if_eq:nnT { \tex_lastskip:D } { \c_zero_skip }

```

```

471         {
472             \nobreak
473             \skip_horizontal:n { \c_zero_skip }
474         }
475         \tex_ignorespaces:D
476     }
477 }
478 }
479 </core>
480 <*latex>
481 \cs_gset_eq:NN \__scontents_bsphack: \@bsphack
482 \cs_gset_eq:NN \__scontents_esphack: \@esphack
483 </latex>

```

(End of definition for __scontents_bsphack: and __scontents_esphack:.)

13.11 Functions for sequences

The *storage system* of the package `SCONTENTS` is done using seq variables. Here we set up the functions that manage them.

```

\__scontents_append_contents:nn
\__scontents_append_contents:Ve

```

The function `\__scontents_append_contents:nn` creates the *sequence* passed in `#1` if it does not exist and appends the $\{\langle body\ env\rangle\}$ of an environment or the $\{\langle argument\rangle\}$ of a command, passed in `#2`, to the right of the *sequence*.

```

484 <*core>
485 \cs_new_protected:Npn \__scontents_append_contents:nn #1 #2
486 {
487     \seq_if_exist:cF { g__scontents_name_#1_seq }
488     { \seq_new:c { g__scontents_name_#1_seq } }
489     \seq_gput_right:cn { g__scontents_name_#1_seq } {#2}
490 }
491 \cs_generate_variant:Nn \__scontents_append_contents:nn { Ve }

```

(End of definition for __scontents_append_contents:nn.)

```

\__scontents_store_to_seq:NN

```

The function `\__scontents_store_to_seq:NN` writes the recorded content in `#1` to the `.log` file if `debug-var` is true and stores it in the *sequence* named by `#2`.

```

492 \cs_new_protected:Npn \__scontents_store_to_seq:NN #1 #2
493 {
494     \bool_if:NT \__scontents_debug_var_bool
495     {
496         \tl_log:N #1
497     }
498     \__scontents_append_contents:Ve #2 { \exp_not:V #1 }
499 }

```

(End of definition for __scontents_store_to_seq:NN.)

```

\__scontents_finish_storing:NNN

```

The function `\__scontents_finish_storing:NNN` works only in the *storing mode*, when the `write-out` key is not active. If the key `force-eol` is false, its default value, `\__scontents_hidden_space_str` is added to the end of the token list passed in `#1`, which contains the $\{\langle body\ env\rangle\}$ of an environment or the $\{\langle argument\rangle\}$ of `\Scontents`.

Then `\__scontents_store_to_seq:NN` stores the content in the *sequence* named by `#2` and, if the boolean passed in `#3`, set by the `print-env` or `print-cmd` key, is true, the content is printed from the *sequence* by `\__scontents_lastfrom_seq:V`. If the *name* of the *sequence* is blank, the “error” `empty-store-name` is raised and nothing is stored or printed.

```

500 \cs_new_protected:Npn \__scontents_finish_storing:NNN #1 #2 #3
501 {
502     \bool_if:NT \__scontents_storing_bool
503     {
504         \tl_if_blank:VTF #2
505         { \msg_error:nn { scontents } { empty-store-name } }
506         {
507             \bool_if:NF \__scontents_forced_eol_bool
508             { \tl_put_right:Ne #1 { \__scontents_hidden_space_str } }
509             \__scontents_store_to_seq:NN #1 #2
510             \bool_if:NT #3 { \__scontents_lastfrom_seq:V #2 }
511         }
512     }
513 }

```

(End of definition for `\__scontents_finish_storing:NNN`.)

```
\__scontents_seq_count:n
\__scontents_seq_item:nn
\__scontents_seq_if_exist:nTF
```

A *sequence* set by the `remember` key has a second copy with the *stored contents* of the previous compilation, whose name uses `prev` instead of `name` (§13.12). The conditional `\__scontents_seq_if_exist:nTF` tests if any of the two *sequences* named by `#1` exists, the function `\__scontents_seq_count:n` returns the larger number of *stored contents* of the two, or `0` if none exists, and the function `\__scontents_seq_item:nn` returns the *stored content* at the *index* passed in `#2`, from the previous compilation if it has it and from the current one otherwise. Without `remember`, and always in plain $\text{\TeX}$ and $\text{\ConTeXt}$, only the current *sequence* exists.

```
514 \prg_new_conditional:Npnn \__scontents_seq_if_exist:n #1 { TF }
515 {
516   \bool_lazy_or:nnTF
517   { \seq_if_exist_p:c { g__scontents_name_#1_seq } }
518   { \seq_if_exist_p:c { g__scontents_prev_#1_seq } }
519   { \prg_return_true: }
520   { \prg_return_false: }
521 }
522 \cs_new:Npn \__scontents_seq_count:n #1
523 {
524   \int_max:nn
525   {
526     \seq_if_exist:cTF { g__scontents_name_#1_seq }
527     { \seq_count:c { g__scontents_name_#1_seq } }
528     { 0 }
529   }
530   {
531     \seq_if_exist:cTF { g__scontents_prev_#1_seq }
532     { \seq_count:c { g__scontents_prev_#1_seq } }
533     { 0 }
534   }
535 }
536 \cs_new:Npn \__scontents_seq_item:nn #1 #2
537 {
538   \bool_lazy_and:nnTF
539   { \seq_if_exist_p:c { g__scontents_prev_#1_seq } }
540   {
541     \int_compare_p:nNn { \int_abs:n {#2} }
542     < { \seq_count:c { g__scontents_prev_#1_seq } + 1 }
543   }
544   { \seq_item:cn { g__scontents_prev_#1_seq } {#2} }
545   { \seq_item:cn { g__scontents_name_#1_seq } {#2} }
546 }
```

(End of definition for `\__scontents_seq_count:n`, `\__scontents_seq_item:nn`, and `\__scontents_seq_if_exist:nTF`.)

```
\__scontents_getfrom_seq:Nn
\__scontents_getfrom_seq:nNn
\__scontents_getfrom_seq_aux:nnn
\__scontents_getfrom_seq:nn
\__scontents_getfrom_seq:nnn
```

The function `\__scontents_getfrom_seq:Nn` retrieves the *stored contents* whose *indexes* are in the sequence passed in `#1` from the *sequence* named by `#2` and the function `\__scontents_getfrom_seq:nNn` retrieves the one whose *index* is passed in `#1`, both with `\__scontents_seq_item:nn`. Both raise the “errors” `undefined-storage` and `index-out-of-range` (§13.7).

```
547 \cs_new:Npn \__scontents_getfrom_seq:Nn #1 #2
548 {
549   \__scontents_seq_if_exist:nTF {#2}
550   {
551     \exp_args:Nf \__scontents_getfrom_seq:nNn
552     { \__scontents_seq_count:n {#2} } #1 {#2}
553   }
554   { \msg_expandable_error:nnn { scontents } { undefined-storage } {#2} }
555 }
556 \cs_new:Npn \__scontents_getfrom_seq:nNn #1 #2 #3
557 {
558   \seq_map_tokens:Nn #2 { \__scontents_getfrom_seq_aux:nnn {#1} {#3} }
559 }
560 \cs_new:Npn \__scontents_getfrom_seq_aux:nnn #1 #2 #3
561 {
562   \exp_args:Nnf \use:n
563   { \__scontents_getfrom_seq:nnn {#1} } { \int_eval:n {#3} } {#2}
564 }
565 \cs_new:Npn \__scontents_getfrom_seq:nn #1 #2
566 {
```

```

567   \__scontents_seq_if_exist:nTF {#2}
568   {
569     \exp_args:Nf \__scontents_getfrom_seq:nnn
570     { \__scontents_seq_count:n {#2} }
571     {#1} {#2}
572   }
573   { \msg_expandable_error:nnn { scontents } { undefined-storage } {#2} }
574 }
575 \cs_new:Npn \__scontents_getfrom_seq:nnn #1 #2 #3
576 {
577   \bool_lazy_or:nnTF
578   { \int_compare_p:nNn {#2} = { 0 } }
579   { \int_compare_p:nNn { \int_abs:n {#2} } > {#1} }
580   {
581     \msg_expandable_error:nnnn { scontents } { index-out-of-range }
582     {#2} {#3} {#1}
583   }
584   { \__scontents_seq_item:nn {#3} {#2} }
585 }

```

(End of definition for `\__scontents_getfrom_seq:Nn` and others.)

```

\__scontents_lastfrom_seq:n
\__scontents_lastfrom_seq:V

```

The function `\__scontents_lastfrom_seq:n` saves the last *stored content* of the *sequence* named by `#1` in `\g__scontents_last_stored_tl` and rescans it with `\tl_retokenize:V` after the group, when the keys `print-env` or `print-cmd` are active.

```

586 \cs_new_protected:Npn \__scontents_lastfrom_seq:n #1
587 {
588   \tl_gset:Ne \g__scontents_last_stored_tl
589   {
590     \seq_item:cn { g__scontents_name_#1_seq } { -1 }
591   }
592   \group_insert_after:N \tl_retokenize:V
593   \group_insert_after:N \g__scontents_last_stored_tl
594   \group_insert_after:N \tl_gclear:N
595   \group_insert_after:N \g__scontents_last_stored_tl
596 }
597 \cs_generate_variant:Nn \__scontents_lastfrom_seq:n { V }
598 </core>

```

(End of definition for `\__scontents_lastfrom_seq:n`.)

13.12 Remembering contents between compilations

The `remember` key (§13.8) makes the *stored contents* of some *sequences* available before they are stored, as `\label` and `\ref` do with the references. At the end of the document the *stored contents* of these *sequences* are written to the file `\jobname.sct` and at the beginning of the next compilation they are read in a second *sequence*, that is used by `\__scontents_seq_item:nn` (§13.11). Each line of the file has the name of the *sequence* and one *stored content*, both converted to hexadecimal digits and separated by ‘=’, so that any character can be written and read again. This is only available in $\TeX$.

13.12.1 Definition of variables for remembering contents

```

\g__scontents_remember_seq
\g__scontents_remember_pending_bool

```

The global *sequence* `\g__scontents_remember_seq` stores, as strings, the names of the *sequences* set by the `remember` key and the boolean `\g__scontents_remember_pending_bool` is true when a *stored content* of one of them is requested and it is not available yet.

```

599 <*core>
600 \seq_new:N \g__scontents_remember_seq
601 \bool_new:N \g__scontents_remember_pending_bool
602 </core>

```

(End of definition for `\g__scontents_remember_seq` and `\g__scontents_remember_pending_bool`.)

```

\g__scontents_remember_ior
\g__scontents_remember_iow
\g__scontents_remember_old_str
\l__scontents_remember_new_str
\l__scontents_remember_name_str
\l__scontents_remember_item_str

```

The streams `\g__scontents_remember_ior` and `\g__scontents_remember_iow` read and write the file `\jobname.sct`. The string `\g__scontents_remember_old_str` saves the lines read from the file and `\l__scontents_remember_new_str` the lines written, to compare them at the end of the document, and the strings `\l__scontents_remember_name_str` and `\l__scontents_remember_item_str` save the name of the *sequence* and the *stored content* of a line.

```

603 <*latex>
604 \ior_new:N \g__scontents_remember_ior

```

```

605 \iow_new:N \g__scontents_remember_iow
606 \str_new:N \g__scontents_remember_old_str
607 \str_new:N \l__scontents_remember_new_str
608 \str_new:N \l__scontents_remember_name_str
609 \str_new:N \l__scontents_remember_item_str
610 \latex

```

(End of definition for `\g__scontents_remember_iow` and others.)

13.12.2 Definition of error messages for remembering contents

`remember-latex-only` The “error” `remember-latex-only` is raised when the `remember` key is used in plain $\text{T}_{\text{E}}\text{X}$ or $\text{ConT}_{\text{E}}\text{Xt}$.

```

611 \*core
612 \msg_new:nnnn { scontents } { remember-latex-only }
613 { The~key~'remember'~is~only~available~in~LaTeX~\msg_line_context:. }
614 {
615     The~stored~contents~can~only~be~used~after~they~are~stored~
616     in~this~format.~The~key~is~ignored.
617 }
618 \core

```

(End of definition for `remember-latex-only`.)

13.12.3 Definition of messages for remembering contents

`remember-rerun` The “warning” `remember-rerun` is issued at the end of the document when the lines written to `\jobname.sct` are not the ones read from it, and otherwise the “warning” `remember-undefined` when a `\stored content` was requested and it was not available.

`remember-undefined`

```

619 \*latex
620 \msg_new:nnn { scontents } { remember-rerun }
621 { Stored~contents~may~have~changed.~Rerun~to~get~them~right. }
622 \msg_new:nnn { scontents } { remember-undefined }
623 { There~were~undefined~stored~contents. }
624 \latex

```

(End of definition for `remember-rerun` and `remember-undefined`.)

13.12.4 Internal functions for remembering contents

`\__scontents_remember:n` The function `\__scontents_remember:n`, used by the `remember` key, adds each name of the comma separated list passed in `#1` to `\g__scontents_remember_seq` as a string, only once. In plain $\text{T}_{\text{E}}\text{X}$ and $\text{ConT}_{\text{E}}\text{Xt}$ it raises the “error” `remember-latex-only`.

```

625 \*core
626 \cs_new_protected:Npn \__scontents_remember:n #1
627 {
628     \__scontents_format_case:nnn
629     {
630         \clist_map_inline:nn {#1}
631         {
632             \seq_if_in:Nef \g__scontents_remember_seq { \tl_to_str:n {##1} }
633             { \seq_gput_right:Nef \g__scontents_remember_seq { \tl_to_str:n {##1} } }
634         }
635     }
636     { \msg_error:nn { scontents } { remember-latex-only } }
637     { \msg_error:nn { scontents } { remember-latex-only } }
638 }

```

(End of definition for `\__scontents_remember:n`.)

`\__scontents_if_pending:nTF`
`\__scontents_if_pending:nF`

The conditionals `\__scontents_if_pending:nnT` and `\__scontents_if_pending:nnF` test if the `\stored content` at the `\index` passed in `#2` of the remembered *sequence* named by `#1` is not available yet, that is, if its absolute value is greater than the number of `\stored contents` of both copies. In that case `\g__scontents_remember_pending_bool` is set to true and the commands print nothing. For the *sequences* that are not remembered it is always false and the “errors” of the commands are raised as usual.

```

639 \prg_new_protected_conditional:Npnn \__scontents_if_pending:nn #1 #2 { T, F }
640 {
641     \bool_lazy_and:nnTF
642     { \seq_if_exist_p:c { g__scontents_prev_#1_seq } }
643     { \int_compare_p:nNn { \int_abs:n {#2} } > { \__scontents_seq_count:n {#1} } } }
644 {

```

```

645     \bool_gset_true:N \g__scontents_remember_pending_bool
646     \prg_return_true:
647   }
648   { \prg_return_false: }
649 }
650 \prg_generate_conditional_variant:Nnn \__scontents_if_pending:n { ne } { F }
651 \</core>

```

(End of definition for `\__scontents_if_pending:nTF` and `\__scontents_if_pending:neF`.)

`\__scontents_remember_read:` The function `\__scontents_remember_read:` reads the file `\jobname.sct`, if it exists, line by line with `\__scontents_remember_read_line:n`, and creates an empty copy of the remembered *sequences* that are not in the file. The function `\__scontents_remember_read_line:w` converts the name of the *sequence* from hexadecimal digits and, if it is remembered, saves the line in `\g__scontents_remember_old_str` and adds the *stored content*, converted from hexadecimal digits, to the copy of the previous compilation. The lines of the *sequences* that are no longer remembered are ignored.

```

652 \*latex>
653 \cs_new_protected:Npn \__scontents_remember_read:
654 {
655   \ior_open:NnT \g__scontents_remember_ior { \c_sys_jobname_str .sct }
656   {
657     \ior_str_map_inline:Nn \g__scontents_remember_ior
658     { \__scontents_remember_read_line:n {##1} }
659     \ior_close:N \g__scontents_remember_ior
660   }
661   \seq_map_inline:Nn \g__scontents_remember_seq
662   {
663     \seq_if_exist:cF { g__scontents_prev_##1_seq }
664     { \seq_new:c { g__scontents_prev_##1_seq } }
665   }
666 }
667 \cs_new_protected:Npn \__scontents_remember_read_line:n #1
668 {
669   \str_if_empty:nF {#1}
670   { \__scontents_remember_read_line:w #1 = \s__scontents_stop }
671 }
672 \cs_new_protected:Npn \__scontents_remember_read_line:w #1 = #2 = #3 \s__scontents_stop
673 {
674   \str_set_convert:Nnnn \l__scontents_remember_name_str {#1} { utf8/hex } { }
675   \seq_if_in:NVT \g__scontents_remember_seq \l__scontents_remember_name_str
676   {
677     \str_gput_right:Nn \g__scontents_remember_old_str { #1 = #2 , }
678     \str_set_convert:Nnnn \l__scontents_remember_item_str {#2} { utf8/hex } { }
679     \seq_if_exist:cF { g__scontents_prev_ \l__scontents_remember_name_str_seq }
680     { \seq_new:c { g__scontents_prev_ \l__scontents_remember_name_str_seq } }
681     \seq_gput_right:cV { g__scontents_prev_ \l__scontents_remember_name_str_seq }
682     \l__scontents_remember_item_str
683   }
684 }

```

(End of definition for `\__scontents_remember_read:`, `\__scontents_remember_read_line:n`, and `\__scontents_remember_read_line:w`.)

`\__scontents_remember_write:` The function `\__scontents_remember_write:` writes the file `\jobname.sct` if some *sequence* is remembered: for each one that exists, `\__scontents_remember_write_seq:n` converts its name to hexadecimal digits and `\__scontents_remember_write_item:n` writes a line for each *stored content*, also saved in `\l__scontents_remember_new_str`. If the lines written are not the ones read, the “warning” `remember-rerun` is issued, otherwise `remember-undefined` if a *stored content* was not available.

```

685 \cs_new_protected:Npn \__scontents_remember_write:
686 {
687   \seq_if_empty:NF \g__scontents_remember_seq
688   {
689     \str_clear:N \l__scontents_remember_new_str
690     \iow_open:Nn \g__scontents_remember_iow { \c_sys_jobname_str .sct }
691     \seq_map_function:NN \g__scontents_remember_seq \__scontents_remember_write_seq:n
692     \iow_close:N \g__scontents_remember_iow
693     \str_if_eq:NNTF \g__scontents_remember_old_str \l__scontents_remember_new_str

```

```

694     {
695         \bool_if:NT \g__scontents_remember_pending_bool
696         { \msg_warning:nn { scontents } { remember-undefined } }
697     }
698     { \msg_warning:nn { scontents } { remember-rerun } }
699 }
700 }
701 \cs_new_protected:Npn \__scontents_remember_write_seq:n #1
702 {
703     \seq_if_exist:cT { g__scontents_name_#1_seq }
704     {
705         \str_set_convert:Nnnn \l__scontents_remember_name_str {#1} { } { utf8/hex }
706         \seq_map_function:cN { g__scontents_name_#1_seq }
707         \__scontents_remember_write_item:n
708     }
709 }
710 \cs_new_protected:Npn \__scontents_remember_write_item:n #1
711 {
712     \str_set_convert:Nnnn \l__scontents_remember_item_str {#1} { } { utf8/hex }
713     \iow_now:Ne \g__scontents_remember_iow
714     { \l__scontents_remember_name_str = \l__scontents_remember_item_str }
715     \str_put_right:Ne \l__scontents_remember_new_str
716     { \l__scontents_remember_name_str = \l__scontents_remember_item_str , }
717 }

```

(End of definition for `\__scontents_remember_write:`, `\__scontents_remember_write_seq:n`, and `\__scontents_remember_write_item:n`.)

The file is read at the beginning of the document and written at the end of it.

```

718 \hook_gput_code:nnn { begindocument } { { scontents }
719   { \__scontents_remember_read: }
720 \hook_gput_code:nnn { enddocument } { { scontents }
721   { \__scontents_remember_write: }
722 </latex>

```

13.13 The command `\newenvsc`

The user-level command `\newenvsc` creates a new environment with the same behavior as the base environment `scontents`, which is itself defined with `\newenvsc` (§13.14). It takes a mandatory argument, the $\langle env\ name \rangle$, and an optional argument with the $\langle initial\ keys \rangle$. The references to `\begin{scontents}` or `\end{scontents}` in this section apply to any environment defined with `\newenvsc`.

13.13.1 Definition of variables for `\newenvsc`

```

\l__scontents_save_every_body_lines_tl
\l__scontents_processed_body_lines_tl
\l__scontents_environment_keys_tl
\l__scontents_nesting_env_int
\l__scontents_nesting_aux_int

```

The token list `\l__scontents_save_every_body_lines_tl` holds the $\langle body\ env \rangle$ of the environment while it is being read and the token list `\l__scontents_processed_body_lines_tl` saves it once the environment is closed. The token list `\l__scontents_environment_keys_tl` saves the $\langle keys \rangle$ passed to the *optional argument* and the integer variables `\l__scontents_nesting_env_int` and `\l__scontents_nesting_aux_int` are used to analyze the nesting of the environment.

```

723 <*core>
724 \tl_new:N \l__scontents_save_every_body_lines_tl
725 \tl_new:N \l__scontents_processed_body_lines_tl
726 \tl_new:N \l__scontents_environment_keys_tl
727 \int_new:N \l__scontents_nesting_env_int
728 \int_new:N \l__scontents_nesting_aux_int

```

(End of definition for `\l__scontents_save_every_body_lines_tl` and others.)

13.13.2 Definition of error messages for `\newenvsc`

```

\__scontents_env_begin_str:n
\__scontents_env_end_str:n

```

The functions `\__scontents_env_begin_str:n` and `\__scontents_env_end_str:n` return the start and the end of the environment named by `#1` as they are written in each format: `\begin{#1}` and `\end{#1}` in $\text{\LaTeX}$, `\#1` and `\end#1` in plain $\text{\TeX}$ and `\start#1` and `\stop#1` in Con $\text{\TeX}$ t. They are used in the messages of the environments.

```

729 \cs_new:Npn \__scontents_env_begin_str:n #1
730 {
731     \__scontents_format_case:nnn
732     { \c_backslash_str begin \c_left_brace_str #1 \c_right_brace_str }
733     { \c_backslash_str #1 }
734     { \c_backslash_str start #1 }
735 }

```

```

736 \cs_new:Npn \__scontents_env_end_str:n #1
737 {
738   \__scontents_format_case:nnn
739   { \c_backslash_str end \c_left_brace_str #1 \c_right_brace_str }
740   { \c_backslash_str end #1 }
741   { \c_backslash_str stop #1 }
742 }

```

(End of definition for `\__scontents_env_begin_str:n` and `\__scontents_env_end_str:n`.)

junk-after-begin
env-already-defined

The “error” **junk-after-begin** is raised when there are characters after the start of the environment or its *optional argument* on the same line and the “error” **env-already-defined** is raised when `\newenvsc` is used with the name of an environment that already exists.

```

743 \msg_new:nnnn { scontents } { junk-after-begin }
744 {
745   Junk~characters~#1~\msg_line_context: :
746   \ \
747   #2
748 }
749 {
750   Nothing~is~allowed~after~the~start~of~the~environment~on~the~same~
751   line.~These~characters~are~ignored.
752 }
753 \msg_new:nnnn { scontents } { env-already-defined }
754 { Environment~'#1'~already~defined! }
755 {
756   You~have~used~\token_to_str:N \newenvsc \c_space_tl
757   with~an~environment~that~already~has~a~definition. \ \
758   The~existing~definition~of~'#1'~will~not~be~altered.
759 }

```

(End of definition for `junk-after-begin` and `env-already-defined`.)

13.13.3 Definition of messages for `\newenvsc`

empty-environment
rescanning-text
multiple-begin

The “warning” **empty-environment** is issued when the environment is empty, the “warning” **rescanning-text** when there are characters after the end of the environment on the same line and the “warning” **multiple-begin** when the environment starts more than once on the same line. The start and the end of the environment are written with `\__scontents_env_begin_str:n` and `\__scontents_env_end_str:n`.

```

760 \msg_new:nnn { scontents } { empty-environment }
761 { Environment~'#1'~is~empty~\msg_line_context:. }
762 \msg_new:nnn { scontents } { rescanning-text }
763 {
764   Rescanning~text~'#1'~after~\__scontents_env_end_str:n {#2}~
765   \msg_line_context:.
766 }
767 \msg_new:nnn { scontents } { multiple-begin }
768 {
769   Multiple~\__scontents_env_begin_str:n { \__scontents_env_name_tl }~
770   \msg_line_context:.
771 }

```

(End of definition for `empty-environment`, `rescanning-text`, and `multiple-begin`.)

13.13.4 Definition of keys for `\newenvsc`

write-env
write-out
print-env
store-env
force-eol
overwrite
unknown

We define the *keys* for the environments defined by `\newenvsc`, the unknown *keys* are passed to `\__scontents_unknown_keys_env:n` (§13.9).

```

772 \keys_define:nn { scontents / scontents }
773 {
774   write-env .code:n = { \__scontents_set_write_store:n {#1} },
775   write-out .code:n = { \__scontents_set_write_out:n {#1} },
776   write-env .value_required:n = true,
777   write-out .value_required:n = true,
778   print-env .meta:nn = { scontents } { print-env = #1 },
779   print-env .default:n = true,
780   store-env .meta:nn = { scontents } { store-env = #1 },
781   force-eol .meta:nn = { scontents } { force-eol = #1 },
782   force-eol .default:n = true,

```

```

783   overwrite .meta:n      = { scontents } { overwrite = #1 },
784   overwrite .default:n   = true,
785   unknown   .code:n      = { \__scontents_unknown_keys_env:n {#1} },
786 }

```

(End of definition for `write-env` and others.)

13.13.5 Internal functions for `\newenvsc`

`\__scontents_grab_opt_arg:w` The function `\__scontents_grab_opt_arg:w` is called with the tokens following `\begin{scontents}` when the next character is a '['. It is defined with `\lcmd[11]` to use its delimited argument processor.

```

787 \NewDocumentCommand \__scontents_grab_opt_arg:w { r[] }
788 {
789   \__scontents_grab_opt_arg:n {#1}
790 }

```

The function `\__scontents_grab_opt_arg:n` is called where '`^^M`' is active, so `\__scontents_normalise_line_` replaces the active '`^^M`' characters by spaces before setting the `(keys)`.

```

791 \cs_new_protected:Npn \__scontents_grab_opt_arg:n #1
792 {
793   \tl_set:Nn \__scontents_environment_keys_tl {#1}
794   \__scontents_normalise_line_ends:N \__scontents_environment_keys_tl
795   \keys_set:nV { scontents / scontents } \__scontents_environment_keys_tl
796   \__scontents_start_after_option:w
797 }

```

(End of definition for `\__scontents_grab_opt_arg:w` and `\__scontents_grab_opt_arg:n`.)

```

\__scontents_normalise_line_ends:N
\__scontents_start_environment:w
\__scontents_start_after_option:w
\__scontents_check_line_process:nn
\__scontents_stop_environment:

```

Here '`^^I`', '`^^L`' and '`^^M`' are active characters, so that the end of line can be used as a delimiter and $\TeX$ does not remove the space-like characters.

The function `\__scontents_start_environment:w` checks if the next token after `\begin{scontents}` is a '[', in that case `\__scontents_grab_opt_arg:w` processes the *optional argument* and calls `\__scontents_start_after_option:w`. In all cases the function `\__scontents_check_line_process:nn` raises the “error” `junk-after-begin` if there is something else on the line and then calls `\__scontents_file_tl_write_start:V`, which reads the content of the environment and stores it in a token list or writes it to an *(external file)*.

At the end of the environment, the function `\__scontents_stop_environment:` calls `\__scontents_file_write_stop:N` and issues the “warning” `empty-environment` if nothing was stored. This part of the code is adapted from the code of the package `xsim`[13] by Clemens Niederberger.

```

798 \group_begin:
799   \char_set_catcode_active:N \^^I
800   \char_set_catcode_active:N \^^L
801   \char_set_catcode_active:N \^^M
802   \cs_new_protected:Npn \__scontents_normalise_line_ends:N #1
803   { \tl_replace_all:Nnn #1 { ^^M } { ~ } }
804   \cs_new_protected:Npn \__scontents_start_environment:w #1 ^^M
805   {
806     \tl_if_head_is_N_type:nTF {#1}
807     {
808       \str_if_eq:eeTF { \tl_head:n {#1} } { [ ] }
809       { \__scontents_grab_opt_arg:w #1 ^^M }
810       { \__scontents_check_line_process:nn { } {#1} }
811     }
812     { \__scontents_check_line_process:nn { } {#1} }
813   }
814   \cs_new_protected:Npn \__scontents_start_after_option:w #1 ^^M
815   { \__scontents_check_line_process:nn { [...] } {#1} }
816   \cs_new_protected:Npn \__scontents_check_line_process:nn #1 #2
817   {
818     \tl_if_blank:nF {#2}
819     {
820       \msg_error:nnee { scontents } { junk-after-begin }
821       { after~ \__scontents_env_begin_str:n { \__scontents_env_name_tl } #1 } {#2}
822     }
823     \__scontents_make_control_chars_active:
824     \__scontents_file_tl_write_start:V \__scontents_file_name_tl
825   }
826   \cs_new_protected:Nn \__scontents_stop_environment:

```

```

827 {
828   \__scontents_file_write_stop:N \__scontents_processed_body_lines_tl
829   \bool_lazy_and:nnT
830     { \__scontents_storing_bool }
831     { \tl_if_empty_p:N \__scontents_processed_body_lines_tl }
832   {
833     \msg_warning:nne { scontents } { empty-environment }
834     { \__scontents_env_name_tl }
835   }
836 }

```

(End of definition for `\__scontents_normalise_line_ends:N` and others.)

```

\__scontents_file_tl_write_start:n
\__scontents_file_tl_write_start:V
\__scontents_setup_verb_processor:
\__scontents_verb_processor_iterate:w
\__scontents_verb_processor_iterate:nnn
\__scontents_env_end_function:
\__scontents_file_write_stop:N
\__scontents_remove_leading_nl:n
\__scontents_remove_leading_nl:nn
\__scontents_remove_leading_nl:w

```

This is the main function to collect the `{(body env)}` of the environment. The function `\__scontents_file_tl_write_start:n` starts a *group*, opens the *⟨output file⟩* if necessary, sets the *verbatim catcodes* and then issues ‘`^^M`’, set equal to `\__scontents_ret:w`, to read the `{(body env)}` line by line until its end. Each line read is appended to the output token list after an active ‘`^^J`’ and written to the *⟨output file⟩*, if any.

The function `\__scontents_setup_verb_processor:` defines `\__scontents_verb_processor_iterate:w`, delimited by the end of the current environment, and `\__scontents_verb_processor_iterate:nnn` decides if the line is a nested `\end{scontents}` or the end of the environment, in which case `\__scontents_env_end_function:` leaves the closing command for the current format and the characters after it are rescanned with the “warning” *rescanning-text*.

At the end of the environment, the function `\__scontents_file_write_stop:N` closes the *⟨output file⟩*, if it was opened, and smuggles the output token list out of the group. The leading ‘`^^J`’ is removed with `\__scontents_remove_leading_nl:n`, which expects an active ‘`^^J`’ at the head of the token list, otherwise a low level TeX “error” is raised.

```

837 \cs_new_protected:Npn \__scontents_file_tl_write_start:n #1
838 {
839   \group_begin:
840     \__scontents_file_if_writable:nTF {#1}
841     {
842       \bool_set_true:N \__scontents_writable_bool
843       \iow_open:Nn \__scontents_file_write_iow {#1}
844     }
845     { \bool_set_false:N \__scontents_writable_bool }
846     \tl_clear:N \__scontents_save_every_body_lines_tl
847     \seq_map_function:NN \__char_special_seq \char_set_catcode_other:N
848     \int_step_function:nnnN { 128 } { 1 } { 255 } \char_set_catcode_letter:n
849     \cs_set_protected:Npe \__scontents_ret:w ##1 ^^M
850     {
851       \exp_not:N \__scontents_verb_processor_iterate:w
852       ##1 \c__scontents_end_env_tl
853       \c__scontents_end_env_tl
854       \exp_not:N \q__scontents_stop
855     }
856     \__scontents_make_control_chars_active:
857     \__scontents_ret:w
858   }
859   \cs_new_protected:Nn \__scontents_setup_verb_processor:
860   {
861     \use:e
862     {
863       \cs_set_protected:Npn \exp_not:N \__scontents_verb_processor_iterate:w
864       ##1 \c__scontents_end_env_tl
865       ##2 \c__scontents_end_env_tl
866       ##3 \exp_not:N \q__scontents_stop
867     } { \__scontents_verb_processor_iterate:nnn {##1} {##2} {##3} }
868   }
869   \cs_new_protected:Npn \__scontents_verb_processor_iterate:nnn #1 #2 #3
870   {
871     \tl_if_blank:nTF {#3}
872     {
873       \__scontents_analyse_nesting:n {#1}
874       \__scontents_verb_processor_output:n {#1}
875     }
876     {
877       \__scontents_if_nested:TF
878       {

```

```

879         \_scontents_nesting_decr:
880         \_scontents_verb_processor_output:e
881         { \exp_not:n {#1} \c_scontents_end_env_tl \exp_not:n {#2} }
882     }
883     {
884         \tl_if_blank:nF {#1}
885         { \_scontents_verb_processor_output:n {#1} }
886         \cs_set_protected:Npe \_scontents_ret:w
887         {
888             \_scontents_env_end_function:
889             \bool_lazy_or:nnF
890             { \tl_if_blank_p:n {#2} }
891             { \str_if_eq_p:ee {#2} { \c_percent_str } }
892             {
893                 \str_if_eq:VnF \c_scontents_hidden_space_str {#2}
894                 {
895                     \msg_warning:nnnn { scontents } { rescanning-text }
896                     {#2} { \tl_use:N \l_scontents_env_name_tl }
897                 }
898                 \tl_retokenize:n {#2}
899             }
900         }
901         \char_set_active_eq:NN ^^M \_scontents_ret:w
902     }
903 }
904 ^^M
905 }
906 \cs_new:Nn \_scontents_env_end_function:
907 {
908     \_scontents_format_case:nnn
909     { \exp_not:N \end { \if_false: } \fi: }
910     { \exp_after:w \exp_not:N \cs:w end }
911     { \exp_after:w \exp_not:N \cs:w stop }
912     \tl_use:N \l_scontents_env_name_tl
913     \_scontents_format_case:nnn
914     { \if_false: { \fi: } }
915     { \cs_end: }
916     { \cs_end: }
917 }
918 \cs_new_protected:Npn \_scontents_file_write_stop:N #1
919 {
920     \bool_if:NT \l_scontents_writable_bool
921     { \iow_close:N \l_scontents_file_write_iow }
922     \use:e
923     {
924         \group_end:
925         \bool_if:NT \l_scontents_storing_bool
926         {
927             \tl_set:Nn \exp_not:N #1
928             {
929                 \exp_args:NV
930                 \_scontents_remove_leading_nl:n \l_scontents_save_every_body_lines_tl
931             }
932         }
933     }
934 }
935 \cs_new:Npn \_scontents_remove_leading_nl:n #1
936 {
937     \tl_if_head_is_N_type:nTF {#1}
938     {
939         \exp_args:Nf
940         \_scontents_remove_leading_nl:nn
941         { \tl_head:n {#1} } {#1}
942     }
943     { \exp_not:n {#1} }
944 }
945 \cs_new:Npn \_scontents_remove_leading_nl:nn #1 #2
946 {
947     \token_if_eq_meaning:NNTF ^^J #1
948     { \exp_not:o { \_scontents_remove_leading_nl:w #2 } }
949     { \exp_not:n {#2} }

```

```

950     }
951     \cs_new:Npn \__scontents_remove_leading_nl:w ^^^J { }

```

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

```

\__scontents_verb_processor_output:n
\__scontents_verb_processor_output:e

```

The function `\__scontents_verb_processor_output:n` does the output of each line read, to the token list and to the *⟨output file⟩*, according to the booleans `\__scontents_writable_bool` and `\__scontents_storing_bool`.

```

952     \cs_new_protected:Npn \__scontents_verb_processor_output:n #1
953     {
954         \bool_if:NT \__scontents_writable_bool
955         { \iow_now:Nn \__scontents_file_write_iow {#1} }
956         \bool_if:NT \__scontents_storing_bool
957         { \tl_put_right:Nn \__scontents_save_every_body_lines_tl { ^^^J #1 } }
958     }
959 \group_end:
960 \cs_generate_variant:Nn \__scontents_verb_processor_output:n { e }
961 \cs_generate_variant:Nn \__scontents_file_tl_write_start:n { V }

```

(End of definition for `\__scontents_verb_processor_output:n`.)

```

\__scontents_analyse_nesting:n
\__scontents_nesting_incr:
\__scontents_nesting_decr:
\__scontents_if_nested:TF

```

The function `\__scontents_analyse_nesting:n` scans the nested `\begin{scontents}` in a line and steps the counter `\__scontents_nesting_env_int`, issuing the “warning” **multiple-begin** if there is more than one. The conditional `\__scontents_if_nested:TF` tests if we are in a nested environment and `\__scontents_nesting_decr:` reduces the nesting level when an `\end{scontents}` is found.

```

962 \cs_new_protected:Npn \__scontents_analyse_nesting:n #1
963 {
964     \int_zero:N \__scontents_nesting_aux_int
965     \__scontents_analyse_nesting_format:n {#1}
966     \int_compare:nNnT { \__scontents_nesting_aux_int } > { 1 }
967     { \msg_warning:nn { scontents } { multiple-begin } }
968 }
969 \cs_new_protected:Nn \__scontents_nesting_incr:
970 {
971     \int_incr:N \__scontents_nesting_env_int
972     \int_incr:N \__scontents_nesting_aux_int
973 }
974 \cs_new_protected:Nn \__scontents_nesting_decr:
975 {
976     \int_decr:N \__scontents_nesting_env_int
977 }
978 \prg_new_protected_conditional:Npnn \__scontents_if_nested: { TF }
979 {
980     \int_compare:nNnTF { \__scontents_nesting_env_int } > { \c_zero_int }
981     { \prg_return_true: }
982     { \prg_return_false: }
983 }

```

(End of definition for `\__scontents_analyse_nesting:n` and others.)

🚫 Multiple `\end{scontents}` on the same line are NOT supported.

```

\__scontents_analyse_nesting_latex:n
\__scontents_analyse_nesting_latex:w
\__scontents_analyse_nesting_generic:nn
\__scontents_analyse_nesting_generic:w
\__scontents_analyse_nesting_generic_process:nn
\__scontents_define_generic_nesting_function:n
\__scontents_analyse_nesting_format:n

```

In $\text{\LaTeX}$ the environments start with `\begin{⟨env⟩}` and no ‘`’` can appear inside `⟨env⟩`, so a macro delimited by ‘`’` is enough to find the nested `\begin{scontents}`.

```

984 \use:e
985 {
986     \cs_new_protected:Npn \exp_not:N \__scontents_analyse_nesting_latex:w #1
987     \c_backslash_str begin \c_left_brace_str #2 \c_right_brace_str
988 } {
989     \__scontents_tl_if_head_is_q_mark:nTF {#2}
990     { \__scontents_use_none_delimit_by_q_stop:w }
991     {
992         \str_if_eq:VnT \__scontents_env_name_tl {#2}
993         { \__scontents_nesting_incr: }
994         \__scontents_analyse_nesting_latex:w
995     }
996 }
997 \cs_new_protected:Npe \__scontents_analyse_nesting_latex:n #1
998 {

```

```

999   \_scontents_analyse_nesting_latex:w #1
1000   \c_backslash_str begin
1001   \c_left_brace_str \exp_not:N \q__scontents_mark \c_right_brace_str
1002   \exp_not:N \q__scontents_stop
1003 }

```

In the other formats there is no “end anchor” to delimit the name of the environment, so we search for the whole command that starts the environment, `\scontents` or `\startscontents`.

```

1004 \cs_new_protected:Npn \_scontents_analyse_nesting_generic_process:nn #1 #2
1005 {
1006   \tl_if_head_is_N_type:nTF {#2}
1007   {
1008     \_scontents_tl_if_head_is_q_mark:nF {#2}
1009     {
1010       \_scontents_nesting_incr:
1011       \_scontents_analyse_nesting_generic:w #2 \q__scontents_stop
1012     }
1013   }
1014   { \_scontents_analyse_nesting_generic:w #2 \q__scontents_stop }
1015 }
1016 \cs_new_protected:Npn \_scontents_analyse_nesting_generic:nn #1 #2
1017 {
1018   \_scontents_define_generic_nesting_function:n {#1}
1019   \use:e
1020   {
1021     \exp_not:N \_scontents_analyse_nesting_generic:w #2
1022     \c_backslash_str #1 \tl_use:N \l__scontents_env_name_tl
1023     \exp_not:N \q__scontents_mark \exp_not:N \q__scontents_stop
1024   }
1025 }
1026 \cs_new_protected:Npn \_scontents_define_generic_nesting_function:n #1
1027 {
1028   \use:e
1029   {
1030     \cs_set_protected:Npn \exp_not:N \_scontents_analyse_nesting_generic:w ##1
1031     \c_backslash_str #1 \tl_use:N \l__scontents_env_name_tl
1032     ##2 \exp_not:N \q__scontents_stop
1033   } { \_scontents_analyse_nesting_generic_process:nn {##1} {##2} }
1034 }
1035 </core>

```

The function `\_scontents_analyse_nesting_format:n` uses the version for the current format.

```

1036 <*loader>
1037 <latex>\cs_new_eq:NN \_scontents_analyse_nesting_format:n
1038 <latex> \_scontents_analyse_nesting_latex:n
1039 <!latex>\cs_new_protected:Npn \_scontents_analyse_nesting_format:n
1040 <plain> { \_scontents_analyse_nesting_generic:nn { } }
1041 <context> { \_scontents_analyse_nesting_generic:nn { start } }
1042 </loader>

```

(End of definition for `\_scontents_analyse_nesting_latex:n` and others.)

`\_scontents_generic_begin:` The function `\_scontents_generic_begin:` leaves the ‘`^^M`’ character active and calls `\_scontents_start_environment:w`. The function `\_scontents_generic_end:` calls `\_scontents_stop_environment:` and then `\_scontents_finish_storing:NNN`, which stores the `{\body env}` in the *sequence* and prints it if the `print-env` key is active (§13.11).

```

1043 <*core>
1044 \cs_new_protected:Npn \_scontents_generic_begin:
1045 {
1046   \char_set_catcode_active:N ^^M
1047   \_scontents_start_environment:w
1048 }
1049 \cs_new_protected:Npn \_scontents_generic_end:
1050 {
1051   \_scontents_stop_environment:
1052   \_scontents_finish_storing:NNN
1053   \l__scontents_processed_body_lines_tl
1054   \l__scontents_name_seq_env_tl
1055   \l__scontents_print_env_bool
1056 }

```

(End of definition for `\__scontents_generic_begin:` and `\__scontents_generic_end:`.)

`\__scontents_setting_env:nn` The function `\__scontents_setting_env:nn` receives the *name* of the environment in **#1** and the *initial keys* in **#2**. It creates the functions `\__scontents_#1_begin:` and `\__scontents_#1_end:`, which set `\l__scontents_env_name_tl` and the *initial keys* and call `\__scontents_generic_begin:` and `\__scontents_generic_end:`, and passes them to `\__scontents_define_env:nnn`.

```

1057 \cs_new_protected:Npn \__scontents_setting_env:nn #1 #2
1058 {
1059   \cs_new_protected:cpn { __scontents_#1_begin: }
1060   {
1061     \tl_set:Nn \l__scontents_env_name_tl {#1}
1062     \tl_set:Nn \l__scontents_cmd_name_tl { newenvsc }
1063     \keys_set:nn { scontents } {#2}
1064     \__scontents_setup_verb_processor:
1065     \__scontents_generic_begin:
1066   }
1067   \cs_new_protected:cpn { __scontents_#1_end: }
1068   { \__scontents_generic_end: }
1069   \exp_args:Nooo \__scontents_define_env:nnn
1070   { \tl_to_str:n {#1} }
1071   { \cs:w __scontents_#1_begin: \cs_end: }
1072   { \cs:w __scontents_#1_end: \cs_end: }
1073 }
1074 </core>

```

The function `\__scontents_define_env:nnn` creates the environment for $\text{\LaTeX}$, plain $\text{\TeX}$ and Con $\text{\TeX}$ t.

```

1075 <*loader>
1076 \cs_new_protected:Npn \__scontents_define_env:nnn #1 #2 #3
1077 {
1078   <latex|plain>   \NewDocumentEnvironment {#1} { }
1079   <context>      \cs_new_protected:cpn { start #1 }
1080   {
1081     <!latex>      \group_begin:
1082                   #2
1083                   }
1084   <context>      \cs_new_protected:cpn { stop #1 }
1085   {
1086                   #3
1087   <!latex>      \group_end:
1088   }
1089 }

```

(End of definition for `\__scontents_setting_env:nn` and `\__scontents_define_env:nnn`.)

`\newenvsc` Defines the user command `\newenvsc` for $\text{\LaTeX}$, plain $\text{\TeX}$ and Con $\text{\TeX}$ t, raising the “error” `env-already-defined` if the environment exists.

```

1090 \NewDocumentCommand \newenvsc { m O{} }
1091 {
1092   <latex|plain>   \cs_if_exist:cTF {#1}
1093   <context>      \cs_if_exist:cTF { start #1 }
1094   { \msg_error:nnn { scontents } { env-already-defined } {#1} }
1095   { \__scontents_setting_env:nn {#1} {#2} }
1096 }
1097 </loader>

```

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

13.14 The environment scontents

`scontents` The base environment `scontents` is defined with `\newenvsc`, in plain $\text{\TeX}$ it is `\scontents ... \endscontents` and in Con $\text{\TeX}$ t `\startscontents ... \stopscontents`.

```

1098 <*loader>
1099 \newenvsc{scontents}
1100 </loader>

```

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

13.15 The environment verbatimsc

The `verbatimsc` environment is a customized version of the standard `verbatim` environment of $\text{\LaTeX}$ used by `\typestored` (§13.20). For plain $\text{\TeX}$ and Con $\text{\TeX}$ t we must add some additional functions.

13.15.1 Definition of variables for verbatimsc

`\_scontents_plain_bool` The boolean variable `\_scontents_plain_bool` is used in the plain $\text{\TeX}$ version of the `verbatimsc` environment.

```
1101 (*plain)
1102 \bool_new:N \_scontents_plain_bool
1103 (/plain)
```

(End of definition for `\_scontents_plain_bool`.)

13.15.2 Internal functions for verbatimsc

`\dospecials` The `verbatim` environment requires `\dospecials`. In case it does not exist when `SCONTENTS` is loaded, we define it with `\l_char_special_seq`.

```
1104 (*!latex)
1105 \cs_if_exist:NF \dospecials
1106 {
1107   \cs_new:Npn \dospecials
1108     { \seq_map_function:NN \l_char_special_seq \do }
1109 }
1110 (/!latex)
```

(End of definition for `\dospecials`.)

`\_scontents_xverb:w` The environment `verbatimsc` needs to find literally its end, this is done by `\_scontents_xverb:w`, delimited by `\g__scontents_end_verbatimsc_tl`, for plain $\text{\TeX}$ and $\text{\LaTeX}$. The function `\_scontents_xverb:` makes the TAB character active, equal to `\_scontents_tabs_to_spaces:`, and calls `\_scontents_xverb:w`.

```
1111 (*loader)
1112 (*!context)
1113 \use:e
1114 {
1115   \cs_new_protected:Npn \exp_not:N \_scontents_xverb:w
1116     #1 \g__scontents_end_verbatimsc_tl
1117   { \exp_not:N \end{verbatimsc} }
1118   { \exp_not:N \endverbatimsc }
1119 }
1120 (/!context)
1121 (/loader)
1122 (*core)
1123 \cs_new_protected:Nn \_scontents_xverb:
1124 {
1125   \char_set_catcode_active:n { 9 }
1126   \char_set_active_eq:nN { 9 } \_scontents_tabs_to_spaces:
1127   \_scontents_xverb:w
1128 }
1129 (/core)
```

(End of definition for `\_scontents_xverb:w` and `\_scontents_xverb:`.)

13.15.3 plain $\text{\TeX}$ version of verbatimsc

`\_scontents_verbatimsc_aux:` In plain $\text{\TeX}$ we emulate the `verbatim` environment of $\text{\LaTeX}$.

```
\_scontents_nolig_list:
\_scontents_vobeyspaces:
\_scontents_xobeysp:
1130 (*plain)
1131 \cs_new_protected:Nn \_scontents_verbatimsc_aux:
1132 {
1133   \skip_vertical:N \parskip
1134   \dim_zero:N \parindent
1135   \skip_set:Nn \parfillskip { 0pt plus 1fil }
1136   \skip_set:Nn \parskip { 0pt plus 0pt minus 0pt }
1137   \tex_par:D
1138   \bool_set_false:N \_scontents_plain_bool
1139   \cs_set:Npn \par
1140     {
1141       \bool_if:NTF \_scontents_plain_bool
1142       {
```

```

1143     \mode_leave_vertical:
1144     \null
1145     \tex_par:D
1146     \penalty \interlinepenalty
1147   }
1148   {
1149     \bool_set_true:N \l__scontents_plain_bool
1150     \mode_if_horizontal:T
1151     { \tex_par:D \penalty \interlinepenalty }
1152   }
1153 }
1154 \cs_set_eq:NN \do \char_set_catcode_other:N
1155 \dospecials \obeylines
1156 \tl_use:N \l__scontents_verb_font_tl
1157 \cs_set_eq:NN \do \__scontents_do_noligs:N
1158 \__scontents_nolig_list:
1159 \tex_everypar:D \exp_after:w
1160 { \tex_the:D \tex_everypar:D \tex_unpenalty:D }
1161 }
1162 \cs_new_protected:Nn \__scontents_nolig_list:
1163 { \do\` \do\< \do\> \do\, \do\' \do\~ }
1164 \cs_new_protected:Nn \__scontents_vobeyspaces:
1165 { \__scontents_set_active_eq:NN \ \__scontents_xobeysp: }
1166 \cs_new_protected:Nn \__scontents_xobeysp:
1167 { \mode_leave_vertical: \nobreak \ }

```

(End of definition for `\__scontents_verbatimsc_aux:` and others.)

`\__scontents_verbatimsc_line:w`

The function `\__scontents_verbatimsc_line:w` discards the rest of the line of `\verbatimsc` up to the active end of line. When `\typestored` rescans the *stored content* a space follows `\verbatimsc` and, once the spaces are active, it would start an empty line at the top of the environment.

```

1168 \group_begin:
1169   \char_set_catcode_active:N ^^M
1170   \cs_new_protected:Npn \__scontents_verbatimsc_line:w #1 ^^M
1171   {
1172     \__scontents_verbatimsc_aux: \frenchspacing \__scontents_vobeyspaces:
1173     \__scontents_xverb:
1174   }
1175 \group_end:

```

(End of definition for `\__scontents_verbatimsc_line:w`.)

`\verbatimsc` Defines the environment `verbatimsc` for plain T_EX. As in the environments of ConT_EXt, the rest of the line of `\verbatimsc` is ignored.

`\endverbatimsc`

```

1176 \cs_new_protected:Npn \verbatimsc
1177 {
1178   \group_begin:
1179   \char_set_catcode_active:N ^^M
1180   \__scontents_verbatimsc_line:w
1181 }
1182 \cs_new_protected:Npn \endverbatimsc
1183 { \group_end: }
1184 \plain

```

(End of definition for `\verbatimsc` and `\endverbatimsc`. These functions are documented on page 8.)

13.15.4 ConT_EXt version of `verbatimsc`

`\startverbatimsc`

`\stopverbatimsc`

In ConT_EXt we use its own tool `\definotyping`.

```

1185 \<loader>
1186 \<context>\definotyping[verbatimsc]
1187 \</loader>

```

(End of definition for `\startverbatimsc` and `\stopverbatimsc`. These functions are documented on page 8.)

13.15.5 L^AT_EX version of `verbatimsc`

`\__scontents_verbatimsc_instance:`

To be compatible with *tagged* PDF the environment `verbatimsc` is defined with the `xtemplate[11]` module of the L^AT_EX kernel and the blocks-code of `latex-lab[14]`. The function `\__scontents_verbatimsc_instance:` declares the instance of `blockenv` used by the environment,

only once at the beginning of the document and only if *tagged* PDF is active. This code is adapted from the answer of Ulrike Fischer to [New verbatim environment with block code \(tagged-pdf\)](#).

```

1188 (*loader)
1189 (*latex)
1190 \cs_new_protected:Nn \__scontents_verbatimsc_instance:
1191 {
1192   \DeclareInstance{blockenv}{verbatimsc}{std}
1193   {
1194     name                = verbatimsc,
1195     tag-name            = \UseStructureName{block/verbatim},
1196     tagging-recipe      = standard,
1197     tagging-suppress-paras = true,
1198     block-instance      = verbatim,
1199     para-instance       = justify,
1200     final-code          = \legacyverbatimsetup{invisible},
1201   }
1202 }
1203 \hook_gput_code:nnn { begindocument } { { scontents }
1204 {
1205   \IfDocumentMetadataTF { \__scontents_verbatimsc_instance: } { { }
1206 }

```

(End of definition for `\__scontents_verbatimsc_instance:`.)

verbatimsc Defines the environment `verbatimsc` for L^AT_EX. The `\endverbatim` in the second argument is only needed for compatibility with the `verbatim[15]` package.

```

1207 \NewDocumentEnvironment { verbatimsc } { { }
1208 {
1209   \IfDocumentMetadataTF
1210   {
1211     \UseInstance{blockenv}{verbatimsc}{}
1212     \@setupverbinvisiblespace\frenchspacing\@vobeyspaces
1213     \__scontents_xverb:
1214   }
1215   {
1216     \cs_set_eq:cN { @xverbatim } \__scontents_xverb:
1217     \verbatim
1218   }
1219 }
1220 {
1221   \IfDocumentMetadataTF
1222   {
1223     \BlockEnvEnd
1224   }
1225   { \endverbatim }
1226 }
1227 (/latex)
1228 (/loader)

```

(End of definition for `verbatimsc`. This function is documented on page 8.)

13.16 The command `\Scontents`

The user-level command `\Scontents` stores its $\{\langle argument \rangle\}$ in a *sequence*. It takes an optional *starred argument*, an optional argument with the $\langle keys \rangle$ and the $\{\langle argument \rangle\}$, which is read in standard mode or, with the *starred argument*, under the *verbatim catcode* regimen. It is adapted from code by Ulrich Diez in [Stringify input - \string on token list](#) and code by user @siracusa in [Convert a macro from Latex2e to expl3](#).

13.16.1 Definition of variables for `\Scontents`

`\l__scontents_Scontents_arg_tl` The token list `\l__scontents_Scontents_arg_tl` saves the $\{\langle argument \rangle\}$ passed to `\Scontents`.

```

1229 (*core)
1230 \tl_new:N \l__scontents_Scontents_arg_tl

```

(End of definition for `\l__scontents_Scontents_arg_tl`.)

13.16.2 Definition of messages for `\Scontents`

`empty-argument-command` The “warning” `empty-argument-command` is issued when the $\{\langle argument \rangle\}$ of `\Scontents` is empty.

```

1231 \msg_new:nnn { scontents } { empty-argument-command }
1232 { Empty~argument~for~\c_backslash_str Scontents~\msg_line_context:. }

```

(End of definition for `empty-argument-command`.)

13.16.3 Definition of keys for `\Scontents`

We define the *keys* for both versions of `\Scontents`, the unknown *keys* are passed to `\__scontents_unknown_keys_cmd:n` (§13.9).

```

1233 \keys_define:nn { scontents / Scontents }
1234 {
1235   write-cmd .code:n          = { \__scontents_set_write_store:n {#1} },
1236   write-out .code:n          = { \__scontents_set_write_out:n {#1} },
1237   write-cmd .value_required:n = true,
1238   write-out .value_required:n = true,
1239   print-cmd .meta:nn         = { scontents } { print-cmd = #1 },
1240   print-cmd .default:n       = true,
1241   store-cmd .meta:nn         = { scontents } { store-cmd = #1 },
1242   force-eol .meta:nn         = { scontents } { force-eol = #1 },
1243   force-eol .default:n       = true,
1244   overwrite .meta:nn         = { scontents } { overwrite = #1 },
1245   overwrite .default:n       = true,
1246   unknown .code:n            = { \__scontents_unknown_keys_cmd:n {#1} },
1247 }

```

(End of definition for `write-cmd` and others.)

13.16.4 Internal functions for `\Scontents`

`\__scontents_Scontents_code:nn` The function `\__scontents_Scontents_code:nn` executes `\__scontents_bsphack:`, opens a group, sets the *keys* passed in *#2* and leaves the TAB character active. With the *starred argument* passed in *#1* it calls `\__scontents_Scontents_verb_arg:w`, otherwise `\__scontents_Scontents_norm_arg:n`.

```

1248 \cs_new_protected:Npn \__scontents_Scontents_code:nn #1 #2
1249 {
1250   \__scontents_bsphack:
1251   \group_begin:
1252     \keys_set:nn { scontents / Scontents } {#2}
1253     \char_set_catcode_active:n { 9 }
1254     \bool_if:NTF #1
1255       { \__scontents_Scontents_verb_arg:w }
1256       { \__scontents_Scontents_norm_arg:n }
1257 }

```

(End of definition for `\__scontents_Scontents_code:nn`.)

`\__scontents_Scontents_verb_arg:w` The function `\__scontents_Scontents_verb_arg:w` saves the *{argument}* under the *verbatim catcode* regimen, using a v-type argument of `ltcmd`, in `\__scontents_Scontents_arg_tl` and replaces every ‘`^^M`’ and `\obeyedline`, added in L^AT_EX release 2024-06-01, by ‘`^^J`’. The command `\obeyedline` is redefined with `\RenewDocumentCommand` because it can be modified by the user, which would return a low-level error.

```

1258 \NewDocumentCommand \__scontents_Scontents_verb_arg:w { +v }
1259 {
1260   \tl_set:Nn \__scontents_Scontents_arg_tl {#1}
1261   \cs_if_exist:NT \obeyedline
1262   {
1263     \RenewDocumentCommand \obeyedline { } { \iow_char:N ^^M }
1264     \tl_replace_all:Nee \__scontents_Scontents_arg_tl
1265       { \obeyedline } { \iow_char:N ^^M }
1266   }
1267   \tl_replace_all:Nev \__scontents_Scontents_arg_tl
1268     { \iow_char:N ^^M } { \c__scontents_newline_tl }
1269   \__scontents_Scontents_finish:
1270 }

```

(End of definition for `\__scontents_Scontents_verb_arg:w`.)

`\__scontents_Scontents_norm_arg:n` The function `\__scontents_Scontents_norm_arg:n` saves the *{argument}* read in the *standard catcode* regimen.

```

1271 \cs_new_protected:Npn \__scontents_Scontents_norm_arg:n #1
1272 {
1273   \tl_set:Nn \__scontents_Scontents_arg_tl {#1}
1274   \__scontents_Scontents_finish:
1275 }

```

(End of definition for `\__scontents_Scontents_norm_arg:n`.)

`\__scontents_Scontents_finish:` The function `\__scontents_Scontents_finish:` issues the “warning” `empty-argument-command` if the $\langle \textit{argument} \rangle$ is empty, writes it with `\__scontents_file_write_cmd:VV` for the `write-out` and `write-cmd` keys and stores and prints it with `\__scontents_finish_storing:NNN`. Finally it closes the group and runs `\__scontents_esphack:` if the `print-cmd` key is not active.

```

1276 \cs_new_protected:Nn \__scontents_Scontents_finish:
1277 {
1278   \tl_if_empty:NT \__scontents_Scontents_arg_tl
1279   { \msg_warning:nn { scontents } { empty-argument-command } }
1280   \__scontents_file_write_cmd:VV \__scontents_file_name_tl \__scontents_Scontents_arg_tl
1281   \__scontents_finish_storing:NNN
1282   \__scontents_Scontents_arg_tl \__scontents_name_seq_cmd_tl \__scontents_print_cmd_bool
1283   \use:e
1284   {
1285     \group_end:
1286     \bool_if:NF \__scontents_print_cmd_bool { \__scontents_esphack: }
1287   }
1288 }

```

(End of definition for `\__scontents_Scontents_finish:`.)

`\Scontents` Defines the user command `\Scontents`.

```

1289 \NewDocumentCommand \Scontents { !s !O{} }
1290 {
1291   \tl_set:Nn \__scontents_cmd_name_tl { Scontents }
1292   \__scontents_Scontents_code:nn {#1} {#2}
1293 }

```

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

13.17 The command `\getstored`

The user-level command `\getstored` takes an optional argument with the $\langle \textit{index} \rangle$, the last $\langle \textit{stored content} \rangle$ by default, and a mandatory argument with the $\langle \textit{seq name} \rangle$. It is robust.

13.17.1 Internal functions for `\getstored`

`\__scontents_getstored:nn` The function `\__scontents_getstored:nn` rescans with `\tl_retokenize:e` the $\langle \textit{stored content} \rangle$ at the $\langle \textit{index} \rangle$ passed in `#1` in the *sequence* named by `#2`, given by `\__scontents_getfrom_seq:nn` (§13.11), unless it is not available yet in a remembered *sequence* (§13.12).

```

1294 \cs_new_protected:Npn \__scontents_getstored:nn #1 #2
1295 {
1296   \__scontents_if_pending:neF {#2} { \int_eval:n {#1} }
1297   {
1298     \tl_retokenize:e
1299     {
1300       \__scontents_getfrom_seq:nn {#1} {#2}
1301     }
1302   }
1303 }

```

(End of definition for `\__scontents_getstored:nn`.)

`\getstored` Defines the user command `\getstored`.

```

1304 \NewDocumentCommand \getstored { O{-1} m }
1305 {
1306   \__scontents_getstored:nn {#1} {#2}
1307 }

```

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

13.18 The command `\foreachsc`

The user-level command `\foreachsc` runs `\getstored` on the $\langle \textit{stored contents} \rangle$ of a *sequence*. It takes an optional argument with the $\langle \textit{keys} \rangle$ and a mandatory argument with the $\langle \textit{seq name} \rangle$, raising the “error” `undefined-storage` if the *sequence* does not exist.

13.18.1 Definition of variables for \foreachsc

The token list `\l__scontents_foreachsc_arg_tl` saves the $\langle\{seq\ name\}\rangle$, the sequence `\l__scontents_foreach_print_seq` collects the code for each $\langle\{stored\ content\}\rangle$, built in the token list `\l__scontents_foreach_body_tl`, and the token list `\g__scontents_foreach_exec_tl` executes it outside the group. The token list `\l__scontents_foreach_cmd_tl` saves the command executed on each $\langle\{stored\ content\}\rangle$, `\getstored` by default. The other variables are set by the $\langle\{keys\}\rangle$ of `\foreachsc`.

```

\l__scontents_foreachsc_arg_tl
\l__scontents_foreach_print_seq
\g__scontents_foreach_exec_tl
\l__scontents_foreach_before_tl
\l__scontents_foreach_before_bool
\l__scontents_foreach_after_tl
\l__scontents_foreach_after_bool
\l__scontents_foreach_start_int
\l__scontents_foreach_step_int
\l__scontents_foreach_stop_int
\l__scontents_foreach_stop_bool
\l__scontents_foreach_wrapper_bool
\l__scontents_foreach_body_tl
\l__scontents_foreach_cmd_tl
1308 \tl_new:N \l__scontents_foreachsc_arg_tl
1309 \seq_new:N \l__scontents_foreach_print_seq
1310 \tl_new:N \g__scontents_foreach_exec_tl
1311 \tl_new:N \l__scontents_foreach_before_tl
1312 \bool_new:N \l__scontents_foreach_before_bool
1313 \tl_new:N \l__scontents_foreach_after_tl
1314 \bool_new:N \l__scontents_foreach_after_bool
1315 \int_new:N \l__scontents_foreach_start_int
1316 \int_new:N \l__scontents_foreach_step_int
1317 \int_new:N \l__scontents_foreach_stop_int
1318 \bool_new:N \l__scontents_foreach_stop_bool
1319 \bool_new:N \l__scontents_foreach_wrapper_bool
1320 \tl_new:N \l__scontents_foreach_body_tl
1321 \tl_new:N \l__scontents_foreach_cmd_tl
1322 \tl_set:Nn \l__scontents_foreach_cmd_tl { \getstored }

```

(End of definition for `\l__scontents_foreachsc_arg_tl` and others.)

13.18.2 Definition of keys for \foreachsc

We define the $\langle\{keys\}\rangle$ for `\foreachsc`, the keys `start`, `step` and `stop` use `\__scontents_set_positive_int:Nnn` (§13.8) and the unknown $\langle\{keys\}\rangle$ are passed to `\__scontents_unknown_keys_cmd:n` (§13.9).

```

before
after
start
stop
step
wrapper
sep
unknown
1323 \keys_define:nn { scontents / foreachsc }
1324 {
1325   before .code:n = {
1326     \bool_set_true:N \l__scontents_foreach_before_bool
1327     \tl_set:Nn \l__scontents_foreach_before_tl { #1 }
1328   },
1329   before .value_required:n = true,
1330   after .code:n = {
1331     \bool_set_true:N \l__scontents_foreach_after_bool
1332     \tl_set:Nn \l__scontents_foreach_after_tl { #1 }
1333   },
1334   after .value_required:n = true,
1335   start .code:n = {
1336     \__scontents_set_positive_int:Nnn
1337     \l__scontents_foreach_start_int { start } { #1 }
1338   },
1339   start .value_required:n = true,
1340   start .initial:n = 1,
1341   stop .code:n = {
1342     \__scontents_set_positive_int:Nnn
1343     \l__scontents_foreach_stop_int { stop } { #1 }
1344     \bool_set:Nn \l__scontents_foreach_stop_bool
1345     {
1346       \int_compare_p:nNn
1347       { \l__scontents_foreach_stop_int } > { 0 }
1348     }
1349   },
1350   stop .value_required:n = true,
1351   step .code:n = {
1352     \__scontents_set_positive_int:Nnn
1353     \l__scontents_foreach_step_int { step } { #1 }
1354   },
1355   step .value_required:n = true,
1356   step .initial:n = 1,
1357   wrapper .code:n = {
1358     \bool_set_true:N \l__scontents_foreach_wrapper_bool
1359     \cs_set_protected:Npn
1360     \__scontents_foreach_wrapper:n ##1 { #1 }
1361   },
1362   wrapper .value_required:n = true,
1363   sep .tl_set:N = \l__scontents_foreach_sep_tl,

```

```

1364     sep      .initial:n      = { },
1365     sep      .value_required:n = true,
1366     unknown  .code:n          = { \__scontents_unknown_keys_cmd:n {#1} },
1367   }

```

(End of definition for *before* and *others*.)

getstored
tpestored
meaningsc

The *(keys)* getstored, tpestored and meaningsc set the command saved in \l__scontents_foreach_cmd_tl.

```

1368 \keys_define:nn { scontents / foreachsc }
1369 {
1370   getstored .code:n      =
1371     { \tl_set:Nn \l__scontents_foreach_cmd_tl { \getstored } },
1372   getstored .value_forbidden:n = true,
1373   tpestored .code:n      =
1374     { \tl_set:Nn \l__scontents_foreach_cmd_tl { \tpestored } },
1375   tpestored .value_forbidden:n = true,
1376   meaningsc .code:n      =
1377     { \tl_set:Nn \l__scontents_foreach_cmd_tl { \meaningsc } },
1378   meaningsc .value_forbidden:n = true,
1379 }

```

(End of definition for *getstored*, *tpestored*, and *meaningsc*.)

13.18.3 Internal functions for \foreachsc

__scontents_foreachsc:nn
 __scontents_foreach_add_body:n
__scontents_foreach_wrap:n
__scontents_foreach_wrap:V

The function __scontents_foreachsc:nn sets the *(keys)* passed in #1 and steps from *start* to *stop*, the number of *(stored contents)* by default, with __scontents_foreach_add_body:n, which adds the command saved in \l__scontents_foreach_cmd_tl, with the code of the *before*, *after* and *wrapper* keys, to \l__scontents_foreach_print_seq. The result, separated by *sep*, is executed after the group. The function __scontents_foreach_wrap:n expands __scontents_foreach_wrapper:n once inside the group, so the code of the *wrapper* key is already in place when the result is executed; the function __scontents_foreach_wrapper:n is local and no longer exists after the group.

```

1380 \cs_new_protected:Npn \__scontents_foreachsc:nn #1 #2
1381 {
1382   \group_begin:
1383     \tl_if_novalue:nF {#1} { \keys_set:nn { scontents / foreachsc } {#1} }
1384     \tl_set:Nn \l__scontents_foreachsc_arg_tl {#2}
1385     \seq_clear:N \l__scontents_foreach_print_seq
1386     \bool_if:NF \l__scontents_foreach_stop_bool
1387     {
1388       \int_set:Nn \l__scontents_foreach_stop_int
1389         { \__scontents_seq_count:n {#2} }
1390     }
1391     \int_step_function:nnnN
1392     { \l__scontents_foreach_start_int }
1393     { \l__scontents_foreach_step_int }
1394     { \l__scontents_foreach_stop_int }
1395     \__scontents_foreach_add_body:n
1396     \tl_gset:Ne \g__scontents_foreach_exec_tl
1397     {
1398       \exp_args:NNV \seq_use:Nn
1399       \l__scontents_foreach_print_seq \l__scontents_foreach_sep_tl
1400     }
1401     \group_end:
1402     \exp_after:w \tl_gclear:N
1403     \exp_after:w \g__scontents_foreach_exec_tl
1404     \g__scontents_foreach_exec_tl
1405   }
1406 \cs_new_protected:Npn \__scontents_foreach_add_body:n #1
1407 {
1408   \tl_set:Ne \l__scontents_foreach_body_tl
1409   {
1410     \exp_not:V \l__scontents_foreach_cmd_tl [#1]
1411     { \exp_not:V \l__scontents_foreachsc_arg_tl }
1412   }
1413   \bool_if:NT \l__scontents_foreach_wrapper_bool
1414   { \__scontents_foreach_wrap:V \l__scontents_foreach_body_tl }
1415   \seq_put_right:Ne \l__scontents_foreach_print_seq
1416   {

```

```

1417     \bool_if:NT \l__scontents_foreach_before_bool
1418     { \exp_not:V \l__scontents_foreach_before_tl }
1419     \exp_not:V \l__scontents_foreach_body_tl
1420     \bool_if:NT \l__scontents_foreach_after_bool
1421     { \exp_not:V \l__scontents_foreach_after_tl }
1422   }
1423 }
1424 \cs_new_protected:Npn \__scontents_foreach_wrap:n #1
1425 { \tl_set:Nn \l__scontents_foreach_body_tl { \__scontents_foreach_wrapper:n {#1} } }
1426 \cs_generate_variant:Nn \__scontents_foreach_wrap:n { V }

```

(End of definition for `\__scontents_foreachsc:nn`, `\__scontents_foreach_add_body:n`, and `\__scontents_foreach_wrap:n`.)

`\foreachsc` Defines the user command `\foreachsc`.

```

1427 \NewDocumentCommand \foreachsc { +o m }
1428 {
1429   \tl_set:Nn \l__scontents_cmd_name_tl { foreachsc }
1430   \__scontents_seq_if_exist:nTF {#2}
1431   { \__scontents_foreachsc:nn {#1} {#2} }
1432   { \msg_error:nnn { scontents } { undefined-storage } {#2} }
1433 }

```

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

13.19 The optional argument of `\typestored`, `\meaningsc` and `\mergesc`

The commands `\typestored` and `\meaningsc`, and `\mergesc` which uses them internally, accept a single *optional argument* with the $\langle index \rangle$, $\langle 1-end \rangle$ or the range $\langle start-stop \rangle$ of the $\langle stored\ contents \rangle$ in the *sequence* along with the $\langle keys \rangle$, this avoids the awkward `\typestored[...][\langle keys \rangle]{...}` syntax. The $\langle indexes \rangle$ and ranges are processed as unknown $\langle keys \rangle$ of `\l3keys[16, 17]`.

13.19.1 Definition of variables for the optional argument

The string variable `\l__scontents_current_seq_name_str` stores the name of the *sequence* passed to `\typestored` or `\meaningsc`, it is used by `\__scontents_parse_type_meaning_key:nn`. The string variable `\l__scontents_index_str` is used to check that an $\langle index \rangle$ only has digits and the token lists `\l__scontents_range_start_tl` and `\l__scontents_range_stop_tl` save the limits of a range $\langle start-stop \rangle$.

```

1434 \str_new:N \l__scontents_current_seq_name_str
1435 \str_new:N \l__scontents_index_str
1436 \tl_new:N \l__scontents_range_start_tl
1437 \tl_new:N \l__scontents_range_stop_tl

```

(End of definition for `\l__scontents_current_seq_name_str` and others.)

13.19.2 Definition of keys for the optional argument

We define the common $\langle keys \rangle$ for `\typestored` and `\meaningsc`. The `write-out`, `overwrite` and `print-cmd` keys are necessary for the implementation of `\mergesc` (§13.22) and the unknown $\langle keys \rangle$ are passed to `\__scontents_parse_type_meaning_key:n`.

```

width-tab
write-out
overwrite
print-cmd
unknown
1438 \keys_define:nn { scontents / typemeaning }
1439 {
1440   width-tab .meta:nn      = { scontents } { width-tab = #1 },
1441   write-out .code:n       = { \__scontents_set_write_out:n {#1} },
1442   write-out .value_required:n = true,
1443   overwrite .meta:nn     = { scontents } { overwrite = #1 },
1444   overwrite .default:n    = true,
1445   print-cmd .bool_set:N   = \l__scontents_print_verb_style_bool,
1446   print-cmd .initial:n    = true,
1447   print-cmd .default:n    = true,
1448   unknown .code:n        = { \__scontents_parse_type_meaning_key:n {#1} },
1449 }

```

(End of definition for `width-tab` and others.)

13.19.3 Internal functions for the optional argument

The function `\__scontents_parse_type_meaning_key:n` checks that the unknown $\langle key \rangle$ has no value, otherwise something odd as `1=1` was used and the “error” `cmd-key-value-unknown` is raised. The conditional `\__scontents_if_digits:nTF` tests if `#1` is not blank and only has digits, it removes

the digits from `\l__scontents_index_str`, so a text that is not a number never reaches an *integer expression* and only the “error” of the package is raised.

The function `\__scontents_range_parser:nnn` receives the *<index>* or the range in #1, the number of *<stored contents>* in #2 and the code that raises the “error” in #3, and splits #1 at the hyphens with `\__scontents_range_parser:nnnw`: without hyphen it is an *<index>*, with a hyphen at the beginning it is a negative *<index>* and otherwise it is a range *<start-stop>*, where `\__scontents_range_limit:nn` replaces *end* by the number of *<stored contents>* and `\__scontents_range_step:nn` adds the *<indexes>* of the range, ascending or descending, to `\l__scontents_seq_item_seq`. Anything else is passed to the code in #3.

```

1450 \cs_new_protected:Npn \__scontents_parse_type_meaning_key:n #1
1451 { \exp_args:NV \__scontents_parse_type_meaning_key:nn \l_keys_key_str {#1} }
1452 \cs_new_protected:Npn \__scontents_parse_type_meaning_key:nn #1 #2
1453 {
1454   \tl_if_blank:nTF {#2}
1455   {
1456     \__scontents_range_parser:nnn {#1}
1457     { \__scontents_seq_count:n { \l__scontents_current_seq_name_str } }
1458     { \msg_error:nnn { scontents } { cmd-key-unknown } }
1459   }
1460   { \msg_error:nnnn { scontents } { cmd-key-value-unknown } {#1} {#2} }
1461 }
1462 \prg_new_protected_conditional:Npnn \__scontents_if_digits:n #1 { TF }
1463 {
1464   \str_set:Ne \l__scontents_index_str { \tl_trim_spaces:n {#1} }
1465   \clist_map_inline:nn { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }
1466   { \str_remove_all:Nn \l__scontents_index_str {##1} }
1467   \tl_if_blank:nTF {#1}
1468   { \prg_return_false: }
1469   {
1470     \str_if_empty:NTF \l__scontents_index_str
1471     { \prg_return_true: }
1472     { \prg_return_false: }
1473   }
1474 }
1475 \prg_generate_conditional_variant:Nnn \__scontents_if_digits:n { V } { TF }
1476 \cs_new_protected:Npn \__scontents_range_parser:nnn #1 #2 #3
1477 { \__scontents_range_parser:nnnw {#1} {#2} {#3} #1 - \q__scontents_mark - \s__scontents_mark }
1478 \cs_new_protected:Npn \__scontents_range_parser:nnnw #1 #2 #3 #4 - #5 - #6 \s__scontents_mark
1479 {
1480   \__scontents_tl_if_head_is_q_mark:nTF {#5}
1481   {
1482     \__scontents_if_digits:nTF {#4}
1483     { \seq_put_right:Ne \l__scontents_seq_item_seq { \int_eval:n {#4} } }
1484     { #3 {#1} }
1485   }
1486   {
1487     \__scontents_tl_if_head_is_q_mark:nTF {#6}
1488     {
1489       \tl_if_blank:nTF {#4}
1490       {
1491         \__scontents_if_digits:nTF {#5}
1492         {
1493           \seq_put_right:Ne \l__scontents_seq_item_seq
1494           { \int_eval:n { - #5 } }
1495         }
1496         { #3 {#1} }
1497       }
1498       {
1499         \tl_set:Ne \l__scontents_range_start_tl
1500         { \__scontents_range_limit:nn {#4} {#2} }
1501         \tl_set:Ne \l__scontents_range_stop_tl
1502         { \__scontents_range_limit:nn {#5} {#2} }
1503         \__scontents_if_digits:VTF \l__scontents_range_start_tl
1504         {
1505           \__scontents_if_digits:VTF \l__scontents_range_stop_tl
1506           {
1507             \exp_args:NVV \__scontents_range_step:nn
1508             \l__scontents_range_start_tl \l__scontents_range_stop_tl
1509           }
1510           { #3 {#1} }

```

```

1511         }
1512         { #3 {#1} }
1513     }
1514 }
1515 { #3 {#1} }
1516 }
1517 }
1518 \cs_new:Npn \__scontents_range_limit:nn #1 #2
1519 {
1520     \str_if_eq:eeTF { \tl_trim_spaces:n {#1} } { end }
1521     {#2}
1522     { \tl_trim_spaces:n {#1} }
1523 }
1524 \cs_new_protected:Npn \__scontents_range_step:nn #1 #2
1525 {
1526     \int_step_inline:nnnn {#1}
1527     { \int_compare:nNnTF {#2} < {#1} { -1 } { 1 } }
1528     {#2}
1529     { \seq_put_right:Nn \l__scontents_seq_item_seq {##1} }
1530 }

```

(End of definition for `\__scontents_parse_type_meaning_key:n` and others.)

```

\__scontents_typemeaning_setup:nn
\l__scontents_pending_bool

```

The function `\__scontents_typemeaning_setup:nn` is the common start of `\typestored` and `\meaningsc`, it saves the name of the *sequence* passed in `#2`, sets the *⟨keys⟩* and the *⟨indexes⟩* passed in `#1` and uses the first *⟨stored content⟩* if no *⟨index⟩* was given. The boolean `\l__scontents_pending_bool` is true if any of them is not available yet in a remembered *sequence* (§13.12).

```

1531 \bool_new:N \l__scontents_pending_bool
1532 \cs_new_protected:Npn \__scontents_typemeaning_setup:nn #1 #2
1533 {
1534     \seq_clear:N \l__scontents_seq_item_seq
1535     \str_set:Ne \l__scontents_current_seq_name_str {#2}
1536     \tl_if_novalue:nF {#1}
1537     { \keys_set:nn { scontents / typemeaning } {#1} }
1538     \seq_if_empty:NT \l__scontents_seq_item_seq
1539     { \seq_set_from_clist:Nn \l__scontents_seq_item_seq { 1 } }
1540     \bool_set_false:N \l__scontents_pending_bool
1541     \seq_map_inline:Nn \l__scontents_seq_item_seq
1542     {
1543         \__scontents_if_pending:nnT {#2} {##1}
1544         { \bool_set_true:N \l__scontents_pending_bool }
1545     }
1546 }

```

(End of definition for `\__scontents_typemeaning_setup:nn` and `\l__scontents_pending_bool`.)

13.20 The command `\typestored`

The user-level command `\typestored` prints the *⟨stored contents⟩* of a *sequence* in the `verbatimsc` environment (§13.15). It takes an optional argument (§13.19) and a mandatory argument with the *⟨seq name⟩*.

13.20.1 Definition of variables for `\typestored`

```
\l__scontents_typestored_arg_tl
```

The token list `\l__scontents_typestored_arg_tl` saves the *⟨stored contents⟩* to print.

```
1547 \tl_new:N \l__scontents_typestored_arg_tl
```

(End of definition for `\l__scontents_typestored_arg_tl`.)

13.20.2 Definition of error messages for `\typestored`

```
empty-variable
```

The “error” `empty-variable` is raised when the *⟨stored contents⟩* to print are blank.

```

1548 \msg_new:nnnn { scontents } { empty-variable }
1549 { Blank~stored~content~to~print~\msg_line_context:. }
1550 {
1551     The~stored~content~only~has~spaces,~so~there~is~nothing~to~print~
1552     in~verbatim~style.~Check~the~stored~content~or~the~index.
1553 }

```

(End of definition for `empty-variable`.)

13.20.3 Internal functions for \typestored

`\__scontents_tpestored:nn`

The function `\__scontents_tpestored:nn` sets the *⟨keys⟩* and the *⟨indexes⟩* passed in *#1* with `\__scontents_tpemeaning_setup:nn`, gets the *⟨stored contents⟩* of the *sequence* named by *#2*, writes them to the `.log` file if `debug-var` is true and prints them with `\__scontents_tpestored:N` and writes them with `\__scontents_file_write_cmd:VV`.

```

1554 \cs_new_protected:Npn \__scontents_tpestored:nn #1 #2
1555 {
1556   \__scontents_bsphack:
1557   \group_begin:
1558     \__scontents_tpemeaning_setup:nn {#1} {#2}
1559     \bool_if:NF \__scontents_pending_bool
1560     {
1561       \tl_set:Nc \__scontents_tpestored_arg_tl
1562         { \__scontents_getfrom_seq:Nn \__scontents_seq_item_seq {#2} }
1563       \__scontents_remove_trailing_eol:N \__scontents_tpestored_arg_tl
1564       \tl_replace_all:NcV \__scontents_tpestored_arg_tl
1565         { \__scontents_hidden_space_str } { \__scontents_newline_tl
1566       \bool_if:NT \__scontents_debug_var_bool
1567         {
1568           \tl_log:N \__scontents_tpestored_arg_tl
1569         }
1570       \tl_if_empty:NF \__scontents_tpestored_arg_tl
1571         {
1572           \bool_if:NT \__scontents_print_verb_style_bool
1573             {
1574               \__scontents_tpestored:N \__scontents_tpestored_arg_tl
1575             }
1576         }
1577       \__scontents_file_write_cmd:VV \__scontents_file_name_tl \__scontents_tpestored_arg_tl
1578     }
1579     \use:e
1580     {
1581       \group_end:
1582       \bool_if:NF \__scontents_print_verb_style_bool { \__scontents_esphack: }
1583     }
1584 }

```

(End of definition for `\__scontents_tpestored:nn`)

`\__scontents_tpestored:N`
`\__scontents_verb_print_EOL:`

The function `\__scontents_tpestored:N` is defined with active carriage return characters to mimic an actual verbatim environment “on the loose”. The content of the token list passed in *#1* is placed in a `verbatimsc` environment and rescanned with `\tl_retokenize:e`.

```

1585 \group_begin:
1586   \char_set_catcode_active:N \^^M
1587   \cs_new_protected:Npn \__scontents_tpestored:N #1
1588   {
1589     \tl_if_blank:VT #1
1590     { \msg_error:nn { scontents } { empty-variable } }
1591     \cs_set_eq:NN \__scontents_verb_print_EOL: \^^M
1592     \cs_set_eq:NN \^^M \scan_stop:
1593     \cs_set_eq:cN { do@noligs } \__scontents_do_noligs:N
1594     \int_set:Nn \tex_newlinechar:D { '\^^J }
1595     \tl_retokenize:e
1596     {
1597       \__scontents_format_case:nnn
1598       { \exp_not:N \begin{verbatimsc} } % LaTeX
1599       { \verbatimsc } % Plain/Generic
1600       { \startverbatimsc } % ConTeXt
1601       \^^M
1602       \exp_not:V #1 \^^M
1603       \g__scontents_end_verbatimsc_tl
1604     }
1605     \cs_set_eq:NN \^^M \__scontents_verb_print_EOL:
1606   }
1607 \group_end:

```

(End of definition for `\__scontents_tpestored:N` and `\__scontents_verb_print_EOL:.`)

`\typestored` Defines the user command `\typestored`.

```

1608 \NewDocumentCommand \typestored { o m }

```

```

1609 {
1610   \tl_set:Nn \l__scontents_cmd_name_tl { typestored }
1611   \__scontents_typestored:nn {#1} {#2}
1612 }

```

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

13.21 The command `\meaningsc`

The user-level command `\meaningsc` prints the meaning of the *⟨stored contents⟩* of a *sequence*. It takes the same arguments as `\typestored` (§13.20).

13.21.1 Definition of variables for `\meaningsc`

`\l__scontents_meaningsc_arg_tl` The token list `\l__scontents_meaningsc_arg_tl` saves the *⟨stored contents⟩* to print.

```

1613 \tl_new:N \l__scontents_meaningsc_arg_tl

```

(End of definition for `\l__scontents_meaningsc_arg_tl`.)

13.21.2 Internal functions for `\meaningsc`

`\__scontents_meaningsc:nn` The function `\__scontents_meaningsc:nn` sets the *⟨keys⟩* and the *⟨indexes⟩* passed in `#1` with `\__scontents_typemeaning_setup:nn` and calls `\__scontents_meaningsc:n` with the *sequence* named by `#2`.

```

1614 \cs_new_protected:Npn \__scontents_meaningsc:nn #1 #2
1615 {
1616   \__scontents_bsphack:
1617   \group_begin:
1618   \__scontents_typemeaning_setup:nn {#1} {#2}
1619   \bool_if:NF \l__scontents_pending_bool { \__scontents_meaningsc:n {#2} }
1620   \use:e
1621   {
1622     \group_end:
1623     \bool_if:NF \l__scontents_print_verb_style_bool { \__scontents_esphack: }
1624   }
1625 }

```

(End of definition for `\__scontents_meaningsc:nn`.)

`\__scontents_meaningsc:n` The function `\__scontents_meaningsc:n` is defined with the TAB character active. It gets the *⟨stored contents⟩*, replaces the ends of line and `\c__scontents_hidden_space_str` by spaces and each TAB character, active or read from a remembered *sequence* (§13.12), by as many spaces as the `width-tab` key was set to, prints them with `\cs_replacement_spec:N` and writes them with `\__scontents_file_write_cmd:VV`.

```

1626 \group_begin:
1627 \char_set_catcode_active:N ^^I
1628 \cs_new_protected:Npn \__scontents_meaningsc:n #1
1629 {
1630   \tl_set:Ne \l__scontents_meaningsc_arg_tl
1631   { \__scontents_getfrom_seq:Nn \l__scontents_seq_item_seq {#1} }
1632   \tl_replace_all:Nen \l__scontents_meaningsc_arg_tl
1633   { \iow_char:N ^^J } { ~ }
1634   \tl_replace_all:Nen \l__scontents_meaningsc_arg_tl
1635   { \c__scontents_hidden_space_str } { ~ }
1636   \bool_if:NT \l__scontents_debug_var_bool
1637   {
1638     \tl_log:N \l__scontents_meaningsc_arg_tl
1639   }
1640   \tl_use:N \l__scontents_verb_font_tl
1641   \tl_replace_all:Nne \l__scontents_meaningsc_arg_tl
1642   { ^^I } { \prg_replicate:nn { \l__scontents_tab_width_int } { ~ } }
1643   \tl_replace_all:Nee \l__scontents_meaningsc_arg_tl
1644   { \iow_char:N ^^I }
1645   { \prg_replicate:nn { \l__scontents_tab_width_int } { ~ } }
1646   \tl_if_empty:NF \l__scontents_meaningsc_arg_tl
1647   {
1648     \bool_if:NT \l__scontents_print_verb_style_bool
1649     {
1650       \cs_replacement_spec:N \l__scontents_meaningsc_arg_tl
1651     }
1652   }

```

```

1653     \__scontents_file_write_cmd:VV \__scontents_file_name_tl \__scontents_meaningsc_arg_tl
1654   }
1655 \group_end:

```

(End of definition for __scontents_meaningsc:n.)

\meaningsc Defines the user command **\meaningsc**.

```

1656 \NewDocumentCommand \meaningsc { o m }
1657 {
1658   \tl_set:Nn \__scontents_cmd_name_tl { meaningsc }
1659   \__scontents_meaningsc:nn {#1} {#2}
1660 }

```

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

13.22 The command \mergesc

The user-level command **\mergesc** takes an optional argument with the *⟨keys⟩* and a mandatory argument with a *comma separated list* of *⟨seq name⟩* followed by an optional *⟨index⟩* or range. It assembles the *⟨stored contents⟩* in a temporary *internal sequence* and passes it to **\typestored** or **\meaningsc**, or rescans it as **\getstored** does.

13.22.1 Definition of variables for \mergesc

The token list **\l__scontents_mergesc_arg_tl** saves the last *⟨stored content⟩* assembled and the token list **\l__scontents_mergesc_keys_tl** saves the *⟨keys⟩* passed to **\typestored** or **\meaningsc**.

```

1661 \tl_new:N \l__scontents_mergesc_arg_tl
1662 \tl_new:N \l__scontents_mergesc_keys_tl

```

(End of definition for \l__scontents_mergesc_arg_tl and \l__scontents_mergesc_keys_tl.)

13.22.2 Definition of error messages for \mergesc

mergesc-missing-key The “error” **mergesc-missing-key** is raised when none of the **\typestored**, **\meaningsc** and **\getstored** keys is passed to **\mergesc**.

```

1663 \msg_new:nnnn { scontents } { mergesc-missing-key }
1664 {
1665   Missing~mandatory~key~'typestored',~'meaningsc'~or~'getstored'~
1666   for~\c_backslash_str mergesc~\msg_line_context:.
1667 }
1668 {
1669   The~command~\c_backslash_str mergesc~needs~one~of~the~keys~
1670   'typestored',~'meaningsc'~or~'getstored'.\\
1671   Check~that~you~have~spelled~the~key~name~correctly.
1672 }

```

(End of definition for mergesc-missing-key.)

13.22.3 Definition of keys for \mergesc

The *⟨keys⟩* **\typestored**, **\meaningsc** and **\getstored** are *mandatory*, they set the function **__scontents_mergesc_cmd:nn**, which by default raises the “error” **mergesc-missing-key**.

```

1673 \keys_define:nn { scontents / mergesc }
1674 {
1675   typestored .code:n =
1676   { \cs_set_eq:NN \__scontents_mergesc_cmd:nn \__scontents_typestored:nn },
1677   typestored .value_forbidden:n = true,
1678   meanings .code:n =
1679   { \cs_set_eq:NN \__scontents_mergesc_cmd:nn \__scontents_meaningsc:nn },
1680   meanings .value_forbidden:n = true,
1681   getstored .code:n =
1682   { \cs_set_eq:NN \__scontents_mergesc_cmd:nn \__scontents_mergesc_getstored:nn },
1683   getstored .value_forbidden:n = true,
1684 }
1685 \cs_new_protected:Npn \__scontents_mergesc_cmd:nn #1 #2
1686 {
1687   \msg_error:nn { scontents } { mergesc-missing-key }
1688 }

```

(End of definition for typestored and others.)

13.22.4 Internal functions for \mergesc

__scontents_mergesc_code:nn

The function `__scontents_mergesc_code:nn` sets the known *(keys)* passed in #1, saves the others in `__scontents_mergesc_keys_tl`, assembles the *internal sequence* from the list passed in #2 and, if it is not empty, passes it to `__scontents_mergesc_cmd:nn`.

```

1689 \cs_new_protected:Npn __scontents_mergesc_code:nn #1 #2
1690 {
1691   \group_begin:
1692   \tl_clear:N __scontents_mergesc_keys_tl
1693   \tl_if_novalue:nF {#1}
1694   {
1695     \keys_set_known:nnN { scontents / mergesc } {#1}
1696     __scontents_mergesc_keys_tl
1697   }
1698   \seq_gclear:c { g__scontents_name_sc!internal_seq }
1699   __scontents_mergesc_parse_list:n {#2}
1700   \seq_if_empty:cF { g__scontents_name_sc!internal_seq }
1701   {
1702     \exp_args:Ne __scontents_mergesc_cmd:nn
1703     { 1-end, \exp_not:V __scontents_mergesc_keys_tl } { sc!internal }
1704   }
1705   \group_end:
1706 }

```

(End of definition for `__scontents_mergesc_code:nn`.)

__scontents_mergesc_parse_list:n
 __scontents_mergesc_parse_item:n
 __scontents_parse_mergesc:nw
 __scontents_parse_mergesc_aux:nw
 __scontents_mergesc_put_item:nn

The function `__scontents_mergesc_parse_list:n` parses each item of the list with `__scontents_mergesc_parse_item:n`, which passes the *{(seq name)}* between braces to `__scontents_parse_mergesc:nw` when the item has no *(index)*, because the *comma separated list* removes the braces around an item. The function `__scontents_parse_mergesc:nw` raises the “error” **undefined-storage** if the *sequence* does not exist and uses all the *(stored contents)* of the *sequence* if there is no *(index)*. The function `__scontents_parse_mergesc_aux:nw` uses `__scontents_range_parser:nnn` (§13.19) and `__scontents_mergesc_put_item:nn` adds each *(stored content)* to the *internal sequence* or raises the “error” **index-out-of-range**, except for the *(stored contents)* that are not available yet in a remembered *sequence* (§13.12). Finally the end of line of the last *(stored content)* is removed with `__scontents_remove_trailing_eol:N`.

```

1707 \cs_new_protected:Npn __scontents_mergesc_parse_list:n #1
1708 {
1709   \clist_map_inline:nn {#1} { __scontents_mergesc_parse_item:n {##1} }
1710   \seq_if_empty:cF { g__scontents_name_sc!internal_seq }
1711   {
1712     \seq_gpop_right:cN { g__scontents_name_sc!internal_seq } __scontents_mergesc_arg_tl
1713     __scontents_remove_trailing_eol:N __scontents_mergesc_arg_tl
1714     \seq_gput_right:cV { g__scontents_name_sc!internal_seq } __scontents_mergesc_arg_tl
1715   }
1716 }
1717 \cs_new_protected:Npn __scontents_mergesc_parse_item:n #1
1718 {
1719   \tl_if_in:nnTF {#1} { [ ] % }
1720   { __scontents_parse_mergesc:nw #1 \s__scontents_stop }
1721   { __scontents_parse_mergesc:nw {#1} \s__scontents_stop }
1722 }
1723 \cs_new_protected:Npn __scontents_parse_mergesc:nw #1
1724 {
1725   __scontents_seq_if_exist:nTF {#1}
1726   {
1727     \peek_remove_spaces:n
1728     {
1729       \peek_charcode:NTF [ % ]
1730       { __scontents_parse_mergesc_aux:nw {#1} }
1731       { __scontents_parse_mergesc_aux:nw {#1} [ 1-end ] }
1732     }
1733   }
1734   {
1735     \msg_error:nnn { scontents } { undefined-storage } {#1}
1736     __scontents_use_none_delimit_by_s_stop:w
1737   }
1738 }
1739 \cs_new_protected:Npn __scontents_parse_mergesc_aux:nw #1 [#2]
1740 {

```

```

1741 \seq_clear:N \l__scontents_seq_item_seq
1742 \clist_map_inline:nn {#2}
1743 {
1744     \__scontents_range_parser:nnn {##1} { \__scontents_seq_count:n {#1} }
1745     { \msg_error:nnn { scontents } { cmd-key-unknown } }
1746 }
1747 \seq_map_inline:Nn \l__scontents_seq_item_seq
1748 { \__scontents_mergesc_put_item:nn {#1} {##1} }
1749 \__scontents_use_none_delimit_by_s_stop:w
1750 }
1751 \cs_new_protected:Npn \__scontents_mergesc_put_item:nn #1 #2
1752 {
1753     \__scontents_if_pending:nnF {#1} {#2}
1754     {
1755         \bool_lazy_or:nnTF
1756         { \int_compare_p:nNn {#2} = { 0 } }
1757         {
1758             \int_compare_p:nNn { \int_abs:n {#2} }
1759             > { \__scontents_seq_count:n {#1} }
1760         }
1761         {
1762             \msg_error:nneee { scontents } { index-out-of-range }
1763             {#2} {#1} { \__scontents_seq_count:n {#1} }
1764         }
1765         {
1766             \seq_gput_right:ce { g__scontents_name_sc!internal_seq }
1767             { \__scontents_seq_item:nn {#1} {#2} }
1768         }
1769     }
1770 }

```

(End of definition for `\__scontents_mergesc_parse_list:n` and others.)

`\__scontents_mergesc_getstored:nn`

The function `\__scontents_mergesc_getstored:nn`, set by the `getstored` key, joins the *stored contents* of the *internal sequence* passed in #2 in `g__scontents_last_stored_tl` separated by an end of line, adds `\c__scontents_hidden_space_str` at the end, because the last one has no end of line, and rescans them with `\tl_retokenize:V` after the group of `\mergesc`, as `\__scontents_lastfrom_seq:n` does (§13.11). The result is the same as using `\getstored` on each *stored content*. The *indexes* passed in #1 are always 1-end and the other *keys*, saved in `\l__scontents_mergesc_keys_tl`, are unknown for it and raise the “errors” of the unknown *keys* (§13.9).

```

1771 \keys_define:nn { scontents / getstored }
1772 {
1773     unknown .code:n = { \__scontents_unknown_keys_cmd:n {#1} },
1774 }
1775 \cs_new_protected:Npn \__scontents_mergesc_getstored:nn #1 #2
1776 {
1777     \keys_set:nV { scontents / getstored } \l__scontents_mergesc_keys_tl
1778     \tl_gset:Ne g__scontents_last_stored_tl
1779     {
1780         \exp_args:NnV \seq_use:cn { g__scontents_name_#2_seq } \c__scontents_newline_tl
1781         \c__scontents_hidden_space_str
1782     }
1783     \group_insert_after:N \tl_retokenize:V
1784     \group_insert_after:N g__scontents_last_stored_tl
1785     \group_insert_after:N \tl_gclear:N
1786     \group_insert_after:N g__scontents_last_stored_tl
1787 }

```

(End of definition for `\__scontents_mergesc_getstored:nn`.)

`\mergesc` Defines the user command `\mergesc`.

```

1788 \NewDocumentCommand \mergesc { o m }
1789 {
1790     \tl_set:Nn \l__scontents_cmd_name_tl { mergesc }
1791     \__scontents_mergesc_code:nn {#1} {#2}
1792 }

```

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

13.23 The command `\setupsc`

`\setupsc` Defines the user command `\setupsc`, which sets the *⟨keys⟩* of the package (§13.8).

```
1793 \NewDocumentCommand \setupsc { +m }
1794 {
1795     \tl_set:Nn \l__scontents_cmd_name_tl { setupsc }
1796     \keys_set:nn { scontents } { #1 }
1797 }
```

(End of definition for `\setupsc`. This function is documented on page 3.)

13.24 The command `\countsc`

`\countsc` Defines the user command `\countsc`, which counts the *⟨stored contents⟩* in the *sequence*. If the *sequence* does not exist, the “error” **undefined-storage** is raised and `0` is returned.

```
1798 \NewExpandableDocumentCommand \countsc { m }
1799 {
1800     \__scontents_seq_if_exist:nTF { #1 }
1801     { \__scontents_seq_count:n { #1 } }
1802     {
1803         \msg_expandable_error:nnn { scontents } { undefined-storage } { #1 }
1804         0
1805     }
1806 }
```

(End of definition for `\countsc`. This function is documented on page 8.)

13.25 The command `\cleanseqsc`

`\cleanseqsc` Defines the user command `\cleanseqsc`, which removes all the *⟨stored contents⟩* in the *sequence*.

```
1807 \NewDocumentCommand \cleanseqsc { m }
1808 {
1809     \seq_gclear_new:c { g__scontents_name_#1_seq }
1810 }
1811 </core>
```

(End of definition for `\cleanseqsc`. This function is documented on page 9.)

13.26 Programming interface

The public functions of **SCONTENTS** for `expl3` (§7), they use the functions for sequences (§13.11).

13.26.1 Definition of variables for the programming interface

`\g__scontents_map_int` The integer variable `\g__scontents_map_int` is the nesting level of `\scontents_map_inline:nn`, used in the name of the function that saves each *⟨inline function⟩*, and the token list `\l__scontents_map_item_tl` saves each *⟨stored content⟩*.

```
1812 <*core>
1813 \int_new:N \g__scontents_map_int
1814 \tl_new:N \l__scontents_map_item_tl
```

(End of definition for `\g__scontents_map_int` and `\l__scontents_map_item_tl`.)

13.26.2 Public functions of the programming interface

`\scontents_if_exist:p:n` The conditional `\scontents_if_exist:nTF` and the function `\scontents_count:n` use `\__scontents_seq_if_exist:nTF` and `\__scontents_seq_count:n`.

```
\scontents_count:n
1815 \prg_new_conditional:Npnn \scontents_if_exist:n #1 { p, T, F, TF }
1816 {
1817     \__scontents_seq_if_exist:nTF { #1 }
1818     { \prg_return_true: }
1819     { \prg_return_false: }
1820 }
1821 \cs_new:Npn \scontents_count:n #1 { \__scontents_seq_count:n { #1 } }
```

(End of definition for `\scontents_if_exist:nTF` and `\scontents_count:n`. These functions are documented on page 9.)

`\scontents_get:nnN` The function `\scontents_get:nnN` evaluates the *⟨index⟩* and passes it to `\__scontents_get:nnN`, which sets the token list passed in `#3` to the *⟨stored content⟩* given by `\__scontents_seq_item:nn` and removes `\c__scontents_hidden_space_str` from its end with `\__scontents_remove_trailing_eol:N`.

Otherwise the token list is empty and the “errors” `undefined-storage` or `index-out-of-range` are raised, except for a `stored content` that is not available yet in a remembered *sequence* (§13.12).

```

1822 \cs_new_protected:Npn \scontents_get:nnN #1 #2 #3
1823   { \exp_args:Nne \__scontents_get:nnN {#1} { \int_eval:n {#2} } #3 }
1824 \cs_new_protected:Npn \__scontents_get:nnN #1 #2 #3
1825   {
1826     \tl_clear:N #3
1827     \__scontents_seq_if_exist:nTF {#1}
1828     {
1829       \__scontents_if_pending:nnF {#1} {#2}
1830       {
1831         \bool_lazy_or:nnTF
1832           { \int_compare_p:nNn {#2} = { 0 } }
1833           {
1834             \int_compare_p:nNn { \int_abs:n {#2} }
1835               > { \__scontents_seq_count:n {#1} }
1836           }
1837           {
1838             \msg_error:nneee { scontents } { index-out-of-range }
1839               {#2} {#1} { \__scontents_seq_count:n {#1} }
1840           }
1841           {
1842             \tl_set:Nc #3 { \__scontents_seq_item:nn {#1} {#2} }
1843             \__scontents_remove_trailing_eol:N #3
1844           }
1845       }
1846     }
1847     { \msg_error:nnn { scontents } { undefined-storage } {#1} }
1848   }

```

(End of definition for `\scontents_get:nnN` and `\__scontents_get:nnN`. This function is documented on page 9.)

```

\scontents_map_inline:nn
\__scontents_map_inline:nn

```

The function `\scontents_map_inline:nn` increases `\g__scontents_map_int`, defines with the `{\inline function}` passed in `#2` a function whose name has the nesting level and calls `\__scontents_map_inline:nn`, which gets each `stored content` with `\__scontents_get:nnN` and passes it to that function. Each nesting level has its own function, so the `{\inline function}` can use `\scontents_map_inline:nn` again.

```

1849 \cs_new_protected:Npn \scontents_map_inline:nn #1 #2
1850   {
1851     \__scontents_seq_if_exist:nTF {#1}
1852     {
1853       \int_gincr:N \g__scontents_map_int
1854       \cs_gset_protected:cpn { __scontents_map_ \int_use:N \g__scontents_map_int :n }
1855         ##1 {#2}
1856       \exp_args:Nne \__scontents_map_inline:nn {#1} { \int_use:N \g__scontents_map_int }
1857       \int_gdecr:N \g__scontents_map_int
1858     }
1859     { \msg_error:nnn { scontents } { undefined-storage } {#1} }
1860   }
1861 \cs_new_protected:Npn \__scontents_map_inline:nn #1 #2
1862   {
1863     \int_step_inline:nn { \__scontents_seq_count:n {#1} }
1864     {
1865       \__scontents_get:nnN {#1} {##1} \l__scontents_map_item_tl
1866       \exp_args:Nc \exp_args:NV { __scontents_map_ #2 :n } \l__scontents_map_item_tl
1867     }
1868   }
1869 </core>

```

(End of definition for `\scontents_map_inline:nn` and `\__scontents_map_inline:nn`. This function is documented on page 9.)

13.27 Finish package

Finish package implementation, in ConTeXt ‘!’ is a letter again (§13.2.1).

```

1870 <context>\char_set_catcode_letter:N \!
1871 <plain|context>\ExplSyntaxOff
1872 <plain|context>\endinput
1873 <latex|core>\file_input_stop:

```

14 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.

A		N	
after	<u>1323</u>	\newenvsc	5, 37, <u>1090</u>
B		not-writing	<u>205</u>
before	<u>1323</u>	O	
C		overwrite	<u>222</u> , <u>772</u> , <u>1233</u> , <u>1438</u>
\cleanseqsc	9, 60, <u>1807</u>	overwrite-file	<u>205</u>
cmd-key-unknown	<u>287</u>	P	
cmd-key-value-unknown	<u>287</u>	Packages :	
Commands provided by SCONTENTS :		docstrip	21
\Scontents	49	expl3	21, 22, 60
\cleanseqsc	60	l3doc	21
\countsc	60	l3keys	52
\foreachsc	52	latex-lab	46
\getstored	49	ltxcmd	21, 39, 48
\meaningsc	57	lua-widow-control	23
\mergesc	59	verbatim	47
\newenvsc	44	xparse	21
\setupsc	60	xsim	39
\typestored	55	xtemplate	46
\countsc	8, 60, <u>1798</u>	positive-integer	<u>196</u>
D		print-all	<u>222</u>
debug-var	<u>222</u>	print-cmd	<u>222</u> , <u>1233</u> , <u>1438</u>
\dospecials	<u>1104</u>	print-env	<u>222</u> , <u>772</u>
E		Q	
empty-argument-command	<u>1231</u>	quark internal commands:	
empty-environment	<u>760</u>	\q__scontents_mark	<u>160</u>
empty-store-name	<u>196</u>	\q__scontents_stop	<u>160</u>
empty-variable	<u>1548</u>	R	
\endscontents	4, <u>1098</u>	remember	<u>222</u>
\endverbatimsc	<u>1176</u>	remember-latex-only	<u>611</u>
env-already-defined	<u>743</u>	remember-rerun	<u>619</u>
env-key-unknown	<u>268</u>	remember-undefined	<u>619</u>
env-key-value-unknown	<u>268</u>	rescanning-text	<u>760</u>
Environments provided by SCONTENTS :		S	
verbatimsc	46, 47	scan internal commands:	
F		\s__scontents_mark	<u>160</u>
force-eol	<u>222</u> , <u>772</u> , <u>1233</u>	\s__scontents_stop	<u>160</u>
\foreachsc	6, 49, <u>1427</u>	\Scontents	5, 47, <u>1289</u>
G		\scontents	4, <u>1098</u>
\getstored	6, 49, <u>1304</u>	scontents	4, <u>1098</u>
getstored	<u>1368</u> , <u>1673</u>	scontents commands:	
I		\scontents_count:n	9, <u>1815</u>
index-out-of-range	<u>178</u>	\scontents_get:nnN	9, <u>1822</u>
J		\scontents_if_exist:nTF	9, <u>1815</u>
junk-after-begin	<u>743</u>	\scontents_if_exist_p:n	9, <u>1815</u>
M		\scontents_map_inline:nn	9, <u>1849</u>
\meaningsc	7, 52, 56, <u>1656</u>	scontents internal commands:	
meaningsc	<u>1368</u> , <u>1673</u>	__scontents_analyse_nesting:n	<u>962</u>
\mergesc	7, 15, 52, 57, <u>1788</u>	__scontents_analyse_nesting_format:n	<u>984</u>
mergesc-missing-key	<u>1663</u>	__scontents_analyse_nesting_generic:nn	<u>984</u>
multiple-begin	<u>760</u>	__scontents_analyse_nesting_generic:w	<u>984</u>
		__scontents_analyse_nesting_generic-process:nn	<u>984</u>
		__scontents_analyse_nesting_latex:n	<u>984</u>

<code>_scontents_analyse_nesting_latex:w</code> ..	984	<code>\l_scontents_index_str</code>	1434
<code>_scontents_append_contents:nn</code>	484	<code>\g_scontents_last_stored_tl</code>	173
<code>_scontents_bsphack:</code>	453	<code>_scontents_lastfrom_seq:n</code>	586
<code>\l_scontents_char_value_int</code>	12	<code>_scontents_make_control_chars_active:</code>	423
<code>_scontents_check_line_process:nn</code> ...	798	<code>_scontents_map_inline:nn</code>	1849
<code>\l_scontents_cmd_name_tl</code>	164	<code>\g_scontents_map_int</code>	1812
<code>\l_scontents_current_seq_name_str</code> ...	1434	<code>\l_scontents_map_item_tl</code>	1812
<code>_scontents_define_env:nnn</code>	1057	<code>_scontents_meaningsc:n</code>	1626
<code>_scontents_define_generic_nesting-</code> <code>function:n</code>	984	<code>_scontents_meaningsc:nn</code>	1614
<code>_scontents_do_noligs:N</code>	412	<code>\l_scontents_meaningsc_arg_tl</code>	1613
<code>\c_scontents_end_env_tl</code>	133	<code>\l_scontents_mergesc_arg_tl</code>	1661
<code>\g_scontents_end_verbatimsc_tl</code>	133	<code>_scontents_mergesc_cmd:nn</code>	1673
<code>_scontents_env_begin_str:n</code>	729	<code>_scontents_mergesc_code:nn</code>	1689
<code>_scontents_env_end_function:</code>	837	<code>_scontents_mergesc_getstored:nn</code> ...	1771
<code>_scontents_env_end_str:n</code>	729	<code>\l_scontents_mergesc_keys_tl</code>	1661
<code>\l_scontents_env_name_tl</code>	133	<code>_scontents_mergesc_parse_item:n</code> ...	1707
<code>\l_scontents_environment_keys_tl</code> ...	723	<code>_scontents_mergesc_parse_list:n</code> ...	1707
<code>_scontents_esphack:</code>	453	<code>_scontents_mergesc_put_item:nn</code> ...	1707
<code>_scontents_file_if_exist:nTF</code>	103	<code>\g_scontents_name_sclinternal_seq</code> ...	171
<code>_scontents_file_if_writable:nTF</code> ...	357	<code>\l_scontents_nesting_aux_int</code>	723
<code>\l_scontents_file_name_tl</code>	165	<code>_scontents_nesting_decr:</code>	962
<code>_scontents_file_tl_write_start:n</code> ...	837	<code>\l_scontents_nesting_env_int</code>	723
<code>_scontents_file_write_cmd:nn</code>	380	<code>_scontents_nesting_incr:</code>	962
<code>\l_scontents_file_write_iow</code>	165	<code>\c_scontents_newline_tl</code>	117
<code>_scontents_file_write_stop:N</code>	837	<code>_scontents_nolig_list:</code>	1130
<code>_scontents_finish_storing:NNN</code>	500	<code>_scontents_normalise_line_ends:N</code> ...	798
<code>_scontents_foreach_add_body:n</code>	1380	<code>_scontents_par:</code>	401
<code>\l_scontents_foreach_after_bool</code>	1308	<code>_scontents_parse_mergesc:nw</code>	1707
<code>\l_scontents_foreach_after_tl</code>	1308	<code>_scontents_parse_mergesc_aux:nw</code> ...	1707
<code>\l_scontents_foreach_before_bool</code> ...	1308	<code>_scontents_parse_type_meaning_key:n</code>	1450
<code>\l_scontents_foreach_before_tl</code>	1308	<code>_scontents_parse_type_meaning_key:nn</code>	1450
<code>\l_scontents_foreach_body_tl</code>	1308	<code>\l_scontents_pending_bool</code>	1531
<code>\l_scontents_foreach_cmd_tl</code>	1308	<code>\l_scontents_plain_bool</code>	1101
<code>\g_scontents_foreach_exec_tl</code>	1308	<code>_scontents_plain_disable_outer_par:</code>	423
<code>\l_scontents_foreach_print_seq</code>	1308	<code>\l_scontents_processed_body_lines_tl</code>	723
<code>\l_scontents_foreach_start_int</code>	1308	<code>_scontents_range_limit:nn</code>	1450
<code>\l_scontents_foreach_step_int</code>	1308	<code>_scontents_range_parser:nnn</code>	1450
<code>\l_scontents_foreach_stop_bool</code>	1308	<code>_scontents_range_parser:nnnw</code>	1450
<code>\l_scontents_foreach_stop_int</code>	1308	<code>\l_scontents_range_start_tl</code>	1434
<code>_scontents_foreach_wrap:n</code>	1380	<code>_scontents_range_step:nn</code>	1450
<code>\l_scontents_foreach_wrapper_bool</code> ...	1308	<code>\l_scontents_range_stop_tl</code>	1434
<code>_scontents_foreachsc:nn</code>	1380	<code>_scontents_remember:n</code>	625
<code>\l_scontents_foreachsc_arg_tl</code>	1308	<code>\g_scontents_remember_iow</code>	603
<code>_scontents_format_case:nnn</code>	98	<code>\g_scontents_remember_iow</code>	603
<code>_scontents_generic_begin:</code>	1043	<code>\l_scontents_remember_item_str</code>	603
<code>_scontents_generic_end:</code>	1043	<code>\l_scontents_remember_name_str</code>	603
<code>_scontents_get:nnN</code>	1822	<code>\l_scontents_remember_new_str</code>	603
<code>_scontents_getfrom_seq:Nn</code>	547	<code>\g_scontents_remember_old_str</code>	603
<code>_scontents_getfrom_seq:nn</code>	547	<code>\g_scontents_remember_pending_bool</code> ...	599
<code>_scontents_getfrom_seq:nNn</code>	547	<code>_scontents_remember_read:</code>	652
<code>_scontents_getfrom_seq:nnn</code>	547	<code>_scontents_remember_read_line:n</code> ...	652
<code>_scontents_getfrom_seq_aux:nnn</code>	547	<code>_scontents_remember_read_line:w</code> ...	652
<code>_scontents_getstored:nn</code>	1294	<code>\g_scontents_remember_seq</code>	599
<code>_scontents_grab_opt_arg:n</code>	787	<code>_scontents_remember_write:</code>	685
<code>_scontents_grab_opt_arg:w</code>	787	<code>_scontents_remember_write_item:n</code> ...	685
<code>\c_scontents_hidden_space_str</code>	174	<code>_scontents_remember_write_seq:n</code> ...	685
<code>_scontents_if_digits:nTF</code>	1450	<code>_scontents_remove_leading_nl:n</code> ...	837
<code>_scontents_if_nested:TF</code>	962	<code>_scontents_remove_leading_nl:nn</code> ...	837
<code>_scontents_if_pending:nnTF</code>	639	<code>_scontents_remove_leading_nl:w</code> ...	837
<code>_scontents_if_pending:nTF</code>	639	<code>_scontents_remove_trailing_eol:N</code> ...	340
		<code>_scontents_remove_trailing_eol:w</code> ...	340

<code>\l__scontents_save_every_body_lines_tl</code>	723	<code>__scontents_verbatimsc_aux:</code>	1130
<code>\l__scontents_save_sf_int</code>	176	<code>__scontents_verbatimsc_instance:</code>	1188
<code>\l__scontents_save_skip</code>	176	<code>__scontents_verbatimsc_line:w</code>	1168
<code>\l__scontents_Scontents_arg_tl</code>	1229	<code>\c__scontents_version_str</code>	12
<code>__scontents_Scontents_code:nn</code>	1248	<code>__scontents_vobeyspaces:</code>	1130
<code>__scontents_Scontents_finish:</code>	1276	<code>\l__scontents_writable_bool</code>	165
<code>__scontents_Scontents_norm_arg:n</code>	1271	<code>\l__scontents_writing_bool</code>	165
<code>__scontents_Scontents_verb_arg:w</code>	1258	<code>__scontents_xobeysp:</code>	1130
<code>__scontents_seq_count:n</code>	514	<code>__scontents_xverb:</code>	1111
<code>__scontents_seq_if_exist:nTF</code>	514	<code>__scontents_xverb:w</code>	1111
<code>__scontents_seq_item:nn</code>	514	<code>sep</code>	1323
<code>\l__scontents_seq_item_seq</code>	171	<code>\setupsc</code>	3, 60, 1793
<code>__scontents_set_active_eq:NN</code>	423	<code>start</code>	1323
<code>__scontents_set_positive_int:Nnn</code>	212	<code>\startscontents</code>	5, 1098
<code>__scontents_set_write_out:n</code>	390	<code>\startverbatimsc</code>	8, 1185
<code>__scontents_set_write_store:n</code>	390	<code>step</code>	1323
<code>__scontents_setting_env:nn</code>	1057	<code>stop</code>	1323
<code>__scontents_setup_verb_processor:</code>	837	<code>\stopscontents</code>	5, 1098
<code>__scontents_start_after_option:w</code>	798	<code>\stopverbatimsc</code>	8, 1185
<code>__scontents_start_environment:w</code>	798	<code>store-all</code>	222
<code>__scontents_stop_environment:</code>	798	<code>store-cmd</code>	222, 1233
<code>__scontents_store_to_seq:NN</code>	492	<code>store-env</code>	222, 772
<code>\l__scontents_storing_bool</code>	165		
<code>__scontents_tab:</code>	401	T	
<code>\l__scontents_tab_width_int</code>	212	<code>tl commands:</code>	
<code>__scontents_tabs_to_spaces:</code>	403	<code>\tl_replace_all:Nnn</code>	157
<code>__scontents_tl_if_head_is_q_mark:nTF</code>	329	<code>\tl_retokenize:n</code>	157
<code>__scontents_typemeaning_setup:nn</code>	1531	<code>\tpestored</code>	7, 52, 54, 1608
<code>__scontents_tpestored:N</code>	1585	<code>tpestored</code>	1368, 1673
<code>__scontents_tpestored:nn</code>	1554		
<code>\l__scontents_tpestored_arg_tl</code>	1547	U	
<code>__scontents_unknown_keys_cmd:n</code>	317	<code>undefined-storage</code>	178
<code>__scontents_unknown_keys_cmd:nn</code>	317	<code>unknown</code>	222, 772, 1233, 1323, 1438
<code>__scontents_unknown_keys_env:n</code>	309		
<code>__scontents_unknown_keys_env:nn</code>	309	V	
<code>__scontents_use_delimit_by_s_stop:nw</code>	325	<code>verb-font</code>	222
<code>__scontents_use_i_delimit_by_s_stop:nw</code>	325	<code>\verbatimsc</code>	1176
<code>__scontents_use_none_delimit_by_q-stop:w</code>	325	<code>verbatimsc</code>	8, 1207
<code>__scontents_use_none_delimit_by_s-stop:w</code>	325		
<code>__scontents_verb_print_EOL:</code>	1585	W	
<code>__scontents_verb_processor_iterate:nnn</code>	837	<code>width-tab</code>	222, 1438
<code>__scontents_verb_processor_iterate:w</code>	837	<code>wrapper</code>	1323
<code>__scontents_verb_processor_output:n</code>	952	<code>write-cmd</code>	1233
		<code>write-env</code>	772
		<code>write-out</code>	772, 1233, 1438
		<code>writing-file</code>	205