SCONTENTS

STORES LaTeX CONTENTS

v2.9 2026-11-24*

©2019–2026 by Pablo González L†

CTAN: https://www.ctan.org/pkg/scontents
https://github.com/pablgonz/scontents

Abstract

This package allows to store 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*⟩.

Contents

1 **Description of the package** 1
2 **Motivation and Acknowledgments** . . . 1
3 **License and Requirements** 2
4 **The scontents package** 2
4.1 Installation 2
4.2 Loading and options 2
4.3 The TAB character 2
4.4 Configuration of the options 3
4.5 Options Overview 4
5 **User interface** 4
5.1 The environment scontents 4
5.2 The command \newenvsc 5
5.3 The command \Scontents 5
5.4 The command \getstored 6
5.5 The command \foreachsc 6
5.6 The command \typestored 7
5.7 The command \meaningsc 7
5.8 The command \mergesc 7
5.9 The environment verbatimsc 8
6 **Other commands provided** 8
6.1 The command \countsc 8
6.2 The command \cleanseqsc 9
7 **Programming interface** 9
8 **The scontents package in action** 9
9 **Examples** 10
9.1 From answers package 10
9.2 From filecontentsdef package . . 10
9.3 From TeX-SX 11
9.4 Customization of verbatimsc 13
9.5 The command \mergesc in action . . 15
9.6 Stored contents before storing them . 16
9.7 The tagged PDF example 17
10 **References** 17
11 **Change history** 18
12 **Index of Documentation** 21
13 **Implementation** 22
14 **Index of Implementation** 64

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 ConTeXt *"buffers"* by making use of ⟨*sequences*⟩.

2 Motivation and Acknowledgments

In LaTeX 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 ConTeXt 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 LaTeX team for their great work and to the different members of the TeX-SX community who have provided great answers and ideas. Here a note of the main ones:

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

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

*This file describes a documentation for v2.9, last revised 2026-11-24.

†E-mail: «pablgonz@educarchile.cl».

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 LaTeX to work (minimum required to LaTeX release 2026-11-01 and expl3 update). This package can be used with plain, context (MkIV and LMTX), xelatex, lualatex, pdflatex and the classical workflow latex»dvips»ps2pdf.

4 The scontents package

4.1 Installation

The package SCONTENTS is present in TeX Live and MiKTeX, 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.

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

4.2 Loading and options

The package is loaded in the usual way:

For LaTeX users

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

For plain TeX users

```
\input scontents.tex
```

For ConTeXt users

```
\usemodule{scontents}
```

The package options are not available for plain TeX and ConTeXt and must be passed using \setupsc (§4.4). The module works in ConTeXt MkIV (context --luatex) and in ConTeXt LMTX (context, with LuaMetaTeX).

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 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 TeX Live distribution, the TAB character is *printable* for latex, pdflatex and lualatex, but if you use xelatex you must add the -8bit option on the command line, otherwise you will get TeX character TAB '^^I' in the ⟨*output file*⟩.

As a general recommendation *"Do not use TAB character unless strictly necessary"*, for example within a *"verbatim environment"* that supports this character such as Verbatim of the packages fancyvrb[4], fvextra[5] or lstlisting of the package listings[6] or when you want to generate a MakeFile file.

[1] Check the answer given by Ulrich Diez in Keyboard TAB character in argument v (xparse).
[2] Check the answer given by Enrico Gregorio in How to output a tabulation into a file.

©2019–2026 by Pablo González L

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 ⟨keys⟩ 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 LaTeX, 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 '↹' 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* and all ⟨*keys*⟩ must be passed separated by commas. The spaces '␣' between \begin{scontents} and the [⟨*key = val*⟩] are ignored.

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 TEX and ConTEXt users

In plain TEX there are no environments as in LATEX. 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.

©2019–2026 by Pablo González L

ConTEXt users should use \startscontents and \stopscontents.

\startscontents
\stopscontents

\startscontents[⟨key=val⟩]
⟨body env⟩
\stopscontents

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

Options for environment

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

store-env = {⟨seq name⟩} default: *contents*

Sets the *name* of the *sequence* in which the {⟨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⟩. The ⟨file.ext⟩ can be in a subdirectory of the working directory, as sub/file.ext, if the subdirectory already exists, TEX does not create it. Absolute paths and paths with .. depend on the openout_any setting of the TEX distribution. XELATEX users using character TAB must add -8bit at the command line, otherwise you will get TEX 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 TEX and ConTEXt 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 TEX and \startmyenv ... \stopmyenv in ConTEXt. 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* '{',

©2019–2026 by Pablo González L

it will be assumed that the {⟨*argument*⟩} is nested within *braces*. Otherwise it will be assumed that {⟨*argument*⟩} is delimited by that first delimiter ⟨*del*⟩ like command `\verb`. Blank lines are preserved, escaped braces '`\{`' and '`\}`' must also be balanced if the {⟨*argument*⟩} is used with *braces* and character `TAB` typed from the keyboard are converted into spaces.

The *starred argument* '`*`' and [⟨*key* = *val*⟩] must NOT be separated by *spaces* '␣' between them and the command. Both versions can be used anywhere in the document. The command `\Scontents` can also be used in the {⟨*argument*⟩} of another command, `\Scontents*` cannot, as it happens with `\verb`.

Options for command

The command options can be configured globally using option in package or the `\setupsc` command and locally using [⟨*key* = *val*⟩].

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

Sets the *name* of the *sequence* in which the {⟨*argument*⟩} is stored. If the *sequence* does not exist, it will be created globally.

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

Sets if the ⟨*stored content*⟩ is displayed or not at the time of running the command. The ⟨*stored content*⟩ is extracted from the *sequence* {⟨*seq name*⟩} set by the key `store-cmd` at the time it is executed.

Options for writing files

The following ⟨*keys*⟩ and the `force-eol` and `overwrite` keys are available for both versions of the command. The `\Scontents*` command writes the {⟨*argument*⟩} to ⟨*file.ext*⟩ exactly as it was typed. The `\Scontents` command reads the {⟨*argument*⟩} 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 {⟨*argument*⟩} literally, use the *starred argument* '`*`'.

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

Sets the *name* of the ⟨*external file*⟩ in which the {⟨*argument*⟩} will be written. The ⟨*file.ext*⟩ will be created in the working directory and the {⟨*argument*⟩} will be *stored* in the *sequence* set by the key `store-cmd` 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 {⟨*argument*⟩} will be written. The ⟨*file.ext*⟩ will be created in the working directory and the {⟨*argument*⟩} will NOT be *stored* in any *sequence* set by the key `store-cmd` 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-cmd` and `write-out` the character `TAB` will be written in ⟨*file.ext*⟩. The ⟨*file.ext*⟩ can be in a subdirectory of the working directory, as `sub/file.ext`, if the subdirectory already exists, TEX does not create it. Absolute paths and paths with `..` depend on the `openout_any` setting of the TEX distribution. XELATEX users using character `TAB` must add `-8bit` at the command line, otherwise you will get TEX character `TAB` '`^^I`' in ⟨*file.ext*⟩.

5.4 The command `\getstored`

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

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

The command is robust and can be used as an {⟨*argument*⟩} for another command.

5.5 The command `\foreachsc`

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

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

Options for command

`getstored` ⟨*without value*⟩ default: *active*

Executes `\getstored` on each ⟨*stored content*⟩, as without ⟨*keys*⟩.

`typestored` ⟨*without value*⟩ default: *not active*

©2019–2026 by Pablo González L

Executes \typestored on each ⟨stored content⟩, which is shown in *verbatim style* (§5.6).

meaningsc ⟨without value⟩ default: *not active*

Executes \meaningsc on each ⟨stored content⟩ (§5.7).

sep = {⟨code⟩} default: *empty*

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

step = {⟨integer⟩} default: *1*

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

start = {⟨integer⟩} default: *1*

Set the *position* of the ⟨stored content⟩ within the *sequence* {⟨seq name⟩} from which to *start* executing. The value must be a ⟨positive integer⟩.

stop = {⟨integer⟩} default: *total*

Set the *position* of the ⟨stored content⟩ within the *sequence* {⟨seq name⟩} at which execution ends. The value must be a ⟨positive integer⟩.

before = {⟨code⟩} default: *empty*

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

after = {⟨code⟩} default: *empty*

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

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

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

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⟩. The ⟨file.ext⟩ can be in a subdirectory of the working directory, as sub/file.ext, if the subdirectory already exists, TEX does not create it. Absolute paths and paths with .. depend on the openout_any setting of the TEX distribution. XELATEX users using character TAB must add -8bit at the command line, otherwise you will get TEX character TAB '^^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

©2019–2026 by Pablo González L

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 TEX 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}
          }
  }{}
```

\startverbatimsc \stopverbatimsc

In plain TEX the environment is \verbatimsc ... \endverbatimsc and in ConTEXt 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.

©2019–2026 by Pablo González L

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 LaTeX, plain TeX and 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 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 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}.
```

©2019–2026 by Pablo González L

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

```
\begin{scontents}[store-env=abstract,print-env=true]
\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}
\end{scontents}
```

and

```
\begin{scontents}[store-env=description, print-env=true]
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}.
\end{scontents}
```

I stored the content in memory and then ran `\getstored` and `\typestored`. 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:

```
$ pdfdetach -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.

```
% arara: lualatex: { branch: developer }
% arara: clean: { extensions: [ aux, log ] }
\documentclass{article}
\usepackage[store-cmd=solutions]{scontents}
\newtheorem{ex}{Exercise}
\setlength{\parindent}{0pt}
\pagestyle{empty}
\begin{document}
\section{Problems}
\begin{ex}
First exercise
\Scontents{First solution.}
\end{ex}

\begin{ex}
Second exercise
\Scontents{Second solution.}
\end{ex}

\section{Solutions}
\foreachsc[sep={\\[10pt]}]{solutions}
\end{document}
```

9.2 From filecontentsdef package

Example 2

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

[3]The cool TEX automation tool: https://www.ctan.org/pkg/arara

```
% arara: lualatex: { branch: developer }
% arara: clean: { extensions: [ aux, log ] }
\documentclass{article}
\usepackage[store-env=defexercise,store-cmd=defexercise]{scontents}
\setlength{\parindent}{0pt}
\pagestyle{empty}
\begin{document}
% not starred
\Scontents{
Prove that \[x^n+y^n=z^n\] is not solvable in positive integers if $n$ is at
most $-3$.\par
}
% starred
\Scontents*|Refute the existence of black holes in less than $140$ characters.|
% write environment to \jobname.txt
\begin{scontents}[write-env=\jobname.txt]
\def\NSA{NSA}%
Prove that factorization is easily done via probabilistic algorithms and
advance evidence from knowledge of the names of its employees in the
seventies that the \NSA\ has known that for 40 years.\par
\end{scontents}
% see all stored
\begin{itemize}
\foreachsc[before={\item }]{defexercise}
\end{itemize}
% \getstored are robust :)
\section{\getstored[2]{defexercise}}
\end{document}
```

9.3 From TeX-SX

Example 3

Adapted from LaTeX equivalent of ConTeXt buffers. Source file .

```
% arara: lualatex: { branch: developer }
% arara: clean: { extensions: [ aux, log ] }
\documentclass{article}
\usepackage[store-cmd=tikz]{scontents}
\usepackage{tikz}
\setlength{\parindent}{0pt}
\pagestyle{empty}
\Scontents{\matrix{ \node (a) {$a$} ; & \node (b) {$b$} ; \\ } ;}
\Scontents{\matrix[ampersand replacement=\&]
{ \node (a) {$a$} ; \& \node (b) {$b$} ; \\ } ;}
\Scontents{\matrix{\node (a) {$a$} ; & \node (b) {$b$} ; \\ } ; }
\begin{document}
\section{tikzpicture}
\begin{tikzpicture}
\getstored[1]{tikz}
\end{tikzpicture}

\begin{tikzpicture}
\getstored[2]{tikz}
\end{tikzpicture}

\begin{tikzpicture}
\getstored{tikz}
\end{tikzpicture}

\begin{scontents}[store-env=buffer]
Hello World!

This is a \verb*|fake poor man's buffer :)|.
\end{scontents}

\section{source tikz}
\typestored[1]{tikz}
\typestored[2]{tikz}
\typestored[3]{tikz}

\section{fake buffer}
```

©2019–2026 by Pablo González L

```
\subsection{real content}
\getstored[1]{buffer}
\subsection{verbatim style}
\typestored[1]{buffer}
\subsection{meaning}
\meaningsc[1]{buffer}

\section{tikz again}
\foreachsc[before={\begin{tikzpicture}},after={\end{tikzpicture}},sep={\\[10pt]}]{tikz}
\end{document}
```

Example 4

Adapted from Collecting contents of environment and store them for later retrieval. Source file.

```
% arara: lualatex: { branch: developer }
% arara: clean: { extensions: [ aux, log ] }
\documentclass{article}
\usepackage{scontents}
\setlength{\parindent}{0pt}
\pagestyle{empty}
\begin{document}
\begin{scontents}[store-env=main]
Something for main A.
\end{scontents}

\begin{scontents}[store-env=main]
Something for \verb|main B|.
\end{scontents}

\begin{scontents}[store-env=other]
Something for \verb|other|.
\end{scontents}

\textbf{Let's print them}

This is first stored in main: \getstored[1]{main}\par
This is second stored in main: \getstored{main}\par
This is stored in other: \getstored{other}

\textbf{Print all of stored in main}\par
\foreachsc[sep={\\[10pt]}]{main}
\end{document}
```

Example 5

Adapted from Collect contents of an environment (that contains verbatim content). Source file.

```
% arara: lualatex: { branch: developer }
% arara: clean: { extensions: [ aux, log ] }
\documentclass{article}
\usepackage{scontents}
\setlength{\parindent}{0pt}
\pagestyle{empty}
\begin{document}
\section{Problem stated the first time}
\begin{scontents}[print-env=true,store-env=problem]
This is normal text.
\verb|This is from the verb command.|
\verb*|This is from the verb* command.|
This is normal text.
\begin{verbatim}
This is from the verbatim environment:
&%{}~
\end{verbatim}
\end{scontents}
\section{Problem restated}
\getstored[1]{problem}
\section{Problem restated once more}
\getstored[1]{problem}
\end{document}
```

©2019–2026 by Pablo González L

Example 6

Adapted from Environment hiding its content. Source file.

```
% arara: lualatex: { branch: developer }
% arara: clean: { extensions: [ aux, log ] }
\documentclass[10pt]{article}
\usepackage{scontents}
\newenvsc{forshort}[store-env=forshort,print-env=false]
\setlength{\parindent}{0pt}
\pagestyle{empty}
\begin{document}

Something in the whole course.

\begin{forshort}
    Just a summary...
\end{forshort}

\end{document}
```

9.4 Customization of `verbatimsc`

Example 7

Customization of verbatimsc using the fvextra[5] and tcolorbox[9] packages. Source file.

```
% arara: lualatex: { branch: developer }
% arara: clean: { extensions: [ aux, log ] }
\documentclass{article}
\usepackage{scontents}
\ExplSyntaxOn
\cs_undefine:N \verbatimsc
\cs_undefine:N \endverbatimsc
\ExplSyntaxOff
\usepackage{fvextra}
\usepackage{xcolor}
\definecolor{mygray}{gray}{0.9}
\usepackage{tcolorbox}
\newenvironment{verbatimsc}%
{\VerbatimEnvironment
\begin{tcolorbox}[colback=mygray, boxsep=0pt, arc=0pt, boxrule=0pt]
\begin{Verbatim}[fontsize=\scriptsize, breaklines, breakafter=*, breaksymbolsep=0.5em,
breakaftersymbolpre={\,\tiny\ensuremath{\rfloor}}]}%
{\end{Verbatim}%
\end{tcolorbox}}
\setlength{\parindent}{0pt}
\pagestyle{empty}
\begin{document}
\section{Test \texttt{\textbackslash begin\{scontents\}} with \texttt{fancyvrb}}
Test \verb+{scontents}+ \par

\begin{scontents}
Using \verb+scontents+ env no \verb+[key=val]+, save in seq \verb+contents+
with index 1.

Prove new \Verb*{ fancyvrb with braces } and environment \verb+Verbatim*+
\begin{verbatim}
 verbatim  environment
\end{verbatim}
\end{scontents}

\section{Test \texttt{\textbackslash Scontents} with \texttt{fancyvrb}}
\Scontents{ We have coded this in \LaTeX: $E=mc^2$.}

\section{Test \texttt{\textbackslash getstored}}
\getstored[1]{contents}\par
\getstored{contents}

\section{Test \texttt{\textbackslash meaningsc}}
\meaningsc[1]{contents}\par
\meaningsc[2]{contents}
```

```
\section{Test \texttt{\textbackslash typestored}}
\typestored[1]{contents}
\typestored[2]{contents}
\end{document}
```

Example 8

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

```
% arara: lualatex: { branch: developer }
% arara: clean: { extensions: [ aux, log ] }
\documentclass{article}
\usepackage{scontents}
\ExplSyntaxOn
\cs_undefine:N \verbatimsc
\cs_undefine:N \endverbatimsc
\ExplSyntaxOff
\usepackage{xcolor}
\usepackage{listings}
\lstnewenvironment{verbatimsc}
  {
   \lstset{
           basicstyle=\small\ttfamily,
           breaklines=true,
           columns=fullflexible,
           language=[LaTeX]TeX,
           numbers=left,
           numbersep=1em,
           numberstyle=\tiny\color{gray},
           keywordstyle=\color{red}
          }
  }{}
\setlength{\parindent}{0pt}
\pagestyle{empty}
\begin{document}
\section{Test \texttt{\textbackslash begin\{scontents\}} with \texttt{listings}}
Test \verb+{scontents}+ \par

\begin{scontents}
Using \verb+scontents+ env no \verb+[key=val]+, save in seq \verb+contents+ with index 1.\par

Prove \lstinline[basicstyle=\ttfamily]| lstinline | and environment \verb+Verbatim*+
\begin{verbatim}
  verbatim  environment
\end{verbatim}
\end{scontents}

\section{Test \texttt{\textbackslash Scontents*} with \texttt{listings}}

\Scontents*+We have coded this in \lstinline[basicstyle=\ttfamily]|\LaTeX: $E=mc^2$|
and more.+

\section{Test \texttt{\textbackslash getstored}}
\getstored{contents}\par
\getstored[1]{contents}

\section{Test \texttt{\textbackslash typestored}}
\typestored[1]{contents}
\typestored[2]{contents}
\end{document}
```

Example 9

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

```
% arara: lualatex: { branch: developer }
% arara: clean: { extensions: [ aux, log ] }
\documentclass{article}
\usepackage{scontents}
\ExplSyntaxOn
\cs_undefine:N \verbatimsc
```

©2019–2026 by Pablo González L

```
\cs_undefine:N \endverbatimsc
\ExplSyntaxOff
\usepackage{minted}
\newminted{tex}{linenos}
\newenvironment{verbatimsc}{\VerbatimEnvironment\begin{texcode}}{\end{texcode}}
\pagestyle{empty}
\setlength{\parindent}{0pt}
\begin{document}
\section{Test \texttt{\textbackslash begin\{scontents\}} with \texttt{minted}}
Test \verb+{scontents}+ \par

\begin{scontents}[overwrite,write-env=\jobname.tsc,force-eol=true]
Using \verb+scontents+ env no \verb+[key=val]+, save in seq \verb+contents+
with index 1.\par

Prove new \Verb*{ new fvextra with braces } and environment \verb+Verbatim*+
\begin{Verbatim}[obeytabs, showtabs, tab=\rightarrowfill, tabcolor=red]
No tab
        One real tab
                Two real Tab plus       one tab
\end{Verbatim}
\end{scontents}

\section{See \Verb{\jobname.tsc}}
Read \Verb{\jobname.tsc} (shows TABs as red arrows):
\VerbatimInput[obeytabs, showtabs, tab=\rightarrowfill, tabcolor=red]{\jobname.tsc}

\section{Test \texttt{\textbackslash Scontents} with \texttt{minted}}

\Scontents{ We have coded \par this in \LaTeX: $E=mc^2$.}

\section{Test \texttt{\textbackslash getstored}}
\getstored[1]{contents}\par
\getstored{contents}

\section{Test \texttt{\textbackslash typestored}}
\typestored[1]{contents}
\typestored[2]{contents}
\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 .

```
% arara: lualatex: { branch: developer }
% arara: clean: { extensions: [ aux, log ] }
\documentclass{article}
\usepackage{scontents}
% Fix part of a MCE that should go before babel's loading
\begin{scontents}[store-env=mce]
\documentclass[french]{article}
\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}
\usepackage{lmodern}
\usepackage[a4paper]{geometry}
\end{scontents}
% Fix part of a MCE that should go after (>=) babel's loading
\begin{scontents}[store-env=mce]
\usepackage{babel}
\begin{document}
\end{scontents}
% Fix part of a MCE that should go after its body
\begin{scontents}[store-env=mce]
\end{document}
\end{scontents}
\begin{document}
\section{First answer}
% Variable part of a MCE that should added to the fixed preamble, before babel's loading
\begin{scontents}[store-env=mce-1]
\usepackage{amsmath}
```

```
\end{scontents}
% Variable part of a MCE being the code snippet
\begin{scontents}[store-env=mce-1]
\begin{align}
  0 & \neq 1 \\
  1 & \neq 0
\end{align}
\end{scontents}
\begin{description}
\item[Preamble's addition]\leavevmode
  \typestored[1]{mce-1}
\item[Code snippet]\leavevmode
  \typestored[2]{mce-1}
\item[MCE]\leavevmode
  \mergesc[typestored, print-cmd=true]
    {
      {mce}[1], {mce-1}[1], {mce}[2], {mce-1}[2], {mce}[3]
    }
\end{description}
\section{Second answer}
% Variable part of a MCE that should added to the fixed preamble, before babel's loading
\begin{scontents}[store-env=mce-2]
\usepackage{amsmath}
\end{scontents}
% Variable part of a MCE being the code snippet
\begin{scontents}[store-env=mce-2]
\begin{flalign}
  0 & \neq 1 \\
  1 & \neq 0
\end{flalign}
\end{scontents}

\begin{description}
\item[Preamble's addition]\leavevmode
  \typestored[1]{mce-2}
\item[Code snippet]\leavevmode
  \typestored[2]{mce-2}
\item[MCE]\leavevmode
  \mergesc[typestored, print-cmd=true, write-out=mce.txt, overwrite=true]
    {
      {mce}[1], {mce-2}[1], {mce}[2], {mce-2}[2], {mce}[3]
    }
\end{description}
\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/pablgonz/scontents/issues/1. It needs two compilations. Source file.

```
% arara: lualatex: { branch: developer }
% arara: lualatex: { branch: developer }
% arara: clean: { extensions: [ aux, log, sct ] }
\documentclass{article}
\usepackage[remember=objectives]{scontents}
\begin{document}
\section{Objectives}
The project has \countsc{objectives} objectives, developed in the
methods: \foreachsc[sep={; }]{objectives}.

The first one is the key of the project: \getstored[1]{objectives}.

\section{Methods}
\subsection{First objective}
\Scontents[store-cmd=objectives, print-cmd]{measure the effect of the method on small samples}

The samples are taken from the previous studies.

\subsection{Second objective}
\begin{scontents}[store-env=objectives, print-env]
```

```
compare the results with the ones of the previous studies
\end{scontents}
\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.

```
% arara: lualatex: { branch: developer }
% arara: clean: { extensions: [ aux, log ] }
\DocumentMetadata{tagging=on, lang=en-US, pdfversion=2.0, pdfstandard=ua-2}
\documentclass{article}
\usepackage{scontents,unicode-math,hyperref}
\hypersetup{pdftitle = {Test scontents package},}
\begin{document}
Some

\begin{scontents}[print-env=true]
    First code \verb|\foo|

    And more code \verb|\bar|
\end{scontents}

Text

\begin{scontents}[print-env=true]
    Second code \verb|\foo|

    And more code \verb|\bar|
\end{scontents}

Text

\Scontents*{code \verb|\baz|}

% \typestored
\typestored[1]{contents}

% \mergesc
\mergesc[typestored]{ {contents}[1-end] }

% \getstored
\getstored[2]{contents}
\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 LaTeX Project. “The expl3 package”. Available from CTAN, https://www.ctan.org/pkg/expl3, 2026.

[3] Fischer, Ulrike. “tagpdf – LaTeX 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 LaTeX”. 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 LaTeX”. 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.

©2019–2026 by Pablo González L

[9] F. Sturm, Thomas. “tcolorbox – Coloured boxes, for LaTeX examples and theorems, etc”. Available from ctan, https://ctan.org/pkg/tcolorbox, 2026.

[10] The LaTeX Project. “The xparse package”. Available from ctan, https://www.ctan.org/pkg/xparse, 2024.

[11] The LaTeX Project. “The LaTeX 2_ε sources”. Available from ctan, https://ctan.org/tex-archive/macros/latex/base, 2026.

[12] The LaTeX Project. “LaTeX 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 LaTeX Project. “latex-lab – LaTeX laboratory”. Available from ctan, https://www.ctan.org/pkg/latex-lab, 2026.

[15] The LaTeX Project. “verbatim – Reimplementation of and extensions to LaTeX verbatim”. Available from ctan, https://www.ctan.org/pkg/verbatim, 2023.

[16] The LaTeX Project. “The LaTeX3 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].

v2.9 (ctan), 2026-11-24
- Spaces between \begin{scontents} and its optional argument are ignored.
- Documented that ⟨*output files*⟩ can be written in existing subdirectories and that \Scontents can be used in the argument of another command.
- Support for ConTeXt LMTX (LuaMetaTeX).

v2.8 (ctan), 2026-11-01
- Update requirement to LaTeX 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 ConTeXt for files written in the same run.
- Fix loading in ConTeXt when expl3 was already loaded by another module.
- The messages of the environments show their start and end as written in plain TeX and ConTeXt.
- 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 TeX.
- Unknown ⟨*keys*⟩ in the package options, \setupsc and the ⟨*initial keys*⟩ of \newenvsc return an “error” that names the command.
- Added regression tests for LaTeX, plain TeX and ConTeXt, run with l3build check.
- New remember key to use the ⟨*stored contents*⟩ before they are stored, only in LaTeX (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).

©2019–2026 by Pablo González L

v2.7 (ctan), 2026-06-01
- Internal environment `verbatimsc` update for *tagged* PDF.
- Update requirement to LaTeX 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 LaTeX release 2025-11-01.

v2.5 (ctan), 2025-11-01
- Replacing `\scantokens` (primitive) with `\tl_retokenize:n`.
- Cleanup warnings and details returned by `expltools`.
- Update minimum required of `expl3` to 2025-07-08.

v2.4 (ctan), 2025-05-15
- Optimization of expansion code from 'x' to 'e'.
- Restructuring code for documentation and implementation.
- Add new keys for `\typestored` and `\meaningsc`.
- Check the version of `expl3` in plain TeX and ConTeXt.

v2.3 (ctan), 2025-04-23
- Adapting the `verbatimsc` environment for *tagged* PDF.
- Update minimum required to LaTeX release 2024-11-01.
- Safer code for replacement `\obeyedline`.

v2.2 (ctan), 2025-03-26
- Fix internal definition for some functions.
- Replace `\peek_charcode_ignore_spaces:NTF` by `\peek_charcode:NTF`.
- Set correct code for `\obeyedline` implement in LaTeX release 2024-06-01.

v2.1 (ctan), 2024-06-14
- Fix `\cleanseqsc` command.
- Add `\mergesc` command.
- Fix internal definition for seq var.
- Fix internal code for `\typestored`.
- Replace `\cs_argument_spec:N` by `\cs_parameter_spec:N`.
- Detect `l3keys2e` package (obsolete in june 2022 LaTeX release).
- Minor adjustments in the documentation.

v2.0 (ctan), 2022-04-04
- Adapting the `verbatimsc` environment (compatibility `verbatim` package).
- Removed compatibility layer for older LaTeX releases.
- Fix loader in plain TeX and ConTeXt.
- Minor adjustments in the documentation.

v1.9 (ctan), 2020-01-21
- Update and improvements in the internal code.
- Updating the generic code for `I/O` verification.
- Add `write-cmd` and `write-out` keys for `\Scontents*`.
- Fix `sep` key in `\foreachsc`.

v1.8 (ctan), 2019-11-18
- Add `\newenvsc` command.
- Fix nested environment in plain TeX and ConTeXt.
- Modified default value in `\getstored`.
- Add `overwrite` key to reduce `I/O` operations.
- Deleted an unnecessary group in the code.

v1.7 (ctan), 2019-10-29
- The `verbatimsc` environment was rewritten.
- Minor adjustments in documentation.

v1.6 (ctan), 2019-10-26
- The internal behavior of `\getstored` has been modified.
- The internal behavior of `\foreachsc` has been modified.
- Corrected file extension for ConTeXt.
- Remove spurious warning.

v1.5 (ctan), 2019-10-24
- Add support for plain TeX and ConTeXt.
- Split internal code for optimization.
- Add support for vertical spaces in `[`⟨*key* = *val*⟩`]`.
- Add `\foreachsc` command.
- Check if `verbatim` package is loaded.

v1.4 (ctan), 2019-10-03
- Add `store-all` key.
- Messages and keys were separated.
- Restructuring of documentation.
- Now the version of `expl3` is checked instead of `xparse`.
- The internal behavior of `force-eol` has been modified.

v1.3 (ctan), 2019-09-24
- The environment `scontents` can now nest.
- Added `force-eol`, `verb-font` and `width-tab` keys.
- The extra space has been removed when you run `\getstored`.
- Internal code has been rewritten more efficiently.
- Remove *starred argument* '*' for `\typestored`.
- Remove `filecontentsdef` dependency.
- Changing `\regex_replace_all:` for `\tl_replace_all:`.

©2019–2026 by Pablo González L

v1.2 (ctan), 2019-08-28
- Restructuring of documentation.
- Added copy of `\tex_scantokens:`.

v1.1 (ctan), 2019-08-12
- Extension of documentation.
- Replace `\tex_endinput:D` by `\file_input_stop:`.

v1.0 (ctan), 2019-07-30
- First public release.

©2019–2026 by Pablo González L

12 Index of Documentation

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

C

Commands provided by SCONTENTS:

\Scontents 5
\cleanseqsc 9
\countsc 8
\foreachsc 6
\getstored 6
\meaningsc 7
\mergesc 7
\newenvsc 5
\setupsc 3
\typestored 7

E

Environments :

filecontentsdefmacro 1
lstlisting 2
Verbatim 2

Environments provided by SCONTENTS:

scontents 4

K

Keys for \foreachsc provided by SCONTENTS:

after 7
before 7
getstored 6
meaningsc 7
sep 7
start 7
step 7
stop 7
typestored 6
wrapper 7

Keys for \Scontents provided by SCONTENTS:

print-cmd 6
store-cmd 6
write-cmd 6
write-out 6

Keys for \setupsc provided by SCONTENTS:

debug-var 3
force-eol 3
overwrite 3
print-all 3
remember 3
store-all 3
verb-font 3
width-tab 3

Keys for environments provided by SCONTENTS:

print-env 5
store-env 5
write-env 5
write-out 5

P

Packages :

answers 10
expl3 . . . . . 1, 2, 9, 18, 19
fancyvrb 2, 3, 8
filecontentsdef 1, 9, 10, 18, 19
fvextra 2, 3, 5, 8, 13
l3keys2e 19
l3seq 1, 9
listings 2, 5, 8, 14
minted 8, 14
tcolorbox 13
verbatim 19
xparse 19

©2019–2026 by Pablo González L

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 LaTeX, scontents.tex for plain TeX and t-scontents.mkiv for ConTeXt, 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 TeX and ConTeXt, 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:

```
⟨@@=scontents⟩
```

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

```
⟨*loader⟩
\def\ScontentsFileDate{2026-11-24}%
\def\ScontentsFileVersion{2.9}%
\def\ScontentsFileDescription{Stores LaTeX contents in memory or files}%
```

The 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 LaTeX release 2026-11-01 is the latest available in TeX Live 2026 and is necessary to be able to implement the package's full compatibility with *tagged* PDF.

```
⟨*latex⟩
\NeedsTeXFormat{LaTeX2e}[2026-11-01]
\ProvidesExplPackage
  {scontents} {\ScontentsFileDate} {\ScontentsFileVersion}
  {\ScontentsFileDescription}
⟨/latex⟩
```

13.2.1 Load expl3-generic and xparse-generic in plain TeX and ConTeXt

\c__scontents_version_str
\l__scontents_char_value_int

The plain TeX and ConTeXt 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 ConTeXt 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).

```
⟨*!latex⟩
⟨context⟩\writestatus{loading}{User Module scontents v\ScontentsFileVersion}
⟨context⟩\unprotect
⟨context⟩\catcode`\!=12\relax
\input expl3-generic.tex
\ExplSyntaxOn
\str_const:ce { c__scontents_version_str } { \ScontentsFileDate\space
  v\ScontentsFileVersion\space \ScontentsFileDescription }
\iow_log:e { Package: ~ scontents ~ \str_use:c { c__scontents_version_str } }
```

Outside of LaTeX we can't use ltcmd[11, 12], part of the 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.

```
\int_new:N  \l__scontents_char_value_int
\int_set:Nn \l__scontents_char_value_int { \char_value_catcode:n { `\@ } }
\char_set_catcode_letter:N \@
\file_input:n { xparse-generic.tex }
\char_set_catcode:nn { `\@ } { \l__scontents_char_value_int }
⟨/!latex⟩
```

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

©2019–2026 by Pablo González L

13.2.2 Checking expl3 version

For plain TEX, LATEX and ConTEXt we must check the minimum requirement, in this case `\tl_retokenize:n` which was added in release 2025-07-08 of `expl3` included in TEX Live 2025.

```
\cs_if_exist:NF \tl_retokenize:n
  {
    \msg_new:nnn { scontents } { expl-too-old }
      {
        Please~install~an~up~to~date~TeX~distribution~or~update~using~
        your~TeX~package~manager~or~from~CTAN. \\
        See~documentation.~Loading~scontents~will~abort!
      }
    \msg_fatal:nn { scontents } { expl-too-old }
    \ExplSyntaxOff
    \file_input_stop:
  }
```

13.2.3 Preventing double loading in plain TEX

In plain TEX, check that the package isn't being loaded twice, LATEX and ConTEXt already defend against that:

```
⟨*plain⟩
\tl_if_exist:NT \l__scontents_cmd_name_tl
  {
    \msg_new:nnn { scontents } { already-loaded }
      {
        The~'scontents'~package~is~already~loaded.~
        Aborting~input~\msg_line_context:.
      }
    \msg_warning:nn { scontents } { already-loaded }
    \ExplSyntaxOff
    \file_input_stop:
  }
⟨/plain⟩
⟨/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`.

```
⟨*core⟩
\def\ScontentsCoreFileDate{2026-11-24}%
\begingroup
  \catcode32=10
  \endlinechar=32
  \def\next{\endgroup}%
  \expandafter\ifx\csname PackageError\endcsname\relax
    \begingroup
      \def\next{\endgroup\endgroup}%
      \def\PackageError#1#2#3%
        {%
          \endgroup
          \errhelp{#3}%
          \errmessage{#1 Error: #2!}%
        }%
  \fi
  \expandafter\ifx\csname ScontentsFileDate\endcsname\relax
    \def\next
      {%
        \PackageError{scontents}{No scontents loader detected}
          {%
            You have attempted to use the scontents code directly rather
            than using the correct loader. Loading of scontents will abort.
          }%
        \endgroup
        \endinput
      }%
  \else
    \ifx\ScontentsFileDate\ScontentsCoreFileDate
    \else
```

```
      \def\next
        {%
          \PackageError{scontents}{Mismatched scontents files detected}
            {%
              You have attempted to load scontents with mismatched files:
              probably you have one or more files 'locally installed' which
              are in conflict. Loading of scontents will abort.
            }%
          \endgroup
          \endinput
        }%
    \fi
  \fi
\next
⟨/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 ConTEXt.

__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 ConTEXt.

```
⟨*loader⟩
\cs_new:Npn \__scontents_format_case:nnn #1 #2 #3
⟨latex⟩   {#1} % LaTeX
⟨plain⟩   {#2} % Plain/Generic
⟨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 ConTEXt 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 LuaTEX, the ⟨*output file*⟩ is always in the working directory.

```
⟨!context⟩\cs_new_eq:NN \__scontents_file_if_exist:nTF \file_if_exist:nTF
⟨*context⟩
\prg_new_conditional:Npnn \__scontents_file_if_exist:n #1 { TF }
  {
    \str_if_eq:eeTF
      {
        \lua_now:e
          { tex.write(lfs.isfile("\lua_escape:e {#1}")~and~"1"~or~"0") }
      }
      { 1 }
      { \prg_return_true: }
      { \prg_return_false: }
  }
⟨/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*⟩.

```
\tl_const:Ne \c__scontents_newline_tl { \iow_char:N \^^J }
```

(*End of definition for* \c__scontents_newline_tl.)

__scontents_retokenize:n

The function __scontents_retokenize:n rescans the ⟨*stored content*⟩ passed in #1. In LATEX, plain TEX and ConTEXt MkIV it is \tl_retokenize:n.

```
⟨!context⟩\cs_new_eq:NN \__scontents_retokenize:n \tl_retokenize:n
```

In ConTEXt LMTX the primitive \scantokens of LuaMetaTeX does not split its argument into lines at the \newlinechar, so all the lines of the ⟨*stored content*⟩ would be rescanned as a single line. There the lines are passed one by one to tex.print, which ends each of them with the \endlinechar, as \scantokens does. The test with \contextlmtxmode is adapted from the file t-lua-widow-control.mkxl of Max Chernoff's lua-widow-control package, \contextlmtxmode is described in https://source.contextgarden.net/tex/context/base/mkxl/context.mkxl.

```
⟨*context⟩
```

```
\bool_if:NTF \contextlmtxmode
  {
    \lua_now:n
      {
        function~scontents_retokenize(s)~
          local~nl~=~string.char(10)~
          for~line~in~(s~..~nl):gmatch("(.-)"~..~nl)~do~
            tex.print(line)~
          end~
        end
      }
    \cs_new_protected:Npn \__scontents_retokenize:n #1
      {
        \lua_now:e
          { scontents_retokenize("\lua_escape:e { \tl_to_str:n {#1} }") }
      }
  }
  { \cs_new_eq:NN \__scontents_retokenize:n \tl_retokenize:n }
⟨/context⟩
```

(*End of definition for* `\__scontents_retokenize:n`.)

`\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.

```
\tl_new:N \g__scontents_end_verbatimsc_tl
\tl_gset_rescan:Nnn \g__scontents_end_verbatimsc_tl
  {
    \char_set_catcode_other:N \\
⟨*latex⟩
    \char_set_catcode_other:N \{
    \char_set_catcode_other:N \}
⟨/latex⟩
  }
⟨latex⟩  { \end{verbatimsc} }
⟨plain⟩  { \endverbatimsc }
⟨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).

```
\tl_new:N    \l__scontents_env_name_tl
\tl_const:Ne \c__scontents_end_env_tl
  {
    \c_backslash_str
⟨latex|plain⟩    end
⟨context⟩    stop
⟨latex⟩    \c_left_brace_str
      \exp_not:N \l__scontents_env_name_tl
⟨latex⟩    \c_right_brace_str
  }
```

(*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:

```
\file_input:n { scontents-code.tex }
⟨/loader⟩
```

13.5 Some kernel variants

`\tl_replace_all:NeV` `\__scontents_retokenize:V` `\__scontents_retokenize:e`

Some nonstandard kernel variants and the variants of `\__scontents_retokenize:n` (§13.3).

```
⟨*core⟩
\cs_generate_variant:Nn \tl_replace_all:Nnn { NeV }
\cs_generate_variant:Nn \__scontents_retokenize:n { V, e }
```

(*End of definition for* `\tl_replace_all:NeV` *and* `\__scontents_retokenize:V`.)

©2019–2026 by Pablo González L

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.

```
\quark_new:N \q__scontents_stop
\quark_new:N \q__scontents_mark
\scan_new:N  \s__scontents_stop
\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).

```
\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 `\__scontents_file_if_writable:nTF` (§13.10).

```
\tl_new:N   \l__scontents_file_name_tl
\iow_new:N  \l__scontents_file_write_iow
\bool_new:N \l__scontents_writing_bool
\bool_new:N \l__scontents_storing_bool
\bool_new:N \l__scontents_writable_bool
\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).

```
\seq_new:N \l__scontents_seq_item_seq
\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.

```
\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 TEX 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.

```
\str_const:Ne \c__scontents_hidden_space_str
  { \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).

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

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

©2019–2026 by Pablo González L

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).

```
\msg_new:nnnn { scontents } { undefined-storage }
  { Storage~named~'#1'~is~not~defined. }
  { No~content~has~been~stored~in~a~sequence~named~'#1'. }
\msg_new:nnnn { scontents } { index-out-of-range }
  {
    \int_compare:nNnTF {#1} = { 0 }
      { Index~of~sequence~'#2'~cannot~be~zero. }
      {
        Index~'#1'~out~of~range~for~'#2',~the~
        \int_compare:nNnTF {#1} > { 0 }
          { maximum~is~#3. }
          { minimum~is~-#3. }
      }
  }
  {
    The~sequence~'#2'~has~#3~stored~contents,~use~an~index~from~
    1~to~#3~or~from~-#3~to~-1.
  }
```

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

positive-integer
empty-store-name

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).

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

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

13.7.2 Definition of messages for writing files

writing-file
overwrite-file
not-writing

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).

```
\msg_new:nnn { scontents } { writing-file }
  { Writing~file~'#1'. }
\msg_new:nnn { scontents } { overwrite-file }
  { Overwriting~file~'#1'. }
\msg_new:nnn { scontents } { not-writing }
  { File~'#1'~already~exists.~Not~writing~\msg_line_context:. }
⟨/core⟩
```

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

13.8 Keys for the package

\l__scontents_tab_width_int
__scontents_set_positive_int:Nnn

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 ConTEXt (§13.2.1).

```
⟨*core⟩
```

```
\int_new:N \l__scontents_tab_width_int
\cs_new_protected:Npn \__scontents_set_positive_int:Nnn #1 #2 #3
  {
    \bool_lazy_or:nnTF
      { \tl_if_blank_p:n {#3} }
      { \int_compare_p:nNn {#3} < { 1 } }
      { \msg_error:nnnn { scontents } { positive-integer } {#2} {#3} }
      { \int_set:Nn #1 {#3} }
  }
```

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

store-env
store-cmd
verb-font
print-env
print-cmd
force-eol
overwrite
width-tab
print-all
store-all
debug-var
remember
unknown

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.

```
\keys_define:nn { scontents }
  {
    store-env .tl_set:N         = \l__scontents_name_seq_env_tl,
    store-env .initial:n        = contents,
    store-env .value_required:n = true,
    store-cmd .tl_set:N         = \l__scontents_name_seq_cmd_tl,
    store-cmd .initial:n        = contents,
    store-cmd .value_required:n = true,
    verb-font .tl_set:N         = \l__scontents_verb_font_tl,
    verb-font .value_required:n = true,
    print-env .bool_set:N       = \l__scontents_print_env_bool,
    print-env .initial:n        = false,
    print-env .default:n        = true,
    print-cmd .bool_set:N       = \l__scontents_print_cmd_bool,
    print-cmd .initial:n        = false,
    print-cmd .default:n        = true,
    force-eol .bool_set:N       = \l__scontents_forced_eol_bool,
    force-eol .initial:n        = false,
    force-eol .default:n        = true,
    overwrite .bool_set:N       = \l__scontents_overwrite_bool,
    overwrite .initial:n        = false,
    overwrite .default:n        = true,
    width-tab .code:n           =
      { \__scontents_set_positive_int:Nnn \l__scontents_tab_width_int { width-tab } {#1} },
    width-tab .initial:n        = 1,
    width-tab .value_required:n = true,
    print-all .meta:n           = { print-env = #1, print-cmd = #1 },
    print-all .default:n        = true,
    store-all .meta:n           = { store-env = #1, store-cmd = #1 },
    store-all .value_required:n = true,
    debug-var .bool_set:N       = \l__scontents_debug_var_bool,
    debug-var .initial:n        = true,
    debug-var .default:n        = true,
    remember  .code:n           = { \__scontents_remember:n {#1} },
    remember  .value_required:n = true,
    remember  .usage:n          = preamble,
    unknown   .code:n           = { \__scontents_unknown_keys_cmd:n {#1} },
  }
⟨/core⟩
```

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

```
⟨loader⟩\keys_define:nn { scontents }
⟨latex⟩  { verb-font .initial:n = \ttfamily }
⟨plain|context⟩  { verb-font .initial:n = \tt }
```

In LaTeX the ⟨*keys*⟩ passed as options to the package are processed and an unknown ⟨*key*⟩ returns an "error".

```
⟨*latex⟩
\tl_set:Nn \l__scontents_cmd_name_tl { usepackage }
\ProcessKeyOptions [ scontents ]
⟨/latex⟩
```

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

©2019–2026 by Pablo González L

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

env-key-unknown
env-key-value-unknown

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).

```
⟨*core⟩
\msg_new:nnnn { scontents } { env-key-unknown }
  {
    The~key~'#1'~is~unknown~by~environment~
    '\l__scontents_env_name_tl'~and~is~being~ignored.
  }
  {
    The~environment~'\l__scontents_env_name_tl'~does~not~have~a~key~called~'#1'.\\
    Check~that~you~have~spelled~the~key~name~correctly.
  }
\msg_new:nnnn { scontents } { env-key-value-unknown }
  {
    The~key~'#1=#2'~is~unknown~by~environment~
    '\l__scontents_env_name_tl'~and~is~being~ignored.
  }
  {
    The~environment~'\l__scontents_env_name_tl'~does~not~have~a~key~called~'#1'.\\
    Check~that~you~have~spelled~the~key~name~correctly.
  }
```

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

cmd-key-unknown
cmd-key-value-unknown

The "errors" cmd-key-unknown and cmd-key-value-unknown are raised when an unknown ⟨*key*⟩, without 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).

```
\msg_new:nnnn { scontents } { cmd-key-unknown }
  {
    The~key~'#1'~is~unknown~by~command~
    \c_backslash_str \l__scontents_cmd_name_tl \c_space_tl
    and~is~being~ignored~\msg_line_context:.
  }
  {
    The~command~\c_backslash_str \l__scontents_cmd_name_tl \c_space_tl
    does~not~have~a~key~called~'#1'.\\
    Check~that~you~have~spelled~the~key~name~correctly.
  }
\msg_new:nnnn { scontents } { cmd-key-value-unknown }
  {
    The~key~'#1=#2'~is~unknown~by~command~
    \c_backslash_str \l__scontents_cmd_name_tl \c_space_tl
    and~is~being~ignored~\msg_line_context:.
  }
  {
    The~command~\c_backslash_str \l__scontents_cmd_name_tl \c_space_tl
    does~not~have~a~key~called~'#1'.\\
    Check~that~you~have~spelled~the~key~name~correctly.
  }
```

(*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
__scontents_unknown_keys_env:nn

The function __scontents_unknown_keys_env:n raises the "error" for an unknown ⟨*key*⟩ passed to an environment defined by \newenvsc.

```
\cs_new_protected:Npn \__scontents_unknown_keys_env:n #1
  { \exp_args:NV \__scontents_unknown_keys_env:nn \l_keys_key_str {#1} }
\cs_new_protected:Npn \__scontents_unknown_keys_env:nn #1 #2
  {
    \tl_if_blank:nTF {#2}
      { \msg_error:nnn { scontents } { env-key-unknown } {#1} }
      { \msg_error:nnnn { scontents } { env-key-value-unknown } {#1} {#2} }
  }
```

©2019–2026 by Pablo González L

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

`\__scontents_unknown_keys_cmd:n`
`\__scontents_unknown_keys_cmd:nn`

The function `\__scontents_unknown_keys_cmd:n` raises the "error" for an unknown ⟨*key*⟩ passed to the commands `\Scontents`, `\foreachsc`, `\mergesc` and `\setupsc`, to the ⟨*initial keys*⟩ of `\newenvsc` and to the package.

```
\cs_new_protected:Npn \__scontents_unknown_keys_cmd:n #1
  { \exp_args:NV \__scontents_unknown_keys_cmd:nn \l_keys_key_str {#1} }
\cs_new_protected:Npn \__scontents_unknown_keys_cmd:nn #1 #2
  {
    \tl_if_blank:nTF {#2}
      { \msg_error:nnn { scontents } { cmd-key-unknown } {#1} }
      { \msg_error:nnnn { scontents } { cmd-key-value-unknown } {#1} {#2} }
  }
```

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

13.10 Utility functions

`\__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`

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

```
\cs_new:Npn \__scontents_use_delimit_by_s_stop:nw #1 \s__scontents_stop {#1}
\cs_new:Npn \__scontents_use_i_delimit_by_s_stop:nw #1 #2 \s__scontents_stop {#1}
\cs_new:Npn \__scontents_use_none_delimit_by_s_stop:w #1 \s__scontents_stop { }
\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`.

```
\prg_new_protected_conditional:Npnn \__scontents_tl_if_head_is_q_mark:n #1
  { F, TF }
  {
    \exp_after:w \if_meaning:w
      \exp_after:w
        \q__scontents_mark \__scontents_use_i_delimit_by_s_stop:nw #1 ? \s__scontents_stop
      \prg_return_true:
    \else:
      \prg_return_false:
    \fi:
  }
```

(*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).

```
\cs_new_protected:Npe \__scontents_remove_trailing_eol:N #1
  {
    \exp_not:N \exp_after:w \exp_not:N \__scontents_remove_trailing_eol:w
      #1 \s__scontents_stop \c__scontents_hidden_space_str \s__scontents_stop \s__scontents_mark
  }
\use:e
  {
    \cs_new_protected:Npn \exp_not:N \__scontents_remove_trailing_eol:w #1
       \c__scontents_hidden_space_str \s__scontents_stop #2 \s__scontents_mark #3
  }  {
       \tl_set:Ne #3
         {
           \tl_if_empty:nTF {#2}
             { \exp_not:o { \__scontents_use_delimit_by_s_stop:nw #1 } }
             { \exp_not:n {#1} }
         }
     }
```

(*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`.

```
\prg_new_protected_conditional:Npnn \__scontents_file_if_writable:n #1 { T, TF }
```

```
  {
    \bool_if:NTF \l__scontents_writing_bool
      {
        \__scontents_file_if_exist:nTF {#1}
          {
            \bool_if:NTF \l__scontents_overwrite_bool
              {
                \msg_info:nne { scontents } { overwrite-file } {#1}
                \prg_return_true:
              }
              {
                \msg_warning:nne { scontents } { not-writing } {#1}
                \prg_return_false:
              }
          }
          {
            \msg_info:nne { scontents } { writing-file } {#1}
            \prg_return_true:
          }
      }
      { \prg_return_false: }
  }
```

(*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.

```
\cs_new_protected:Npn \__scontents_file_write_cmd:nn #1 #2
  {
    \__scontents_file_if_writable:nT {#1}
      {
        \iow_open:Nn \l__scontents_file_write_iow {#1}
        \iow_now:Nn  \l__scontents_file_write_iow {#2}
        \iow_close:N \l__scontents_file_write_iow
      }
  }
\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).

```
\cs_new_protected:Npn \__scontents_set_write_out:n #1
  {
    \bool_set_false:N \l__scontents_storing_bool
    \bool_set_true:N \l__scontents_writing_bool
    \tl_set:Nn \l__scontents_file_name_tl {#1}
  }
\cs_new_protected:Npn \__scontents_set_write_store:n #1
  {
    \__scontents_set_write_out:n {#1}
    \bool_set_true:N \l__scontents_storing_bool
  }
```

(*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.

```
\cs_new:Ne \__scontents_tab: { \c_space_tl }
\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.

```
\group_begin:
  \char_set_catcode_active:N \^^@
  \char_set_lccode:nn { 0 } { 32 }
  \tex_lowercase:D
    {
      \group_end:
      \cs_new:Npn \__scontents_tabs_to_spaces:
        { \prg_replicate:nn { \l__scontents_tab_width_int } { ^^@ } }
    }
```

(*End of definition for* __scontents_tabs_to_spaces:.)

__scontents_do_noligs:N

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

```
\cs_new_protected:Npn \__scontents_do_noligs:N #1
  {
    \char_set_catcode_active:N #1
    \cs_set:cpe { __scontents_active_char_ \token_to_str:N #1 : }
      {
        \mode_leave_vertical:
        \tex_kern:D \c_zero_dim
        \tex_char:D `\exp_not:N #1
      }
    \char_set_active_eq:Nc #1 { __scontents_active_char_ \token_to_str:N #1 : }
  }
```

(*End of definition for* __scontents_do_noligs:N.)

__scontents_set_active_eq:NN
__scontents_plain_disable_outer_par:
__scontents_make_control_chars_active:

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

```
\cs_new_protected:Npn \__scontents_set_active_eq:NN #1
  {
    \char_set_catcode_active:N #1
    \char_set_active_eq:NN #1
  }
⟨/core⟩
⟨*loader⟩
\group_begin:
⟨plain⟩  \char_set_catcode_active:n { `\* }
  \cs_new_protected:Nn \__scontents_plain_disable_outer_par:
⟨*plain⟩
    {
      \group_begin:
        \char_set_lccode:nn { `\* } { `\^^L }
        \tex_lowercase:D { \group_end:
        \tex_let:D * \scan_stop:
      }
    }
⟨/plain⟩
⟨latex|context⟩    { }
\group_end:
⟨/loader⟩
⟨*core⟩
\cs_new_protected:Nn \__scontents_make_control_chars_active:
  {
    \__scontents_plain_disable_outer_par:
    \__scontents_set_active_eq:NN \^^I \__scontents_tab:
    \__scontents_set_active_eq:NN \^^L \__scontents_par:
    \__scontents_set_active_eq:NN \^^M \__scontents_ret:w
  }
```

(*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:
__scontents_esphack:

We emulate \@bsphack and \@esphack for plain TeX and ConTeXt, this is necessary to prevent *unwanted spaces* when the print-cmd key is false. In LaTeX the kernel definitions are used.

```
\cs_new_protected:Nn \__scontents_bsphack:
```

```
  {
    \scan_stop:
    \mode_if_horizontal:T
      {
        \skip_set_eq:NN \l__scontents_save_skip \tex_lastskip:D
        \int_set_eq:NN \l__scontents_save_sf_int \tex_spacefactor:D
      }
  }
\cs_new_protected:Nn \__scontents_esphack:
  {
    \scan_stop:
    \mode_if_horizontal:T
      {
        \int_set_eq:NN \tex_spacefactor:D \l__scontents_save_sf_int
        \dim_compare:nNnT { \l__scontents_save_skip } > { \c_zero_skip }
          {
            \skip_if_eq:nnT { \tex_lastskip:D } { \c_zero_skip }
              {
                \nobreak
                \skip_horizontal:n { \c_zero_skip }
              }
            \tex_ignorespaces:D
          }
      }
  }
⟨/core⟩
⟨*latex⟩
\cs_gset_eq:NN \__scontents_bsphack: \@bsphack
\cs_gset_eq:NN \__scontents_esphack: \@esphack
⟨/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 {⟨*body env*⟩} of an environment or the {⟨*argument*⟩} of a command, passed in `#2`, to the right of the *sequence*.

```
⟨*core⟩
\cs_new_protected:Npn \__scontents_append_contents:nn #1 #2
  {
    \seq_if_exist:cF { g__scontents_name_#1_seq }
      { \seq_new:c { g__scontents_name_#1_seq } }
    \seq_gput_right:cn { g__scontents_name_#1_seq } {#2}
  }
\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`.

```
\cs_new_protected:Npn \__scontents_store_to_seq:NN #1 #2
  {
    \bool_if:NT \l__scontents_debug_var_bool
      {
        \tl_log:N #1
      }
    \__scontents_append_contents:Ve #2 { \exp_not:V #1 }
  }
```

(*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, `\c__scontents_hidden_space_str` is added to the end of the token list passed in `#1`, which contains the {⟨*body env*⟩} of an environment or the {⟨*argument*⟩} 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.

```
\cs_new_protected:Npn \__scontents_finish_storing:NNN #1 #2 #3
  {
    \bool_if:NT \l__scontents_storing_bool
      {
        \tl_if_blank:VTF #2
          { \msg_error:nn { scontents } { empty-store-name } }
          {
            \bool_if:NF \l__scontents_forced_eol_bool
              { \tl_put_right:Ne #1 { \c__scontents_hidden_space_str } }
            \__scontents_store_to_seq:NN #1 #2
            \bool_if:NT #3 { \__scontents_lastfrom_seq:V #2 }
          }
      }
  }
```

(*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 TEX and ConTEXt, only the current *sequence* exists.

```
\prg_new_conditional:Npnn \__scontents_seq_if_exist:n #1 { TF }
  {
    \bool_lazy_or:nnTF
      { \seq_if_exist_p:c { g__scontents_name_#1_seq } }
      { \seq_if_exist_p:c { g__scontents_prev_#1_seq } }
      { \prg_return_true: }
      { \prg_return_false: }
  }
\cs_new:Npn \__scontents_seq_count:n #1
  {
    \int_max:nn
      {
        \seq_if_exist:cTF { g__scontents_name_#1_seq }
          { \seq_count:c { g__scontents_name_#1_seq } }
          { 0 }
      }
      {
        \seq_if_exist:cTF { g__scontents_prev_#1_seq }
          { \seq_count:c { g__scontents_prev_#1_seq } }
          { 0 }
      }
  }
\cs_new:Npn \__scontents_seq_item:nn #1 #2
  {
    \bool_lazy_and:nnTF
      { \seq_if_exist_p:c { g__scontents_prev_#1_seq } }
      {
        \int_compare_p:nNn { \int_abs:n {#2} }
          < { \seq_count:c { g__scontents_prev_#1_seq } + 1 }
      }
      { \seq_item:cn { g__scontents_prev_#1_seq } {#2} }
      { \seq_item:cn { g__scontents_name_#1_seq } {#2} }
  }
```

(*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:nn` 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).

```
\cs_new:Npn \__scontents_getfrom_seq:Nn #1 #2
```

©2019–2026 by Pablo González L

```
  {
    \__scontents_seq_if_exist:nTF {#2}
      {
        \exp_args:Nf \__scontents_getfrom_seq:nNn
          { \__scontents_seq_count:n {#2} } #1 {#2}
      }
      { \msg_expandable_error:nnn { scontents } { undefined-storage } {#2} }
  }
\cs_new:Npn \__scontents_getfrom_seq:nNn #1 #2 #3
  {
    \seq_map_tokens:Nn #2 { \__scontents_getfrom_seq_aux:nnn {#1} {#3} }
  }
\cs_new:Npn \__scontents_getfrom_seq_aux:nnn #1 #2 #3
  {
    \exp_args:Nnf \use:n
      { \__scontents_getfrom_seq:nnn {#1} } { \int_eval:n {#3} } {#2}
  }
\cs_new:Npn \__scontents_getfrom_seq:nn #1 #2
  {
    \__scontents_seq_if_exist:nTF {#2}
      {
        \exp_args:Nf \__scontents_getfrom_seq:nnn
          { \__scontents_seq_count:n {#2} }
          {#1} {#2}
      }
      { \msg_expandable_error:nnn { scontents } { undefined-storage } {#2} }
  }
\cs_new:Npn \__scontents_getfrom_seq:nnn #1 #2 #3
  {
    \bool_lazy_or:nnTF
      { \int_compare_p:nNn {#2} = { 0 } }
      { \int_compare_p:nNn { \int_abs:n {#2} } > {#1} }
      {
        \msg_expandable_error:nnnnn { scontents } { index-out-of-range }
          {#2} {#3} {#1}
      }
      { \__scontents_seq_item:nn {#3} {#2} }
  }
```

(*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 `\__scontents_retokenize:V` after the group, when the keys `print-env` or `print-cmd` are active.

```
\cs_new_protected:Npn \__scontents_lastfrom_seq:n #1
  {
    \tl_gset:Ne \g__scontents_last_stored_tl
      {
        \seq_item:cn { g__scontents_name_#1_seq } { -1 }
      }
    \group_insert_after:N \__scontents_retokenize:V
    \group_insert_after:N \g__scontents_last_stored_tl
    \group_insert_after:N \tl_gclear:N
    \group_insert_after:N \g__scontents_last_stored_tl
  }
\cs_generate_variant:Nn \__scontents_lastfrom_seq:n { V }
⟨/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 LaTeX.

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.

```
⟨*core⟩
\seq_new:N  \g__scontents_remember_seq
\bool_new:N \g__scontents_remember_pending_bool
⟨/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.

```
⟨*latex⟩
\ior_new:N \g__scontents_remember_ior
\iow_new:N \g__scontents_remember_iow
\str_new:N \g__scontents_remember_old_str
\str_new:N \l__scontents_remember_new_str
\str_new:N \l__scontents_remember_name_str
\str_new:N \l__scontents_remember_item_str
⟨/latex⟩
```

(*End of definition for* \g__scontents_remember_ior *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 TEX or ConTEXt.

```
⟨*core⟩
\msg_new:nnnn { scontents } { remember-latex-only }
  { The~key~'remember'~is~only~available~in~LaTeX~\msg_line_context:. }
  {
    The~stored~contents~can~only~be~used~after~they~are~stored~
    in~this~format.~The~key~is~ignored.
  }
⟨/core⟩
```

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

13.12.3 Definition of messages for remembering contents

remember-rerun
remember-undefined

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.

```
⟨*latex⟩
\msg_new:nnn { scontents } { remember-rerun }
  { Stored~contents~may~have~changed.~Rerun~to~get~them~right. }
\msg_new:nnn { scontents } { remember-undefined }
  { There~were~undefined~stored~contents. }
⟨/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 TEX and ConTEXt it raises the "error" remember-latex-only.

```
⟨*core⟩
\cs_new_protected:Npn \__scontents_remember:n #1
  {
    \__scontents_format_case:nnn
      {
        \clist_map_inline:nn {#1}
          {
            \seq_if_in:NeF \g__scontents_remember_seq { \tl_to_str:n {##1} }
              { \seq_gput_right:Ne \g__scontents_remember_seq { \tl_to_str:n {##1} } }
```

©2019–2026 by Pablo González L

```
      }
    }
    { \msg_error:nn { scontents } { remember-latex-only } }
    { \msg_error:nn { scontents } { remember-latex-only } }
  }
```

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

`\__scontents_if_pending:nTF`
`\__scontents_if_pending:neF`

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.

```
\prg_new_protected_conditional:Npnn \__scontents_if_pending:nn #1 #2 { T, F }
  {
    \bool_lazy_and:nnTF
      { \seq_if_exist_p:c { g__scontents_prev_#1_seq } }
      { \int_compare_p:nNn { \int_abs:n {#2} } > { \__scontents_seq_count:n {#1} } }
      {
        \bool_gset_true:N \g__scontents_remember_pending_bool
        \prg_return_true:
      }
      { \prg_return_false: }
  }
\prg_generate_conditional_variant:Nnn \__scontents_if_pending:nn { ne } { F }
⟨/core⟩
```

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

`\__scontents_remember_read:`
`\__scontents_remember_read_line:n`
`\__scontents_remember_read_line:w`

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.

```
⟨*latex⟩
\cs_new_protected:Npn \__scontents_remember_read:
  {
    \ior_open:NnT \g__scontents_remember_ior { \c_sys_jobname_str .sct }
      {
        \ior_str_map_inline:Nn \g__scontents_remember_ior
          { \__scontents_remember_read_line:n {##1} }
        \ior_close:N \g__scontents_remember_ior
      }
    \seq_map_inline:Nn \g__scontents_remember_seq
      {
        \seq_if_exist:cF { g__scontents_prev_##1_seq }
          { \seq_new:c { g__scontents_prev_##1_seq } }
      }
  }
\cs_new_protected:Npn \__scontents_remember_read_line:n #1
  {
    \str_if_empty:nF {#1}
      { \__scontents_remember_read_line:w #1 = = \s__scontents_stop }
  }
\cs_new_protected:Npn \__scontents_remember_read_line:w #1 = #2 = #3 \s__scontents_stop
  {
    \str_set_convert:Nnnn \l__scontents_remember_name_str {#1} { utf8/hex } { }
    \seq_if_in:NVT \g__scontents_remember_seq \l__scontents_remember_name_str
      {
        \str_gput_right:Nn \g__scontents_remember_old_str { #1 = #2 , }
        \str_set_convert:Nnnn \l__scontents_remember_item_str {#2} { utf8/hex } { }
        \seq_if_exist:cF { g__scontents_prev_ \l__scontents_remember_name_str _seq }
          { \seq_new:c { g__scontents_prev_ \l__scontents_remember_name_str _seq } }
        \seq_gput_right:cV { g__scontents_prev_ \l__scontents_remember_name_str _seq }
          \l__scontents_remember_item_str
      }
  }
```

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

©2019–2026 by Pablo González L

__scontents_remember_write:
__scontents_remember_write_seq:n
__scontents_remember_write_item:n

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.

```
\cs_new_protected:Npn \__scontents_remember_write:
  {
    \seq_if_empty:NF \g__scontents_remember_seq
      {
        \str_clear:N \l__scontents_remember_new_str
        \iow_open:Nn \g__scontents_remember_iow { \c_sys_jobname_str .sct }
        \seq_map_function:NN \g__scontents_remember_seq \__scontents_remember_write_seq:n
        \iow_close:N \g__scontents_remember_iow
        \str_if_eq:NNTF \g__scontents_remember_old_str \l__scontents_remember_new_str
          {
            \bool_if:NT \g__scontents_remember_pending_bool
              { \msg_warning:nn { scontents } { remember-undefined } }
          }
          { \msg_warning:nn { scontents } { remember-rerun } }
      }
  }
\cs_new_protected:Npn \__scontents_remember_write_seq:n #1
  {
    \seq_if_exist:cT { g__scontents_name_#1_seq }
      {
        \str_set_convert:Nnnn \l__scontents_remember_name_str {#1} { } { utf8/hex }
        \seq_map_function:cN { g__scontents_name_#1_seq }
          \__scontents_remember_write_item:n
      }
  }
\cs_new_protected:Npn \__scontents_remember_write_item:n #1
  {
    \str_set_convert:Nnnn \l__scontents_remember_item_str {#1} { } { utf8/hex }
    \iow_now:Ne \g__scontents_remember_iow
      { \l__scontents_remember_name_str = \l__scontents_remember_item_str }
    \str_put_right:Ne \l__scontents_remember_new_str
      { \l__scontents_remember_name_str = \l__scontents_remember_item_str , }
  }
```

(*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.

```
\hook_gput_code:nnn { begindocument } { scontents }
  { \__scontents_remember_read: }
\hook_gput_code:nnn { enddocument } { scontents }
  { \__scontents_remember_write: }
⟨/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 {⟨*env name*⟩}, and an optional argument with the ⟨*initial keys*⟩. 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 {⟨*body env*⟩} 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 ⟨*keys*⟩ 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.

```
⟨*core⟩
\tl_new:N  \l__scontents_save_every_body_lines_tl
\tl_new:N  \l__scontents_processed_body_lines_tl
\tl_new:N  \l__scontents_environment_keys_tl
\int_new:N \l__scontents_nesting_env_int
\int_new:N \l__scontents_nesting_aux_int
```

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

©2019–2026 by Pablo González L

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 LaTeX, \#1 and \end#1 in plain TeX and \start#1 and \stop#1 in ConTeXt. They are used in the messages of the environments.

```
\cs_new:Npn \__scontents_env_begin_str:n #1
  {
    \__scontents_format_case:nnn
      { \c_backslash_str begin \c_left_brace_str #1 \c_right_brace_str }
      { \c_backslash_str #1 }
      { \c_backslash_str start #1 }
  }
\cs_new:Npn \__scontents_env_end_str:n #1
  {
    \__scontents_format_case:nnn
      { \c_backslash_str end \c_left_brace_str #1 \c_right_brace_str }
      { \c_backslash_str end #1 }
      { \c_backslash_str stop #1 }
  }
```

(*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.

```
\msg_new:nnnn { scontents } { junk-after-begin }
  {
    Junk~characters~#1~\msg_line_context: :
    \\ \\
    #2
  }
  {
    Nothing~is~allowed~after~the~start~of~the~environment~on~the~same~
    line.~These~characters~are~ignored.
  }
\msg_new:nnnn { scontents } { env-already-defined }
  { Environment~'#1'~already~defined! }
  {
    You~have~used~\token_to_str:N \newenvsc \c_space_tl
    with~an~environment~that~already~has~a~definition. \\ \\
    The~existing~definition~of~'#1'~will~not~be~altered.
  }
```

(*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.

```
\msg_new:nnn { scontents } { empty-environment }
  { Environment~'#1'~is~empty~\msg_line_context:. }
\msg_new:nnn { scontents } { rescanning-text }
  {
    Rescanning~text~'#1'~after~\__scontents_env_end_str:n {#2}~
    \msg_line_context:.
  }
\msg_new:nnn { scontents } { multiple-begin }
  {
    Multiple~\__scontents_env_begin_str:n { \l__scontents_env_name_tl }~
    \msg_line_context:.
  }
```

(*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).

```
\keys_define:nn { scontents / scontents }
  {
    write-env .code:n           = { \__scontents_set_write_store:n {#1} },
    write-out .code:n           = { \__scontents_set_write_out:n {#1} },
    write-env .value_required:n = true,
    write-out .value_required:n = true,
    print-env .meta:nn          = { scontents } { print-env = #1 },
    print-env .default:n        = true,
    store-env .meta:nn          = { scontents } { store-env = #1 },
    force-eol .meta:nn          = { scontents } { force-eol = #1 },
    force-eol .default:n        = true,
    overwrite .meta:nn          = { scontents } { overwrite = #1 },
    overwrite .default:n        = true,
    unknown   .code:n           = { \__scontents_unknown_keys_env:n {#1} },
  }
```

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

13.13.5 Internal functions for \newenvsc

__scontents_grab_opt_arg:w
__scontents_grab_opt_arg:n

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 ltcmd[11] to use its delimited argument processor.

```
\NewDocumentCommand \__scontents_grab_opt_arg:w { r[] }
  {
    \__scontents_grab_opt_arg:n {#1}
  }
```

The function __scontents_grab_opt_arg:n is called where '^^M' is active, so __scontents_normalise_line_ends:N replaces the active '^^M' characters by spaces before setting the ⟨*keys*⟩.

```
\cs_new_protected:Npn \__scontents_grab_opt_arg:n #1
  {
    \tl_set:Nn \l__scontents_environment_keys_tl {#1}
    \__scontents_normalise_line_ends:N \l__scontents_environment_keys_tl
    \keys_set:nV { scontents / scontents } \l__scontents_environment_keys_tl
    \__scontents_start_after_option:w
  }
```

(*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_environment:n
__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 removes the spaces at the beginning and at the end of the rest of the line of \begin{scontents} and __scontents_start_environment:n checks if it starts with 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.

```
\group_begin:
  \char_set_catcode_active:N \^^I
  \char_set_catcode_active:N \^^L
  \char_set_catcode_active:N \^^M
  \cs_new_protected:Npn \__scontents_normalise_line_ends:N #1
    { \tl_replace_all:Nnn #1 { ^^M } { ~ } }
  \cs_new_protected:Npn \__scontents_start_environment:w #1 ^^M
    { \tl_trim_spaces_apply:nN {#1} \__scontents_start_environment:n }
  \cs_new_protected:Npn \__scontents_start_environment:n #1
    {
      \tl_if_head_is_N_type:nTF {#1}
        {
          \str_if_eq:eeTF { \tl_head:n {#1} } { [ }
```

```
            { \__scontents_grab_opt_arg:w #1 ^^M }
            { \__scontents_check_line_process:nn { } {#1} }
        }
        { \__scontents_check_line_process:nn { } {#1} }
    }
  \cs_new_protected:Npn \__scontents_start_after_option:w #1 ^^M
    { \__scontents_check_line_process:nn { [...] } {#1} }
  \cs_new_protected:Npn \__scontents_check_line_process:nn #1 #2
    {
      \tl_if_blank:nF {#2}
        {
          \msg_error:nnee { scontents } { junk-after-begin }
            { after~ \__scontents_env_begin_str:n { \l__scontents_env_name_tl } #1 } {#2}
        }
      \__scontents_make_control_chars_active:
      \__scontents_file_tl_write_start:V \l__scontents_file_name_tl
    }
  \cs_new_protected:Nn \__scontents_stop_environment:
    {
      \__scontents_file_write_stop:N \l__scontents_processed_body_lines_tl
      \bool_lazy_and:nnT
        { \l__scontents_storing_bool }
        { \tl_if_empty_p:N \l__scontents_processed_body_lines_tl }
        {
          \msg_warning:nne { scontents } { empty-environment }
            { \l__scontents_env_name_tl }
        }
    }
```

(*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.

```
  \cs_new_protected:Npn \__scontents_file_tl_write_start:n #1
    {
      \group_begin:
        \__scontents_file_if_writable:nTF {#1}
          {
            \bool_set_true:N \l__scontents_writable_bool
            \iow_open:Nn \l__scontents_file_write_iow {#1}
          }
          { \bool_set_false:N \l__scontents_writable_bool }
        \tl_clear:N \l__scontents_save_every_body_lines_tl
        \seq_map_function:NN \l_char_special_seq \char_set_catcode_other:N
        \int_step_function:nnnN { 128 } { 1 } { 255 } \char_set_catcode_letter:n
        \cs_set_protected:Npe \__scontents_ret:w ##1 ^^M
          {
            \exp_not:N \__scontents_verb_processor_iterate:w
            ##1 \c__scontents_end_env_tl
                \c__scontents_end_env_tl
                \exp_not:N \q__scontents_stop
          }
        \__scontents_make_control_chars_active:
        \__scontents_ret:w
    }
  \cs_new_protected:Nn \__scontents_setup_verb_processor:
    {
```

```
      \use:e
        {
          \cs_set_protected:Npn \exp_not:N \__scontents_verb_processor_iterate:w
            ##1 \c__scontents_end_env_tl
            ##2 \c__scontents_end_env_tl
            ##3 \exp_not:N \q__scontents_stop
          }   { \__scontents_verb_processor_iterate:nnn {##1} {##2} {##3} }
    }
  \cs_new_protected:Npn \__scontents_verb_processor_iterate:nnn #1 #2 #3
    {
      \tl_if_blank:nTF {#3}
        {
          \__scontents_analyse_nesting:n {#1}
          \__scontents_verb_processor_output:n {#1}
        }
        {
          \__scontents_if_nested:TF
            {
              \__scontents_nesting_decr:
              \__scontents_verb_processor_output:e
                { \exp_not:n {#1} \c__scontents_end_env_tl \exp_not:n {#2} }
            }
            {
              \tl_if_blank:nF {#1}
                { \__scontents_verb_processor_output:n {#1} }
              \cs_set_protected:Npe \__scontents_ret:w
                {
                  \__scontents_env_end_function:
                  \bool_lazy_or:nnF
                    { \tl_if_blank_p:n {#2} }
                    { \str_if_eq_p:ee {#2} { \c_percent_str } }
                    {
                      \str_if_eq:VnF \c__scontents_hidden_space_str {#2}
                        {
                          \msg_warning:nnnn { scontents } { rescanning-text }
                            {#2} { \tl_use:N \l__scontents_env_name_tl }
                        }
                      \__scontents_retokenize:n {#2}
                    }
                }
              \char_set_active_eq:NN ^^M \__scontents_ret:w
            }
        }
      ^^M
    }
  \cs_new:Nn \__scontents_env_end_function:
    {
      \__scontents_format_case:nnn
        { \exp_not:N \end { \if_false: } \fi: }
        { \exp_after:w \exp_not:N \cs:w end }
        { \exp_after:w \exp_not:N \cs:w stop }
      \tl_use:N \l__scontents_env_name_tl
      \__scontents_format_case:nnn
        { \if_false: { \fi: } }
        { \cs_end: }
        { \cs_end: }
    }
  \cs_new_protected:Npn \__scontents_file_write_stop:N #1
    {
      \bool_if:NT \l__scontents_writable_bool
        { \iow_close:N \l__scontents_file_write_iow }
      \use:e
        {
          \group_end:
          \bool_if:NT \l__scontents_storing_bool
            {
              \tl_set:Nn \exp_not:N #1
                {
                  \exp_args:NV
                    \__scontents_remove_leading_nl:n \l__scontents_save_every_body_lines_tl
                }
```

```
            }
        }
    }
  \cs_new:Npn \__scontents_remove_leading_nl:n #1
    {
      \tl_if_head_is_N_type:nTF {#1}
        {
          \exp_args:Nf
            \__scontents_remove_leading_nl:nn
              { \tl_head:n {#1} } {#1}
        }
        { \exp_not:n {#1} }
    }
  \cs_new:Npn \__scontents_remove_leading_nl:nn #1 #2
    {
      \token_if_eq_meaning:NNTF ^^J #1
        { \exp_not:o { \__scontents_remove_leading_nl:w #2 } }
        { \exp_not:n {#2} }
    }
  \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 `\l__scontents_writable_bool` and `\l__scontents_storing_bool`.

```
  \cs_new_protected:Npn \__scontents_verb_processor_output:n #1
    {
      \bool_if:NT \l__scontents_writable_bool
        { \iow_now:Nn \l__scontents_file_write_iow {#1} }
      \bool_if:NT \l__scontents_storing_bool
        { \tl_put_right:Nn \l__scontents_save_every_body_lines_tl { ^^J #1 } }
    }
\group_end:
\cs_generate_variant:Nn \__scontents_verb_processor_output:n { e }
\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 `\l__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.

```
\cs_new_protected:Npn \__scontents_analyse_nesting:n #1
  {
    \int_zero:N \l__scontents_nesting_aux_int
    \__scontents_analyse_nesting_format:n {#1}
    \int_compare:nNnT { \l__scontents_nesting_aux_int } > { 1 }
      { \msg_warning:nn { scontents } { multiple-begin } }
  }
\cs_new_protected:Nn \__scontents_nesting_incr:
  {
    \int_incr:N \l__scontents_nesting_env_int
    \int_incr:N \l__scontents_nesting_aux_int
  }
\cs_new_protected:Nn \__scontents_nesting_decr:
  {
    \int_decr:N \l__scontents_nesting_env_int
  }
\prg_new_protected_conditional:Npnn \__scontents_if_nested: { TF }
  {
    \int_compare:nNnTF { \l__scontents_nesting_env_int } > { \c_zero_int }
      { \prg_return_true: }
      { \prg_return_false: }
  }
```

(*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
ontents_analyse_nesting_generic_process:nn
contents_define_generic_nesting_function:n
__scontents_analyse_nesting_format:n

In 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}`.

```
\use:e
  {
    \cs_new_protected:Npn \exp_not:N \__scontents_analyse_nesting_latex:w #1
      \c_backslash_str begin \c_left_brace_str #2 \c_right_brace_str
  }  {
      \__scontents_tl_if_head_is_q_mark:nTF {#2}
        { \__scontents_use_none_delimit_by_q_stop:w }
        {
          \str_if_eq:VnT \l__scontents_env_name_tl {#2}
            { \__scontents_nesting_incr: }
          \__scontents_analyse_nesting_latex:w
        }
    }
\cs_new_protected:Npe \__scontents_analyse_nesting_latex:n #1
  {
    \__scontents_analyse_nesting_latex:w #1
      \c_backslash_str begin
        \c_left_brace_str \exp_not:N \q__scontents_mark \c_right_brace_str
    \exp_not:N \q__scontents_stop
  }
```

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`.

```
\cs_new_protected:Npn \__scontents_analyse_nesting_generic_process:nn #1 #2
  {
    \tl_if_head_is_N_type:nTF {#2}
      {
        \__scontents_tl_if_head_is_q_mark:nF {#2}
          {
            \__scontents_nesting_incr:
            \__scontents_analyse_nesting_generic:w #2 \q__scontents_stop
          }
      }
      { \__scontents_analyse_nesting_generic:w #2 \q__scontents_stop }
  }
\cs_new_protected:Npn \__scontents_analyse_nesting_generic:nn #1 #2
  {
    \__scontents_define_generic_nesting_function:n {#1}
    \use:e
      {
        \exp_not:N \__scontents_analyse_nesting_generic:w #2
          \c_backslash_str #1 \tl_use:N \l__scontents_env_name_tl
            \exp_not:N \q__scontents_mark \exp_not:N \q__scontents_stop
      }
  }
\cs_new_protected:Npn \__scontents_define_generic_nesting_function:n #1
  {
    \use:e
      {
        \cs_set_protected:Npn \exp_not:N \__scontents_analyse_nesting_generic:w ##1
          \c_backslash_str #1 \tl_use:N \l__scontents_env_name_tl
            ##2 \exp_not:N \q__scontents_stop
      }  { \__scontents_analyse_nesting_generic_process:nn {##1} {##2} }
  }
⟨/core⟩
```

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

```
⟨*loader⟩
⟨latex⟩\cs_new_eq:NN \__scontents_analyse_nesting_format:n
⟨latex⟩  \__scontents_analyse_nesting_latex:n
⟨!latex⟩\cs_new_protected:Npn \__scontents_analyse_nesting_format:n
⟨plain⟩  { \__scontents_analyse_nesting_generic:nn { } }
⟨context⟩  { \__scontents_analyse_nesting_generic:nn { start } }
⟨/loader⟩
```

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

__scontents_generic_begin:
__scontents_generic_end:

The function `\__scontents_generic_begin:` leaves the '^^M' character active and calls `\__scontents_start_environment:w`. The function `\__scontents_generic_end:` calls `\__`

©2019–2026 by Pablo González L

`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).

```
⟨*core⟩
\cs_new_protected:Npn \__scontents_generic_begin:
  {
    \char_set_catcode_active:N \^^M
    \__scontents_start_environment:w
  }
\cs_new_protected:Npn \__scontents_generic_end:
  {
    \__scontents_stop_environment:
    \__scontents_finish_storing:NNN
      \l__scontents_processed_body_lines_tl
      \l__scontents_name_seq_env_tl
      \l__scontents_print_env_bool
  }
```

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

`\__scontents_setting_env:nn` `\__scontents_define_env:nnn`

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`.

```
\cs_new_protected:Npn \__scontents_setting_env:nn #1 #2
  {
    \cs_new_protected:cpn { __scontents_#1_begin: }
      {
        \tl_set:Nn \l__scontents_env_name_tl {#1}
        \tl_set:Nn \l__scontents_cmd_name_tl { newenvsc }
        \keys_set:nn { scontents } {#2}
        \__scontents_setup_verb_processor:
        \__scontents_generic_begin:
      }
    \cs_new_protected:cpn { __scontents_#1_end: }
      { \__scontents_generic_end: }
    \exp_args:Nooo \__scontents_define_env:nnn
      { \tl_to_str:n {#1} }
      { \cs:w __scontents_#1_begin: \cs_end: }
      { \cs:w __scontents_#1_end: \cs_end: }
  }
⟨/core⟩
```

The function `\__scontents_define_env:nnn` creates the environment for LaTeX, plain TeX and ConTeXt.

```
⟨*loader⟩
\cs_new_protected:Npn \__scontents_define_env:nnn #1 #2 #3
  {
⟨latex|plain⟩    \NewDocumentEnvironment {#1} { }
⟨context⟩    \cs_new_protected:cpn { start #1 }
      {
⟨!latex⟩        \group_begin:
          #2
      }
⟨context⟩    \cs_new_protected:cpn { stop #1 }
      {
          #3
⟨!latex⟩        \group_end:
      }
  }
```

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

`\newenvsc`

Defines the user command `\newenvsc` for LaTeX, plain TeX and ConTeXt, raising the "error" `env-already-defined` if the environment exists.

```
\NewDocumentCommand \newenvsc { m O{} }
  {
⟨latex|plain⟩    \cs_if_exist:cTF {#1}
⟨context⟩    \cs_if_exist:cTF { start #1 }
      { \msg_error:nnn { scontents } { env-already-defined } {#1} }
      { \__scontents_setting_env:nn {#1} {#2} }
```

©2019–2026 by Pablo González L

```
  }
⟨/loader⟩
```

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

13.14 The environment `scontents`

`scontents` `\scontents` `\endscontents` `\startscontents` `\stopscontents`

The base environment `scontents` is defined with `\newenvsc`, in plain TEX it is `\scontents` ... `\endscontents` and in ConTEXt `\startscontents` ... `\stopscontents`.

```
⟨*loader⟩
\newenvsc{scontents}
⟨/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 LATEX used by `\typestored` (§13.20). For plain TEX and ConTEXt we must add some additional functions.

13.15.1 Definition of variables for `verbatimsc`

`\l__scontents_plain_bool`

The boolean variable `\l__scontents_plain_bool` is used in the plain TEX version of the `verbatimsc` environment.

```
⟨*plain⟩
\bool_new:N \l__scontents_plain_bool
⟨/plain⟩
```

(*End of definition for* `\l__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`.

```
⟨*!latex⟩
\cs_if_exist:NF \dospecials
  {
    \cs_new:Npn \dospecials
      { \seq_map_function:NN \l_char_special_seq \do }
  }
⟨/!latex⟩
```

(*End of definition for* `\dospecials`.)

`\__scontents_xverb:w` `\__scontents_xverb:`

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 TEX and LATEX. The function `\__scontents_xverb:` makes the TAB character active, equal to `\__scontents_tabs_to_spaces:`, and calls `\__scontents_xverb:w`.

```
⟨*loader⟩
⟨*!context⟩
\use:e
  {
    \cs_new_protected:Npn \exp_not:N \__scontents_xverb:w
      #1 \g__scontents_end_verbatimsc_tl
⟨latex⟩       { #1 \exp_not:N \end{verbatimsc} }
⟨plain⟩       { #1 \exp_not:N \endverbatimsc }
  }
⟨/!context⟩
⟨/loader⟩
⟨*core⟩
\cs_new_protected:Nn \__scontents_xverb:
  {
    \char_set_catcode_active:n { 9 }
    \char_set_active_eq:nN { 9 } \__scontents_tabs_to_spaces:
    \__scontents_xverb:w
  }
⟨/core⟩
```

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

©2019–2026 by Pablo González L

13.15.3 plain TEX version of verbatimsc

__scontents_verbatimsc_aux:
__scontents_nolig_list:
__scontents_vobeyspaces:
__scontents_xobeysp:

In plain TEX we emulate the verbatim environment of LATEX.

```
⟨*plain⟩
\cs_new_protected:Nn \__scontents_verbatimsc_aux:
  {
    \skip_vertical:N \parskip
    \dim_zero:N \parindent
    \skip_set:Nn \parfillskip { 0pt plus 1fil }
    \skip_set:Nn \parskip { 0pt plus0pt minus0pt }
    \tex_par:D
    \bool_set_false:N \l__scontents_plain_bool
    \cs_set:Npn \par
      {
        \bool_if:NTF \l__scontents_plain_bool
          {
            \mode_leave_vertical:
            \null
            \tex_par:D
            \penalty \interlinepenalty
          }
          {
            \bool_set_true:N \l__scontents_plain_bool
            \mode_if_horizontal:T
              { \tex_par:D \penalty \interlinepenalty }
          }
      }
    \cs_set_eq:NN \do \char_set_catcode_other:N
    \dospecials \obeylines
    \tl_use:N \l__scontents_verb_font_tl
    \cs_set_eq:NN \do \__scontents_do_noligs:N
    \__scontents_nolig_list:
    \tex_everypar:D \exp_after:w
      { \tex_the:D \tex_everypar:D \tex_unpenalty:D }
  }
\cs_new_protected:Nn \__scontents_nolig_list:
  { \do\`\do\<\do\>\do\,\do\'\do\- }
\cs_new_protected:Nn \__scontents_vobeyspaces:
  { \__scontents_set_active_eq:NN \  \__scontents_xobeysp: }
\cs_new_protected:Nn \__scontents_xobeysp:
  { \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.

```
\group_begin:
  \char_set_catcode_active:N \^^M
  \cs_new_protected:Npn \__scontents_verbatimsc_line:w #1 ^^M
    {
      \__scontents_verbatimsc_aux: \frenchspacing \__scontents_vobeyspaces:
      \__scontents_xverb:
    }
\group_end:
```

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

\verbatimsc
\endverbatimsc

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

```
\cs_new_protected:Npn \verbatimsc
  {
    \group_begin:
      \char_set_catcode_active:N \^^M
      \__scontents_verbatimsc_line:w
  }
\cs_new_protected:Npn \endverbatimsc
  { \group_end: }
⟨/plain⟩
```

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

©2019–2026 by Pablo González L

13.15.4 ConTEXt version of verbatimsc

\startverbatimsc
\stopverbatimsc

In ConTEXt we use its own tool \definetyping.

```
⟨*loader⟩
⟨context⟩\definetyping[verbatimsc]
⟨/loader⟩
```

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

13.15.5 LATEX version of verbatimsc

__scontents_verbatimsc_instance:

To be compatible with *tagged* PDF the environment verbatimsc is defined with the xtemplate[11] module of the LATEX 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).

```
⟨*loader⟩
⟨*latex⟩
\cs_new_protected:Nn \__scontents_verbatimsc_instance:
  {
    \DeclareInstance{blockenv}{verbatimsc}{std}
      {
        name                  = verbatimsc,
        tag-name              = \UseStructureName{block/verbatim},
        tagging-recipe        = standard,
        tagging-suppress-paras = true,
        block-instance        = verbatim,
        para-instance         = justify,
        final-code            = \legacyverbatimsetup{invisible},
      }
  }
\hook_gput_code:nnn { begindocument } { scontents }
  {
    \IfDocumentMetadataTF { \__scontents_verbatimsc_instance: } { }
  }
```

(*End of definition for* __scontents_verbatimsc_instance:.)

verbatimsc

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

```
\NewDocumentEnvironment { verbatimsc } { }
  {
    \IfDocumentMetadataTF
      {
        \UseInstance{blockenv}{verbatimsc}{}
        \@setupverbinvisiblespace\frenchspacing\@vobeyspaces
        \__scontents_xverb:
      }
      {
        \cs_set_eq:cN { @xverbatim } \__scontents_xverb:
        \verbatim
      }
  }
  {
    \IfDocumentMetadataTF
      {
        \BlockEnvEnd
      }
      { \endverbatim }
  }
⟨/latex⟩
⟨/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 {⟨*argument*⟩} in a *sequence*. It takes an optional *starred argument*, an optional argument with the ⟨*keys*⟩ and the {⟨*argument*⟩}, 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 {⟨argument⟩} passed to \Scontents.

```
⟨*core⟩
\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 {⟨argument⟩} of \Scontents is empty.

```
\msg_new:nnn { scontents } { empty-argument-command }
  { 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

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

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

```
\keys_define:nn { scontents / Scontents }
  {
    write-cmd .code:n           = { \__scontents_set_write_store:n {#1} },
    write-out .code:n           = { \__scontents_set_write_out:n {#1} },
    write-cmd .value_required:n = true,
    write-out .value_required:n = true,
    print-cmd .meta:nn          = { scontents } { print-cmd = #1 },
    print-cmd .default:n        = true,
    store-cmd .meta:nn          = { scontents } { store-cmd = #1 },
    force-eol .meta:nn          = { scontents } { force-eol = #1 },
    force-eol .default:n        = true,
    overwrite .meta:nn          = { scontents } { overwrite = #1 },
    overwrite .default:n        = true,
    unknown   .code:n           = { \__scontents_unknown_keys_cmd:n {#1} },
  }
```

(*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.

```
\cs_new_protected:Npn \__scontents_Scontents_code:nn #1 #2
  {
    \__scontents_bsphack:
    \group_begin:
      \keys_set:nn { scontents / Scontents } {#2}
      \char_set_catcode_active:n { 9 }
      \bool_if:NTF #1
        { \__scontents_Scontents_verb_arg:w }
        { \__scontents_Scontents_norm_arg:n }
  }
```

(*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 \l__scontents_Scontents_arg_tl and replaces every '^^M' by '^^J'. In LaTeX it also replaces \obeyedline, added in LaTeX release 2024-06-01. The command \obeyedline is redefined with \RenewDocumentCommand because it can be modified by the user, which would return a low-level error. In ConTeXt \obeyedline is a command of the format, protected in LMTX, and is not changed.

```
\NewDocumentCommand \__scontents_Scontents_verb_arg:w { +v }
  {
    \tl_set:Nn \l__scontents_Scontents_arg_tl {#1}
    \__scontents_format_case:nnn
      {
        \cs_if_exist:NT \obeyedline
          {
            \RenewDocumentCommand \obeyedline { } { \iow_char:N \^^M }
```

©2019–2026 by Pablo González L

```
            \tl_replace_all:Nee \l__scontents_Scontents_arg_tl
              { \obeyedline } { \iow_char:N \^^M }
          }
      }
      { }
      { }
    \tl_replace_all:NeV \l__scontents_Scontents_arg_tl
      { \iow_char:N \^^M } \c__scontents_newline_tl
    \__scontents_Scontents_finish:
  }
```

(*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.

```
\cs_new_protected:Npn \__scontents_Scontents_norm_arg:n #1
  {
    \tl_set:Nn \l__scontents_Scontents_arg_tl {#1}
    \__scontents_Scontents_finish:
  }
```

(*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 {⟨*argument*⟩} 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.

```
\cs_new_protected:Nn \__scontents_Scontents_finish:
  {
    \tl_if_empty:NT \l__scontents_Scontents_arg_tl
      { \msg_warning:nn { scontents } { empty-argument-command } }
    \__scontents_file_write_cmd:VV \l__scontents_file_name_tl \l__scontents_Scontents_arg_tl
    \__scontents_finish_storing:NNN
      \l__scontents_Scontents_arg_tl \l__scontents_name_seq_cmd_tl \l__scontents_print_cmd_bool
    \use:e
      {
        \group_end:
        \bool_if:NF \l__scontents_print_cmd_bool { \__scontents_esphack: }
      }
  }
```

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

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

```
\NewDocumentCommand \Scontents { !s !O{} }
  {
    \tl_set:Nn \l__scontents_cmd_name_tl { Scontents }
    \__scontents_Scontents_code:nn {#1} {#2}
  }
```

(*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 ⟨*index*⟩, the last ⟨*stored content*⟩ by default, and a mandatory argument with the {⟨*seq name*⟩}. It is robust.

13.17.1 Internal functions for `\getstored`

`\__scontents_getstored:nn` The function `\__scontents_getstored:nn` rescans with `\__scontents_retokenize:e` the ⟨*stored content*⟩ at the ⟨*index*⟩ 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).

```
\cs_new_protected:Npn \__scontents_getstored:nn #1 #2
  {
    \__scontents_if_pending:neF {#2} { \int_eval:n {#1} }
      {
        \__scontents_retokenize:e
          {
            \__scontents_getfrom_seq:nn {#1} {#2}
          }
      }
  }
```

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

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

```
\NewDocumentCommand \getstored { O{-1} m }
  {
    \__scontents_getstored:nn {#1} {#2}
  }
```

(*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 ⟨*stored contents*⟩ of a *sequence*. It takes an optional argument with the ⟨*keys*⟩ and a mandatory argument with the {⟨*seq name*⟩}, raising the "error" `undefined-storage` if the *sequence* does not exist.

13.18.1 Definition of variables for \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`

The token list `\l__scontents_foreachsc_arg_tl` saves the {⟨*seq name*⟩}, the sequence `\l__scontents_foreach_print_seq` collects the code for each ⟨*stored content*⟩, 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 ⟨*stored content*⟩, `\getstored` by default. The other variables are set by the ⟨*keys*⟩ of `\foreachsc`.

```
\tl_new:N   \l__scontents_foreachsc_arg_tl
\seq_new:N  \l__scontents_foreach_print_seq
\tl_new:N   \g__scontents_foreach_exec_tl
\tl_new:N   \l__scontents_foreach_before_tl
\bool_new:N \l__scontents_foreach_before_bool
\tl_new:N   \l__scontents_foreach_after_tl
\bool_new:N \l__scontents_foreach_after_bool
\int_new:N  \l__scontents_foreach_start_int
\int_new:N  \l__scontents_foreach_step_int
\int_new:N  \l__scontents_foreach_stop_int
\bool_new:N \l__scontents_foreach_stop_bool
\bool_new:N \l__scontents_foreach_wrapper_bool
\tl_new:N   \l__scontents_foreach_body_tl
\tl_new:N   \l__scontents_foreach_cmd_tl
\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

`before` `after` `start` `stop` `step` `wrapper` `sep` `unknown`

We define the ⟨*keys*⟩ for `\foreachsc`, the keys `start`, `step` and `stop` use `\__scontents_set_positive_int:Nnn` (§13.8) and the unknown ⟨*keys*⟩ are passed to `\__scontents_unknown_keys_cmd:n` (§13.9).

```
\keys_define:nn { scontents / foreachsc }
  {
    before  .code:n           = {
                                  \bool_set_true:N \l__scontents_foreach_before_bool
                                  \tl_set:Nn \l__scontents_foreach_before_tl {#1}
                                },
    before  .value_required:n = true,
    after   .code:n           = {
                                  \bool_set_true:N \l__scontents_foreach_after_bool
                                  \tl_set:Nn \l__scontents_foreach_after_tl {#1}
                                },
    after   .value_required:n = true,
    start   .code:n           = {
                                  \__scontents_set_positive_int:Nnn
                                    \l__scontents_foreach_start_int { start } {#1}
                                },
    start   .value_required:n = true,
    start   .initial:n        = 1,
    stop    .code:n           = {
                                  \__scontents_set_positive_int:Nnn
                                    \l__scontents_foreach_stop_int { stop } {#1}
                                  \bool_set:Nn \l__scontents_foreach_stop_bool
                                    {
                                      \int_compare_p:nNn
```

```
                                        { \l__scontents_foreach_stop_int } > { 0 }
                                    }
                                },
    stop     .value_required:n = true,
    step     .code:n           = {
                                    \__scontents_set_positive_int:Nnn
                                      \l__scontents_foreach_step_int { step } {#1}
                                },
    step     .value_required:n = true,
    step     .initial:n        = 1,
    wrapper  .code:n           = {
                                    \bool_set_true:N \l__scontents_foreach_wrapper_bool
                                    \cs_set_protected:Npn
                                      \__scontents_foreach_wrapper:n ##1 {#1}
                                },
    wrapper  .value_required:n = true,
    sep      .tl_set:N         = \l__scontents_foreach_sep_tl,
    sep      .initial:n        = { },
    sep      .value_required:n = true,
    unknown  .code:n           = { \__scontents_unknown_keys_cmd:n {#1} },
  }
```

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

getstored
typestored
meaningsc

The ⟨*keys*⟩ `getstored`, `typestored` and `meaningsc` set the command saved in `\l__scontents_foreach_cmd_tl`.

```
\keys_define:nn { scontents / foreachsc }
  {
    getstored  .code:n            =
      { \tl_set:Nn \l__scontents_foreach_cmd_tl { \getstored } },
    getstored  .value_forbidden:n = true,
    typestored .code:n            =
      { \tl_set:Nn \l__scontents_foreach_cmd_tl { \typestored } },
    typestored .value_forbidden:n = true,
    meaningsc  .code:n            =
      { \tl_set:Nn \l__scontents_foreach_cmd_tl { \meaningsc } },
    meaningsc  .value_forbidden:n = true,
  }
```

(*End of definition for* `getstored`, `typestored`, *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.

```
\cs_new_protected:Npn \__scontents_foreachsc:nn #1 #2
  {
    \group_begin:
      \tl_if_novalue:nF {#1} { \keys_set:nn { scontents / foreachsc } {#1} }
      \tl_set:Nn \l__scontents_foreachsc_arg_tl {#2}
      \seq_clear:N \l__scontents_foreach_print_seq
      \bool_if:NF \l__scontents_foreach_stop_bool
        {
          \int_set:Nn \l__scontents_foreach_stop_int
            { \__scontents_seq_count:n {#2} }
        }
      \int_step_function:nnnN
        { \l__scontents_foreach_start_int }
        { \l__scontents_foreach_step_int }
        { \l__scontents_foreach_stop_int }
        \__scontents_foreach_add_body:n
      \tl_gset:Ne \g__scontents_foreach_exec_tl
        {
          \exp_args:NNV \seq_use:Nn
```

©2019–2026 by Pablo González L

```
            \l__scontents_foreach_print_seq \l__scontents_foreach_sep_tl
        }
    \group_end:
    \exp_after:w \tl_gclear:N
    \exp_after:w \g__scontents_foreach_exec_tl
    \g__scontents_foreach_exec_tl
  }
\cs_new_protected:Npn \__scontents_foreach_add_body:n #1
  {
    \tl_set:Ne \l__scontents_foreach_body_tl
      {
        \exp_not:V \l__scontents_foreach_cmd_tl [#1]
          { \exp_not:V \l__scontents_foreachsc_arg_tl }
      }
    \bool_if:NT \l__scontents_foreach_wrapper_bool
      { \__scontents_foreach_wrap:V \l__scontents_foreach_body_tl }
    \seq_put_right:Ne \l__scontents_foreach_print_seq
      {
        \bool_if:NT \l__scontents_foreach_before_bool
          { \exp_not:V \l__scontents_foreach_before_tl }
        \exp_not:V \l__scontents_foreach_body_tl
        \bool_if:NT \l__scontents_foreach_after_bool
          { \exp_not:V \l__scontents_foreach_after_tl }
      }
  }
\cs_new_protected:Npn \__scontents_foreach_wrap:n #1
  { \tl_set:No \l__scontents_foreach_body_tl { \__scontents_foreach_wrapper:n {#1} } }
\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`.

```
\NewDocumentCommand \foreachsc { +o m }
  {
    \tl_set:Nn \l__scontents_cmd_name_tl { foreachsc }
    \__scontents_seq_if_exist:nTF {#2}
      { \__scontents_foreachsc:nn {#1} {#2} }
      { \msg_error:nnn { scontents } { undefined-storage } {#2} }
  }
```

(*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 ⟨*index*⟩, ⟨*1-end*⟩ or the range ⟨*start-stop*⟩ of the ⟨*stored contents*⟩ in the *sequence* along with the ⟨*keys*⟩, this avoids the awkward `\typestored[...][`⟨*keys*⟩`]{...}` syntax. The ⟨*indexes*⟩ and ranges are processed as unknown ⟨*keys*⟩ of `l3keys`[16, 17].

13.19.1 Definition of variables for the optional argument

`\l__scontents_current_seq_name_str` `\l__scontents_index_str` `\l__scontents_range_start_tl` `\l__scontents_range_stop_tl`

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 ⟨*index*⟩ only has digits and the token lists `\l__scontents_range_start_tl` and `\l__scontents_range_stop_tl` save the limits of a range ⟨*start-stop*⟩.

```
\str_new:N \l__scontents_current_seq_name_str
\str_new:N \l__scontents_index_str
\tl_new:N  \l__scontents_range_start_tl
\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

`width-tab` `write-out` `overwrite` `print-cmd` `unknown`

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

```
\keys_define:nn { scontents / typemeaning }
```

©2019–2026 by Pablo González L

```
  {
    width-tab .meta:nn          = { scontents } { width-tab = #1 },
    write-out .code:n           = { \__scontents_set_write_out:n {#1} },
    write-out .value_required:n = true,
    overwrite .meta:nn          = { scontents } { overwrite = #1 },
    overwrite .default:n        = true,
    print-cmd .bool_set:N       = \l__scontents_print_verb_style_bool,
    print-cmd .initial:n        = true,
    print-cmd .default:n        = true,
    unknown   .code:n           = { \__scontents_parse_type_meaning_key:n {#1} },
  }
```

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

13.19.3 Internal functions for the optional argument

`\__scontents_parse_type_meaning_key:n`
`\__scontents_parse_type_meaning_key:nn`
`\__scontents_range_parser:nnn`
`\__scontents_range_parser:nnnw`
`\__scontents_range_limit:nn`
`\__scontents_range_step:nn`
`\__scontents_if_digits:nTF`

The function `\__scontents_parse_type_meaning_key:n` checks that the unknown ⟨*key*⟩ 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`.

```
\cs_new_protected:Npn \__scontents_parse_type_meaning_key:n #1
  { \exp_args:NV \__scontents_parse_type_meaning_key:nn \l_keys_key_str {#1} }
\cs_new_protected:Npn \__scontents_parse_type_meaning_key:nn #1 #2
  {
    \tl_if_blank:nTF {#2}
      {
        \__scontents_range_parser:nnn {#1}
          { \__scontents_seq_count:n { \l__scontents_current_seq_name_str } }
          { \msg_error:nnn { scontents } { cmd-key-unknown } }
      }
      { \msg_error:nnnn { scontents } { cmd-key-value-unknown } {#1} {#2} }
  }
\prg_new_protected_conditional:Npnn \__scontents_if_digits:n #1 { TF }
  {
    \str_set:Ne \l__scontents_index_str { \tl_trim_spaces:n {#1} }
    \clist_map_inline:nn { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }
      { \str_remove_all:Nn \l__scontents_index_str {##1} }
    \tl_if_blank:nTF {#1}
      { \prg_return_false: }
      {
        \str_if_empty:NTF \l__scontents_index_str
          { \prg_return_true: }
          { \prg_return_false: }
      }
  }
\prg_generate_conditional_variant:Nnn \__scontents_if_digits:n { V } { TF }
\cs_new_protected:Npn \__scontents_range_parser:nnn #1 #2 #3
  { \__scontents_range_parser:nnnw {#1} {#2} {#3} #1 - \q__scontents_mark - \s__scontents_mark }
\cs_new_protected:Npn \__scontents_range_parser:nnnw #1 #2 #3 #4 - #5 - #6 \s__scontents_mark
  {
    \__scontents_tl_if_head_is_q_mark:nTF {#5}
      {
        \__scontents_if_digits:nTF {#4}
          { \seq_put_right:Ne \l__scontents_seq_item_seq { \int_eval:n {#4} } }
          { #3 {#1} }
      }
      {
        \__scontents_tl_if_head_is_q_mark:nTF {#6}
          {
            \tl_if_blank:nTF {#4}
              {
                \__scontents_if_digits:nTF {#5}
```

```
                    {
                      \seq_put_right:Ne \l__scontents_seq_item_seq
                        { \int_eval:n { - #5 } }
                    }
                    { #3 {#1} }
                }
                {
                  \tl_set:Ne \l__scontents_range_start_tl
                    { \__scontents_range_limit:nn {#4} {#2} }
                  \tl_set:Ne \l__scontents_range_stop_tl
                    { \__scontents_range_limit:nn {#5} {#2} }
                  \__scontents_if_digits:VTF \l__scontents_range_start_tl
                    {
                      \__scontents_if_digits:VTF \l__scontents_range_stop_tl
                        {
                          \exp_args:NVV \__scontents_range_step:nn
                            \l__scontents_range_start_tl \l__scontents_range_stop_tl
                        }
                        { #3 {#1} }
                    }
                    { #3 {#1} }
                }
            }
            { #3 {#1} }
        }
    }
\cs_new:Npn \__scontents_range_limit:nn #1 #2
    {
      \str_if_eq:eeTF { \tl_trim_spaces:n {#1} } { end }
        {#2}
        { \tl_trim_spaces:n {#1} }
    }
\cs_new_protected:Npn \__scontents_range_step:nn #1 #2
    {
      \int_step_inline:nnnn {#1}
        { \int_compare:nNnTF {#2} < {#1} { -1 } { 1 } }
        {#2}
        { \seq_put_right:Nn \l__scontents_seq_item_seq {##1} }
    }
```

(*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).

```
\bool_new:N \l__scontents_pending_bool
\cs_new_protected:Npn \__scontents_typemeaning_setup:nn #1 #2
    {
      \seq_clear:N \l__scontents_seq_item_seq
      \str_set:Ne \l__scontents_current_seq_name_str {#2}
      \tl_if_novalue:nF {#1}
        { \keys_set:nn { scontents / typemeaning } {#1} }
      \seq_if_empty:NT \l__scontents_seq_item_seq
        { \seq_set_from_clist:Nn \l__scontents_seq_item_seq { 1 } }
      \bool_set_false:N \l__scontents_pending_bool
      \seq_map_inline:Nn \l__scontents_seq_item_seq
        {
          \__scontents_if_pending:nnT {#2} {##1}
            { \bool_set_true:N \l__scontents_pending_bool }
        }
    }
```

(*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.

```
\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.

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

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

13.20.3 Internal functions for \typestored

__scontents_typestored:nn The function __scontents_typestored:nn sets the ⟨*keys*⟩ and the ⟨*indexes*⟩ passed in #1 with __scontents_typemeaning_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_typestored:N and writes them with __scontents_file_write_cmd:VV.

```
\cs_new_protected:Npn \__scontents_typestored:nn #1 #2
  {
    \__scontents_bsphack:
    \group_begin:
      \__scontents_typemeaning_setup:nn {#1} {#2}
      \bool_if:NF \l__scontents_pending_bool
        {
          \tl_set:Ne \l__scontents_typestored_arg_tl
            { \__scontents_getfrom_seq:Nn \l__scontents_seq_item_seq {#2} }
          \__scontents_remove_trailing_eol:N \l__scontents_typestored_arg_tl
          \tl_replace_all:NeV \l__scontents_typestored_arg_tl
            { \c__scontents_hidden_space_str } \c__scontents_newline_tl
          \bool_if:NT \l__scontents_debug_var_bool
            {
              \tl_log:N \l__scontents_typestored_arg_tl
            }
          \tl_if_empty:NF \l__scontents_typestored_arg_tl
            {
              \bool_if:NT \l__scontents_print_verb_style_bool
                {
                  \__scontents_typestored:N \l__scontents_typestored_arg_tl
                }
            }
          \__scontents_file_write_cmd:VV \l__scontents_file_name_tl \l__scontents_typestored_arg_
        }
      \use:e
        {
          \group_end:
          \bool_if:NF \l__scontents_print_verb_style_bool { \__scontents_esphack: }
        }
  }
```

(*End of definition for* __scontents_typestored:nn.)

__scontents_typestored:N __scontents_verb_print_EOL: The function __scontents_typestored: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 __scontents_retokenize:e.

```
\group_begin:
  \char_set_catcode_active:N \^^M
  \cs_new_protected:Npn \__scontents_typestored:N #1
    {
      \tl_if_blank:VT #1
        { \msg_error:nn { scontents } { empty-variable } }
      \cs_set_eq:NN \__scontents_verb_print_EOL: ^^M
      \cs_set_eq:NN ^^M \scan_stop:
```

©2019–2026 by Pablo González L

```
      \cs_set_eq:cN { do@noligs } \__scontents_do_noligs:N
      \int_set:Nn \tex_newlinechar:D { `\^^J }
      \__scontents_retokenize:e
        {
          \__scontents_format_case:nnn
            { \exp_not:N \begin{verbatimsc} } % LaTeX
            { \verbatimsc } % Plain/Generic
            { \startverbatimsc } % ConTeXt
            ^^M
          \exp_not:V #1 ^^M
          \g__scontents_end_verbatimsc_tl
        }
      \cs_set_eq:NN ^^M \__scontents_verb_print_EOL:
    }
\group_end:
```

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

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

```
\NewDocumentCommand \typestored { o m }
  {
    \tl_set:Nn \l__scontents_cmd_name_tl { typestored }
    \__scontents_typestored:nn {#1} {#2}
  }
```

(*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.

```
\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`.

```
\cs_new_protected:Npn \__scontents_meaningsc:nn #1 #2
  {
    \__scontents_bsphack:
    \group_begin:
      \__scontents_typemeaning_setup:nn {#1} {#2}
      \bool_if:NF \l__scontents_pending_bool { \__scontents_meaningsc:n {#2} }
      \use:e
        {
          \group_end:
          \bool_if:NF \l__scontents_print_verb_style_bool { \__scontents_esphack: }
        }
  }
```

(*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 as a string with `\tl_to_str:N` and writes them with `\__scontents_file_write_cmd:VV`. The string is made with `\tl_to_str:N` and not with `\cs_replacement_spec:N` because in ConTEXt LMTX the `\meaning` of LuaMetaTeX writes one space for several consecutive spaces.

```
\group_begin:
  \char_set_catcode_active:N \^^I
  \cs_new_protected:Npn \__scontents_meaningsc:n #1
    {
      \tl_set:Ne \l__scontents_meaningsc_arg_tl
```

©2019–2026 by Pablo González L

```
        { \__scontents_getfrom_seq:Nn \l__scontents_seq_item_seq {#1} }
      \tl_replace_all:Nen \l__scontents_meaningsc_arg_tl
        { \iow_char:N \^^J } { ~ }
      \tl_replace_all:Nen \l__scontents_meaningsc_arg_tl
        { \c__scontents_hidden_space_str } { ~ }
      \bool_if:NT \l__scontents_debug_var_bool
        {
          \tl_log:N \l__scontents_meaningsc_arg_tl
        }
      \tl_use:N \l__scontents_verb_font_tl
      \tl_replace_all:Nne \l__scontents_meaningsc_arg_tl
        { ^^I } { \prg_replicate:nn { \l__scontents_tab_width_int } { ~ } }
      \tl_replace_all:Nee \l__scontents_meaningsc_arg_tl
        { \iow_char:N \^^I }
        { \prg_replicate:nn { \l__scontents_tab_width_int } { ~ } }
      \tl_if_empty:NF \l__scontents_meaningsc_arg_tl
        {
          \bool_if:NT \l__scontents_print_verb_style_bool
            {
              \tl_to_str:N \l__scontents_meaningsc_arg_tl
            }
        }
      \__scontents_file_write_cmd:VV \l__scontents_file_name_tl \l__scontents_meaningsc_arg_tl
    }
\group_end:
```

(*End of definition for* `\__scontents_meaningsc:n`.)

`\meaningsc` Defines the user command `\meaningsc`.

```
\NewDocumentCommand \meaningsc { o m }
  {
    \tl_set:Nn \l__scontents_cmd_name_tl { meaningsc }
    \__scontents_meaningsc:nn {#1} {#2}
  }
```

(*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`

`\l__scontents_mergesc_arg_tl` `\l__scontents_mergesc_keys_tl` 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`.

```
\tl_new:N \l__scontents_mergesc_arg_tl
\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`.

```
\msg_new:nnnn { scontents } { mergesc-missing-key }
  {
    Missing~mandatory~key~'typestored',~'meaningsc'~or~'getstored'~
    for~\c_backslash_str mergesc~\msg_line_context:.
  }
  {
    The~command~\c_backslash_str mergesc~needs~one~of~the~keys~
    'typestored',~'meaningsc'~or~'getstored'.\\
    Check~that~you~have~spelled~the~key~name~correctly.
  }
```

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

13.22.3 Definition of keys for \mergesc

typestored
meaningsc
getstored
__scontents_mergesc_cmd:nn

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.

```
\keys_define:nn { scontents / mergesc }
  {
    typestored .code:n            =
      { \cs_set_eq:NN \__scontents_mergesc_cmd:nn \__scontents_typestored:nn },
    typestored .value_forbidden:n = true,
    meaningsc  .code:n            =
      { \cs_set_eq:NN \__scontents_mergesc_cmd:nn \__scontents_meaningsc:nn },
    meaningsc  .value_forbidden:n = true,
    getstored  .code:n            =
      { \cs_set_eq:NN \__scontents_mergesc_cmd:nn \__scontents_mergesc_getstored:nn },
    getstored  .value_forbidden:n = true,
  }
\cs_new_protected:Npn \__scontents_mergesc_cmd:nn #1 #2
  {
    \msg_error:nn { scontents } { mergesc-missing-key }
  }
```

(*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 \l__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.

```
\cs_new_protected:Npn \__scontents_mergesc_code:nn #1 #2
  {
    \group_begin:
      \tl_clear:N \l__scontents_mergesc_keys_tl
      \tl_if_novalue:nF {#1}
        {
          \keys_set_known:nnN { scontents / mergesc } {#1}
            \l__scontents_mergesc_keys_tl
        }
      \seq_gclear:c { g__scontents_name_sc!internal_seq }
      \__scontents_mergesc_parse_list:n {#2}
      \seq_if_empty:cF { g__scontents_name_sc!internal_seq }
        {
          \exp_args:Ne \__scontents_mergesc_cmd:nn
            { 1-end, \exp_not:V \l__scontents_mergesc_keys_tl } { sc!internal }
        }
    \group_end:
  }
```

(*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.

```
\cs_new_protected:Npn \__scontents_mergesc_parse_list:n #1
  {
    \clist_map_inline:nn {#1} { \__scontents_mergesc_parse_item:n {##1} }
    \seq_if_empty:cF { g__scontents_name_sc!internal_seq }
      {
        \seq_gpop_right:cN { g__scontents_name_sc!internal_seq } \l__scontents_mergesc_arg_tl
        \__scontents_remove_trailing_eol:N \l__scontents_mergesc_arg_tl
        \seq_gput_right:cV { g__scontents_name_sc!internal_seq } \l__scontents_mergesc_arg_tl
      }
  }
```

©2019–2026 by Pablo González L

```
\cs_new_protected:Npn \__scontents_mergesc_parse_item:n #1
  {
    \tl_if_in:nnTF {#1} { [ } % ]
      { \__scontents_parse_mergesc:nw #1 \s__scontents_stop }
      { \__scontents_parse_mergesc:nw {#1} \s__scontents_stop }
  }
\cs_new_protected:Npn \__scontents_parse_mergesc:nw #1
  {
    \__scontents_seq_if_exist:nTF {#1}
      {
        \peek_remove_spaces:n
          {
            \peek_charcode:NTF [ % ]
              { \__scontents_parse_mergesc_aux:nw {#1} }
              { \__scontents_parse_mergesc_aux:nw {#1} [ 1-end ] }
          }
      }
      {
        \msg_error:nnn { scontents } { undefined-storage } {#1}
        \__scontents_use_none_delimit_by_s_stop:w
      }
  }
\cs_new_protected:Npn \__scontents_parse_mergesc_aux:nw #1 [#2]
  {
    \seq_clear:N \l__scontents_seq_item_seq
    \clist_map_inline:nn {#2}
      {
        \__scontents_range_parser:nnn {##1} { \__scontents_seq_count:n {#1} }
          { \msg_error:nnn { scontents } { cmd-key-unknown } }
      }
    \seq_map_inline:Nn \l__scontents_seq_item_seq
      { \__scontents_mergesc_put_item:nn {#1} {##1} }
    \__scontents_use_none_delimit_by_s_stop:w
  }
\cs_new_protected:Npn \__scontents_mergesc_put_item:nn #1 #2
  {
    \__scontents_if_pending:nnF {#1} {#2}
      {
        \bool_lazy_or:nnTF
          { \int_compare_p:nNn {#2} = { 0 } }
          {
            \int_compare_p:nNn { \int_abs:n {#2} }
              > { \__scontents_seq_count:n {#1} }
          }
          {
            \msg_error:nneee { scontents } { index-out-of-range }
              {#2} {#1} { \__scontents_seq_count:n {#1} }
          }
          {
            \seq_gput_right:ce { g__scontents_name_sc!internal_seq }
              { \__scontents_seq_item:nn {#1} {#2} }
          }
      }
  }
```

(*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 `\__scontents_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).

```
\keys_define:nn { scontents / getstored }
  {
    unknown .code:n = { \__scontents_unknown_keys_cmd:n {#1} },
  }
\cs_new_protected:Npn \__scontents_mergesc_getstored:nn #1 #2
  {
    \keys_set:nV { scontents / getstored } \l__scontents_mergesc_keys_tl
```

©2019–2026 by Pablo González L

```
    \tl_gset:Ne \g__scontents_last_stored_tl
      {
        \exp_args:NnV \seq_use:cn { g__scontents_name_#2_seq } \c__scontents_newline_tl
        \c__scontents_hidden_space_str
      }
    \group_insert_after:N \__scontents_retokenize:V
    \group_insert_after:N \g__scontents_last_stored_tl
    \group_insert_after:N \tl_gclear:N
    \group_insert_after:N \g__scontents_last_stored_tl
  }
```

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

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

```
\NewDocumentCommand \mergesc { o m }
  {
    \tl_set:Nn \l__scontents_cmd_name_tl { mergesc }
    \__scontents_mergesc_code:nn {#1} {#2}
  }
```

(*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).

```
\NewDocumentCommand \setupsc { +m }
  {
    \tl_set:Nn \l__scontents_cmd_name_tl { setupsc }
    \keys_set:nn { scontents } {#1}
  }
```

(*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.

```
\NewExpandableDocumentCommand \countsc { m }
  {
    \__scontents_seq_if_exist:nTF {#1}
      { \__scontents_seq_count:n {#1} }
      {
        \msg_expandable_error:nnn { scontents } { undefined-storage } {#1}
        0
      }
  }
```

(*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*.

```
\NewDocumentCommand \cleanseqsc { m }
  {
    \seq_gclear_new:c { g__scontents_name_#1_seq }
  }
⟨/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` `\l__scontents_map_item_tl` 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*⟩.

```
⟨*core⟩
\int_new:N \g__scontents_map_int
\tl_new:N  \l__scontents_map_item_tl
```

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

©2019–2026 by Pablo González L

13.26.2 Public functions of the programming interface

\scontents_if_exist_p:n
\scontents_if_exist:nTF
\scontents_count:n

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

```
\prg_new_conditional:Npnn \scontents_if_exist:n #1 { p, T, F, TF }
  {
    \__scontents_seq_if_exist:nTF {#1}
      { \prg_return_true: }
      { \prg_return_false: }
  }
\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
__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).

```
\cs_new_protected:Npn \scontents_get:nnN #1 #2 #3
  { \exp_args:Nne \__scontents_get:nnN {#1} { \int_eval:n {#2} } #3 }
\cs_new_protected:Npn \__scontents_get:nnN #1 #2 #3
  {
    \tl_clear:N #3
    \__scontents_seq_if_exist:nTF {#1}
      {
        \__scontents_if_pending:nnF {#1} {#2}
          {
            \bool_lazy_or:nnTF
              { \int_compare_p:nNn {#2} = { 0 } }
              {
                \int_compare_p:nNn { \int_abs:n {#2} }
                  > { \__scontents_seq_count:n {#1} }
              }
              {
                \msg_error:nneee { scontents } { index-out-of-range }
                  {#2} {#1} { \__scontents_seq_count:n {#1} }
              }
              {
                \tl_set:Ne #3 { \__scontents_seq_item:nn {#1} {#2} }
                \__scontents_remove_trailing_eol:N #3
              }
          }
      }
      { \msg_error:nnn { scontents } { undefined-storage } {#1} }
  }
```

(*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.

```
\cs_new_protected:Npn \scontents_map_inline:nn #1 #2
  {
    \__scontents_seq_if_exist:nTF {#1}
      {
        \int_gincr:N \g__scontents_map_int
        \cs_gset_protected:cpn { __scontents_map_ \int_use:N \g__scontents_map_int :n }
          ##1 {#2}
        \exp_args:Nne \__scontents_map_inline:nn {#1} { \int_use:N \g__scontents_map_int }
        \int_gdecr:N \g__scontents_map_int
      }
      { \msg_error:nnn { scontents } { undefined-storage } {#1} }
  }
\cs_new_protected:Npn \__scontents_map_inline:nn #1 #2
  {
```

```
    \int_step_inline:nn { \__scontents_seq_count:n {#1} }
      {
        \__scontents_get:nnN {#1} {##1} \l__scontents_map_item_tl
        \exp_args:Nc \exp_args:NV { __scontents_map_ #2 :n } \l__scontents_map_item_tl
      }
  }
⟨/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).

```
⟨context⟩\char_set_catcode_letter:N \!
⟨plain|context⟩\ExplSyntaxOff
⟨plain|context⟩\endinput
⟨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

after 1336

B

before 1336

C

\cleanseqsc *9*, *61*, 1820
cmd-key-unknown 293
cmd-key-value-unknown 293
Commands provided by SCONTENTS:
\Scontents *50*
\cleanseqsc *61*
\countsc *61*
\foreachsc *53*
\getstored *51*
\meaningsc *58*
\mergesc *61*
\newenvsc *45*
\setupsc *61*
\typestored *57*
\countsc *8*, *61*, 1811

D

debug-var 228
\dospecials 1112

E

empty-argument-command 1239
empty-environment 766
empty-store-name 202
empty-variable 1561
\endscontents *4*, 1106
\endverbatimsc 1184
env-already-defined 749
env-key-unknown 274
env-key-value-unknown 274
Environments provided by SCONTENTS:
verbatimsc *47*, *48*

F

force-eol 228, 778, 1241
\foreachsc *6*, *51*, 1440

G

\getstored *6*, *50*, 1317
getstored 1381, 1686

I

index-out-of-range 184

J

junk-after-begin 749

M

\meaningsc *7*, *53*, *57*, 1669
meaningsc 1381, 1686
\mergesc *7*, *15*, *53*, *58*, 1801
mergesc-missing-key 1676
multiple-begin 766

N

\newenvsc *5*, *38*, 1098
not-writing 211

O

overwrite 228, 778, 1241, 1451
overwrite-file 211

P

Packages :
docstrip *22*
expl3 . . . . . *22*, *23*, *61*
l3doc *22*
l3keys *53*
latex-lab *48*
ltcmd *22*, *40*, *49*
lua-widow-control *24*
verbatim *48*
xparse *22*
xsim *40*
xtemplate *48*
positive-integer 202
print-all 228
print-cmd 228, 1241, 1451
print-env 228, 778

Q

quark internal commands:
\q__scontents_mark 166
\q__scontents_stop 166

R

remember 228
remember-latex-only 617
remember-rerun 625
remember-undefined 625
rescanning-text 766

S

scan internal commands:
\s__scontents_mark 166
\s__scontents_stop 166
\Scontents *5*, *48*, 1302
\scontents *4*, 1106
scontents *4*, 1106
scontents commands:
\scontents_count:n *9*, 1828
\scontents_get:nnN *9*, 1835
\scontents_if_exist:nTF *9*, 1828
\scontents_if_exist_p:n *9*, 1828
\scontents_map_inline:nn *9*, 1862
scontents internal commands:
__scontents_analyse_nesting:n 970
__scontents_analyse_nesting_format:n . 992
__scontents_analyse_nesting_generic:nn 992
__scontents_analyse_nesting_generic:w 992
__scontents_analyse_nesting_generic_-process:nn 992
__scontents_analyse_nesting_latex:n . . 992
__scontents_analyse_nesting_latex:w . . 992

©2019–2026 by Pablo González L

__scontents_append_contents:nn 490
__scontents_bsphack: 459
\l__scontents_char_value_int 12
__scontents_check_line_process:nn ... 804
\l__scontents_cmd_name_tl 170
\l__scontents_current_seq_name_str ... 1447
__scontents_define_env:nnn 1065
__scontents_define_generic_nesting_-
function:n 992
__scontents_do_noligs:N 418
\c__scontents_end_env_tl 139
\g__scontents_end_verbatimsc_tl 139
__scontents_env_begin_str:n 735
__scontents_env_end_function: 845
__scontents_env_end_str:n 735
\l__scontents_env_name_tl 139
\l__scontents_environment_keys_tl 729
__scontents_esphack: 459
__scontents_file_if_exist:nTF 103
__scontents_file_if_writable:nTF 363
\l__scontents_file_name_tl 171
__scontents_file_tl_write_start:n ... 845
__scontents_file_write_cmd:nn 386
\l__scontents_file_write_iow 171
__scontents_file_write_stop:N 845
__scontents_finish_storing:NNN 506
__scontents_foreach_add_body:n 1393
\l__scontents_foreach_after_bool 1321
\l__scontents_foreach_after_tl 1321
\l__scontents_foreach_before_bool 1321
\l__scontents_foreach_before_tl 1321
\l__scontents_foreach_body_tl 1321
\l__scontents_foreach_cmd_tl 1321
\g__scontents_foreach_exec_tl 1321
\l__scontents_foreach_print_seq 1321
\l__scontents_foreach_start_int 1321
\l__scontents_foreach_step_int 1321
\l__scontents_foreach_stop_bool 1321
\l__scontents_foreach_stop_int 1321
__scontents_foreach_wrap:n 1393
\l__scontents_foreach_wrapper_bool ... 1321
__scontents_foreachsc:nn 1393
\l__scontents_foreachsc_arg_tl 1321
__scontents_format_case:nnn 98
__scontents_generic_begin: 1051
__scontents_generic_end: 1051
__scontents_get:nnN 1835
__scontents_getfrom_seq:Nn 553
__scontents_getfrom_seq:nn 553
__scontents_getfrom_seq:nNn 553
__scontents_getfrom_seq:nnn 553
__scontents_getfrom_seq_aux:nnn 553
__scontents_getstored:nn 1307
__scontents_grab_opt_arg:n 793
__scontents_grab_opt_arg:w 793
\c__scontents_hidden_space_str 180
__scontents_if_digits:nTF 1463
__scontents_if_nested:TF 970
__scontents_if_pending:nnTF 645
__scontents_if_pending:nTF 645
\l__scontents_index_str 1447
\g__scontents_last_stored_tl 179
__scontents_lastfrom_seq:n 592
__scontents_make_control_chars_active: 429
__scontents_map_inline:nn 1862
\g__scontents_map_int 1825
\l__scontents_map_item_tl 1825
__scontents_meaningsc:n 1639
__scontents_meaningsc:nn 1627
\l__scontents_meaningsc_arg_tl 1626
\l__scontents_mergesc_arg_tl 1674
__scontents_mergesc_cmd:nn 1686
__scontents_mergesc_code:nn 1702
__scontents_mergesc_getstored:nn 1784
\l__scontents_mergesc_keys_tl 1674
__scontents_mergesc_parse_item:n 1720
__scontents_mergesc_parse_list:n 1720
__scontents_mergesc_put_item:nn 1720
\g__scontents_name_sc!internal_seq ... 177
\l__scontents_nesting_aux_int 729
__scontents_nesting_decr: 970
\l__scontents_nesting_env_int 729
__scontents_nesting_incr: 970
\c__scontents_newline_tl 117
__scontents_nolig_list: 1138
__scontents_normalise_line_ends:N ... 804
__scontents_par: 407
__scontents_parse_mergesc:nw 1720
__scontents_parse_mergesc_aux:nw 1720
__scontents_parse_type_meaning_key:n . 1463
__scontents_parse_type_meaning_key:nn 1463
\l__scontents_pending_bool 1544
\l__scontents_plain_bool 1109
__scontents_plain_disable_outer_par: . 429
\l__scontents_processed_body_lines_tl . 729
__scontents_range_limit:nn 1463
__scontents_range_parser:nnn 1463
__scontents_range_parser:nnnw 1463
\l__scontents_range_start_tl 1447
__scontents_range_step:nn 1463
\l__scontents_range_stop_tl 1447
__scontents_remember:n 631
\g__scontents_remember_ior 609
\g__scontents_remember_iow 609
\l__scontents_remember_item_str 609
\l__scontents_remember_name_str 609
\l__scontents_remember_new_str 609
\g__scontents_remember_old_str 609
\g__scontents_remember_pending_bool ... 605
__scontents_remember_read: 658
__scontents_remember_read_line:n 658
__scontents_remember_read_line:w 658
\g__scontents_remember_seq 605
__scontents_remember_write: 691
__scontents_remember_write_item:n ... 691
__scontents_remember_write_seq:n 691
__scontents_remove_leading_nl:n 845
__scontents_remove_leading_nl:nn 845
__scontents_remove_leading_nl:w 845
__scontents_remove_trailing_eol:N ... 346
__scontents_remove_trailing_eol:w ... 346

©2019–2026 by Pablo González L

__scontents_retokenize:n 118, 163
\l__scontents_save_every_body_lines_tl 729
\l__scontents_save_sf_int 182
\l__scontents_save_skip 182
\l__scontents_Scontents_arg_tl 1237
__scontents_Scontents_code:nn 1256
__scontents_Scontents_finish: 1289
__scontents_Scontents_norm_arg:n 1284
__scontents_Scontents_verb_arg:w 1266
__scontents_seq_count:n 520
__scontents_seq_if_exist:nTF 520
__scontents_seq_item:nn 520
\l__scontents_seq_item_seq 177
__scontents_set_active_eq:NN 429
__scontents_set_positive_int:Nnn 218
__scontents_set_write_out:n 396
__scontents_set_write_store:n 396
__scontents_setting_env:nn 1065
__scontents_setup_verb_processor: ... 845
__scontents_start_after_option:w 804
__scontents_start_environment:n 804
__scontents_start_environment:w 804
__scontents_stop_environment: 804
__scontents_store_to_seq:NN 498
\l__scontents_storing_bool 171
__scontents_tab: 407
\l__scontents_tab_width_int 218
__scontents_tabs_to_spaces: 409
__scontents_tl_if_head_is_q_mark:nTF . 335
__scontents_typemeaning_setup:nn 1544
__scontents_typestored:N 1598
__scontents_typestored:nn 1567
\l__scontents_typestored_arg_tl 1560
__scontents_unknown_keys_cmd:n 323
__scontents_unknown_keys_cmd:nn 323
__scontents_unknown_keys_env:n 315
__scontents_unknown_keys_env:nn 315
__scontents_use_delimit_by_s_stop:nw . 331
__scontents_use_i_delimit_by_s_stop:nw 331
__scontents_use_none_delimit_by_q_-
stop:w 331
__scontents_use_none_delimit_by_s_-
stop:w 331
__scontents_verb_print_EOL: 1598
__scontents_verb_processor_iterate:nnn 845
__scontents_verb_processor_iterate:w . 845
__scontents_verb_processor_output:n .. 960
__scontents_verbatimsc_aux: 1138
__scontents_verbatimsc_instance: 1196
__scontents_verbatimsc_line:w 1176
\c__scontents_version_str 12
__scontents_vobeyspaces: 1138
\l__scontents_writable_bool 171
\l__scontents_writing_bool 171
__scontents_xobeysp: 1138
__scontents_xverb: 1119
__scontents_xverb:w 1119
sep 1336
\setupsc 3, 61, 1806
start 1336
\startscontents 5, 1106
\startverbatimsc 8, 1193
step 1336
stop 1336
\stopscontents 5, 1106
\stopverbatimsc 8, 1193
store-all 228
store-cmd 228, 1241
store-env 228, 778

T

tl commands:
\tl_replace_all:Nnn 163
\typestored 7, 53, 55, 1621
typestored 1381, 1686

U

undefined-storage 184
unknown 228, 778, 1241, 1336, 1451

V

verb-font 228
\verbatimsc 1184
verbatimsc 8, 1215

W

width-tab 228, 1451
wrapper 1336
write-cmd 1241
write-env 778
write-out 778, 1241, 1451
writing-file 211

©2019–2026 by Pablo González L