

# cssyntax

A syntax core for readable command names

Pedagogical documentation – v0.1.2

July 17, 2026

## Contents

|           |                                                                     |          |
|-----------|---------------------------------------------------------------------|----------|
| <b>1</b>  | <b>Why this package</b>                                             | <b>1</b> |
| <b>2</b>  | <b>Naming convention</b>                                            | <b>2</b> |
| <b>3</b>  | <b>Getting started</b>                                              | <b>2</b> |
| <b>4</b>  | <b>Definition primitives</b>                                        | <b>2</b> |
| <b>5</b>  | <b>Expansion, introspection, and dynamic definition</b>             | <b>3</b> |
| 5.1       | Dynamic command definition . . . . .                                | 3        |
| <b>6</b>  | <b>Groups</b>                                                       | <b>3</b> |
| <b>7</b>  | <b>Generic helpers</b>                                              | <b>4</b> |
| <b>8</b>  | <b>Files</b>                                                        | <b>4</b> |
| <b>9</b>  | <b>Low-level environment declaration</b>                            | <b>4</b> |
| <b>10</b> | <b>Booleans</b>                                                     | <b>4</b> |
| <b>11</b> | <b>The <code>_fun_test/</code> family – native TeX conditionals</b> | <b>5</b> |
| <b>12</b> | <b>The <code>_fun_check/</code> family – declarative tests</b>      | <b>5</b> |
| 12.1      | Position-sensitive lookahead tests . . . . .                        | 6        |
| <b>13</b> | <b>Quick reference</b>                                              | <b>6</b> |

## 1 Why this package

In LaTeX, a command name can only be made of letters. As soon as you want long, structured internal names – for instance to make clear which module, which variable type, and which precise function a command belongs to – you quickly run into unreadable names: `\lwnxvarboxexlabel` tells the eye nothing, whereas `lw_nx_var_box/ex_label` tells a story (the `lw` bundle, the `nx` module, a *box*-type *variable*, concerning the *label* of an *example*).

`cssyntax` does exactly one thing: it temporarily turns a few punctuation characters (`_`, `:`, `/`) into letters, for as long as it takes to define or use this kind of structured name. It also provides a

handful of readable aliases for common TeX primitives, and a small toolkit for writing packages: dynamic command definition, low-level environment declaration, native and declarative conditional tests, and position-sensitive lookahead tests for building variadic-argument macros.

This package has no domain of its own: it knows nothing about linguistics, mathematics, or page layout. Any package author can use it to structure their own internal command names.

**Key point.** `cssyntax` is a tool *for package authors*. An end user composing a document never needs to type an `_fun_...` command directly – these names live in the internal code of packages that *use* `cssyntax`, not in the user’s document.

## 2 Naming convention

`container_module_type_subtype/description`

: and / are interchangeable separators, both always active at the same time. A project usually picks one for its new code and keeps the other only for compatibility with an already-published module.

**Key point.** Digits must never appear in a name built with `cssyntax`. They must stay recognisable as digits everywhere else (dimensions, counters, arithmetic). Use words (**zero**, **one**, **two**...) to number a series of commands.

## 3 Getting started

```
\usepackage{cssyntax}

\csSyntaxOn
% ... your structured command definitions ...
\csSyntaxOff
```

`\csSyntaxOn` activates `_`, `:` and `/` as letters. `\csSyntaxOff` restores their catcode. Neither command contains any of the three characters it activates, so both remain usable before the extended syntax is in effect. `\csSyntaxEnabled` / `\csSyntaxDisabled` are identical aliases.

**Warning.** Always close an open scope with `\csSyntaxOff` before the end of the file. A `_` left active would break subscripts in math mode ( $\$x_i\$$ ) throughout the rest of the document.

## 4 Definition primitives

| Command                                | Equivalent to                | Behaviour              |
|----------------------------------------|------------------------------|------------------------|
| <code>_fun_define/cs</code>            | <code>\def</code>            | no expansion           |
| <code>_fun_define/exp</code>           | <code>\edef</code>           | local expansion        |
| <code>_fun_define/global</code>        | <code>\gdef</code>           | global, no expansion   |
| <code>_fun_define/global_exp</code>    | <code>\xdef</code>           | global, with expansion |
| <code>_fun_define/protected_exp</code> | <code>\protected@edef</code> | protected expansion    |
| <code>_fun_copy/cs</code>              | <code>\let</code>            | copies a meaning       |

**Key point.** `_fun_copy/cs`, `_fun_save/cs`, `_fun_use/cs`, `_fun_reset/cs` and `_fun_swap/cs` keep a raw `\let` internally: they implement idempotent, repeatable operations, which the package’s own `\NewCommandCopy`-based alias declarations (see below) would break if used in their place.

## 5 Expansion, introspection, and dynamic definition

| Command                          | Equivalent to                     |                    |
|----------------------------------|-----------------------------------|--------------------|
| <code>_fun_expand/after</code>   | <code>\expandafter</code>         | reorder expansion  |
| <code>_fun_expand/prevent</code> | <code>\noexpand</code>            | protect a token    |
| <code>_fun_build/cs{name}</code> | <code>\csname...\endcsname</code> | build & invoke     |
| <code>_fun_get/cs_string</code>  | <code>\string</code>              | printable form     |
| <code>_fun_get/cs_meaning</code> | <code>\meaning</code>             | inspect definition |

### 5.1 Dynamic command definition

`_fun_build/cs` only *invokes* an existing command built from a string. `_fun_build/def` and `_fun_build/redef` *define* a new one instead.

```
\csSyntaxOn
\_fun_build/def{lw_test_greet}{#1}{Hello , #1!}
\csSyntaxOff
\lw_test_greet{World} % -> "Hello , World!"
```

| Command                                                     | Wraps                    | Collision behaviour    |
|-------------------------------------------------------------|--------------------------|------------------------|
| <code>_fun_build/def{name}&lt;param-text&gt;{body}</code>   | <code>\cs_new:cpn</code> | refuses if name exists |
| <code>_fun_build/redef{name}&lt;param-text&gt;{body}</code> | <code>\cs_set:cpn</code> | silently overwrites    |

**Warning.** The parameter-text argument must be passed *without* extra braces around it when forwarded into `\cs_new:cpn`/`\cs_set:cpn` from inside a wrapper – adding a brace group there raises *"Illegal parameter number"*.

**Key point.** `_fun_build/cs`'s underlying `\csname...\endcsname` has a side effect worth knowing: if the name was never defined, it is implicitly created as `\relax` rather than raising an error. A typo in a dynamically-built name fails silently instead of loudly.

## 6 Groups

TeX distinguishes a box-construction scope from a general-purpose local-values scope.

| Command                             | Equivalent to            | Use                                     |
|-------------------------------------|--------------------------|-----------------------------------------|
| <code>_fun_enter/group</code>       | <code>\begingroup</code> | general-purpose (default)               |
| <code>_fun_exit/group</code>        | <code>\endgroup</code>   |                                         |
| <code>_fun_enter/token_group</code> | <code>\bgroup</code>     | literal brace token, e.g. box delimiter |
| <code>_fun_exit/token_group</code>  | <code>\egroup</code>     |                                         |

```
\csSyntaxOn
\hbox\_fun_enter/token_group some text\_fun_exit/token_group
\csSyntaxOff
```

## 7 Generic helpers

| Command                                        | Role                                      |
|------------------------------------------------|-------------------------------------------|
| <code>_fun_save/cs{name}</code>                | store a command's current definition      |
| <code>_fun_use/cs{name}</code>                 | recall a stored definition                |
| <code>_fun_reset/cs{name}</code>               | reset to <code>\relax</code>              |
| <code>_fun_swap/cs{a}{b}</code>                | exchange two definitions                  |
| <code>_fun_pick/first, _fun_pick/second</code> | <code>\@firstoftwo / \@secondoftwo</code> |
| <code>_fun_gobble/one, _fun_gobble/two</code>  | <code>\@gobble / \@gobbletwo</code>       |

## 8 Files

| Command                                                  | Behaviour                                          |
|----------------------------------------------------------|----------------------------------------------------|
| <code>_fun_load/{file}</code>                            | unconditional load (alias of <code>\input</code> ) |
| <code>_fun_load/if_exists{file}{found}{not found}</code> | conditional load with both branches                |

**Warning.** `_fun_load/if_exists` is built on `\IfFileExists` combined with `\input`, *not* on `\InputIfFileExists` – the latter is a 2-argument kernel command with no built-in "not found" branch; passing it a third argument leaves it dangling in the input stream instead of raising an error.

`_fun_enter/source` (alias `_fun_enter/input`) is a no-op marker for symmetry. `_fun_exit/source` (alias `_fun_exit/input`) is a genuine `\endinput`, scoped strictly to the file it appears in.

## 9 Low-level environment declaration

Emulates `\begin/\end` at a low level, anywhere – including the preamble, where `\newenvironment`-based environments are often unsafe because they assume the main vertical list already exists.

```
\csSyntaxOn
\_fun_declare/env{myenv}{[enter] }{ [exit]}
\csSyntaxOff
\_fun_use/myenv{content} % -> "[enter] content [exit]"
```

| Command                                                 | Role                                                                     |
|---------------------------------------------------------|--------------------------------------------------------------------------|
| <code>_fun_declare/env{name}{enter}{exit}</code>        | no automatic grouping                                                    |
| <code>_fun_declare/auto_group{name}{enter}{exit}</code> | wraps enter in <code>\begingroup</code> , exit in <code>\endgroup</code> |

Each declaration also generates a matching `_fun_use/name{content}` that opens, typesets, and closes – usable identically regardless of which of the two declaration commands built it.

## 10 Booleans

Built on xifthen's `\newboolean/\setboolean/\boolean`: a name collision raises an explicit error instead of silently overwriting.

| Command                                                            | Role                      |
|--------------------------------------------------------------------|---------------------------|
| <code>_fun_declare/bool{name}</code>                               | create, initialised false |
| <code>_fun_set/bool_true</code> , <code>_fun_set/bool_false</code> | set the value             |

**Key point.** Every boolean lives under an internal `_user_bool/` prefix, protecting against collisions with the rest of the document – but not against two different bundle modules picking the same short name. Callers should still prefix their own module name, e.g. `_fun_declare/bool{nx_note_forced_margin}`.

## 11 The `_fun_test/` family – native TeX conditionals

One command opens a test, a single common command closes it, whatever kind of test was opened.

```
\csSyntaxOn
\_fun_enter/test_bool{queue_done}
  \_fun_doif/true{Done.}
  \_fun_doif/false{One more word.}
\_fun_exit/test
\csSyntaxOff
```

| Opening                                     | Equivalent to                                                             |
|---------------------------------------------|---------------------------------------------------------------------------|
| <code>_fun_enter/test_bool{name}</code>     | reads a declared boolean                                                  |
| <code>_fun_enter/test_dim{a}{rel}{b}</code> | <code>\ifdim</code> <span style="float:right">rel: &lt;,&lt;=,&gt;</span> |
| <code>_fun_enter/test_num{a}{rel}{b}</code> | <code>\ifnum</code>                                                       |
| <code>_fun_enter/test_cs{a}{b}</code>       | <code>\ifx</code>                                                         |
| <code>_fun_enter/test_case{n}</code>        | <code>\ifcase</code>                                                      |

  

| Branch                                                   | Role                                                          |
|----------------------------------------------------------|---------------------------------------------------------------|
| <code>_fun_doif/true</code>                              | executed if true                                              |
| <code>_fun_doif/false</code> (alias <code>/else</code> ) | executed otherwise (a real <code>\else</code> )               |
| <code>_fun_doif/case</code>                              | one branch of <code>\ifcase</code> (a real <code>\or</code> ) |
| <code>_fun_exit/test</code>                              | closes any of the above (a real <code>\fi</code> )            |

**Warning.** `_fun_doif/false`, `/case` and `_fun_exit/test` are genuine aliases of `\else`, `\or`, `\fi` (built with a true meaning-copy), not macros expanding to them – TeX recognises these during conditional-skipping by comparing meaning without expanding tokens it scans past. `_fun_doif/case` therefore takes no index argument: write `_fun_doif/case <code>` directly.

## 12 The `_fun_check/` family – declarative tests

These take their whole content – subject plus named `true/else` branches – in one call. There is no matching `_fun_exit/check`.

```
\csSyntaxOn
\_fun_check/pkg_loaded{hyperref}{
  true = {Loaded.},
  else = {Not loaded.}
}
\csSyntaxOff
```

| Command                                 | Role                        |
|-----------------------------------------|-----------------------------|
| <code>_fun_check/defined{cs}</code>     | is <code>cs</code> defined? |
| <code>_fun_check/pkg_loaded{pkg}</code> | is <code>pkg</code> loaded? |
| <code>_fun_check/streq{a}{b}</code>     | are the strings equal?      |

## 12.1 Position-sensitive lookahead tests

`_fun_check/if_next_char` and `_fun_check/if_next_group` inspect whatever token comes right after their own call, instead of testing a value already in hand – the building block for macros that accept a flexible, unbounded number of `{group}{group}...` arguments.

```
\csSyntaxOn
\_fun_check/if_next_group{
  true  = {,\lw_test_list_sep},
  else  = {}
}
\csSyntaxOff
```

| Command                                    | Tests                                                                           |
|--------------------------------------------|---------------------------------------------------------------------------------|
| <code>_fun_check/if_next_char{char}</code> | does this character follow? (wraps <code>\@ifnextchar</code> )                  |
| <code>_fun_check/if_next_group</code>      | does a <code>{...}</code> group follow? (wraps <code>\peek_catcode:NTF</code> ) |

**Warning.** These two are the only members of `_fun_check/` that close `cssyntax`’s own syntax scope themselves, as the very first action of their body. Do *not* write `\csSyntaxOff` between the call and the content being tested – doing so inserts `\csSyntaxOff` itself as the token being peeked at, hiding the real one. Both are defined with `\cs_new:cpn` while `\ExplSyntaxOn` is active at authoring time, not merely invoked at run time: a literal space typed in an ordinary `\def` body becomes a permanent catcode-10 token regardless of `\ExplSyntaxOn` called later, and such a token inside `\peek_catcode:NTF`’s own branch was found to make it misreport its result. A second, independent finding: `\keys_set:nn` must never be called *before* a `\peek_catcode:NTF` in the same expansion – even an unrelated call placed earlier is enough to break the peek. It is safe only when placed textually inside the peek’s own true/false branches.

## 13 Quick reference

| Family                   | Look for                                                                                         |
|--------------------------|--------------------------------------------------------------------------------------------------|
| Bootstrap                | <code>\csSyntaxOn</code> / <code>\csSyntaxOff</code>                                             |
| Definition               | <code>_fun_define/*</code> , <code>_fun_copy/cs</code>                                           |
| Expansion & dynamic def. | <code>_fun_expand/*</code> , <code>_fun_build/cs def redef</code> , <code>_fun_get/*</code>      |
| Groups                   | <code>_fun_enter exit/group</code> , <code>_fun_enter exit/token_group</code>                    |
| Helpers                  | <code>_fun_save use reset swap/cs</code> , <code>_fun_pick/*</code> , <code>_fun_gobble/*</code> |
| Files                    | <code>_fun_load/</code> , <code>_fun_load/if_exists</code> , <code>_fun_enter exit/source</code> |
| Environments             | <code>_fun_declare/env auto_group</code> , generated <code>_fun_use/*</code>                     |
| Booleans                 | <code>_fun_declare/bool</code> , <code>_fun_set/bool_true false</code>                           |
| Native tests             | <code>_fun_enter/test_* ... _fun_doif/* ... _fun_exit/test</code>                                |
| Declarative tests        | <code>_fun_check/defined pkg_loaded streq</code>                                                 |
| Lookahead tests          | <code>_fun_check/if_next_char if_next_group</code>                                               |