

Warnings and errors for the expl3 analysis tool

Vít Starý Novotný

2026-03-04

Contents

Introduction	4
1 Preprocessing	4
No standard delimiters [W100]	4
Unexpected delimiters [W101]	5
Expl3 material in non-expl3 parts [E102]	5
Line too long [S103]	6
Multiple delimiters <code>\ProvidesExpl*</code> in a single file [E104]	6
Needlessly ignored issue [S105]	6
2 Lexical analysis	6
“Do not use” argument specifiers [W200]	6
Unknown argument specifiers [E201]	7
Deprecated control sequences [W202]	7
Missing stylistic whitespaces [S204]	7
Too many closing braces [E208]	7
Invalid characters [E209]	8
3 Syntactic analysis	8
Unexpected function call argument [E300]	8
End of expl3 part within function call [E301]	8
Unbraced n-type function call argument [W302]	8
Braced N-type function call argument [W303]	9
Unexpected parameter number [E304]	9
Expanding an unexpandable variable or constant [T305]	9
4 Semantic analysis	10
4.1 Functions and conditional functions	10
Unused private function [W401]	10
Unused private function variant [W402]	10
Function variant of incompatible type [T403]	11
Protected predicate function [E404]	11
Function variant for an undefined function [E405]	11

	Calling an undefined function [E408]	12
	Function variant of deprecated type [W410]	12
	Indirect function definition from an undefined function [E411]	13
	Malformed function name [S412]	14
4.2	Variables and constants	15
	Malformed variable or constant name [S413]	15
	Malformed quark or scan mark name [S414]	16
	Unused variable or constant [W415]	16
	Setting an undeclared variable [W416]	17
	Setting a variable as a constant [E417]	17
	Setting a constant [E418]	17
	Using an undeclared variable or constant [W419]	17
	Locally setting a global variable [E420]	18
	Globally setting a local variable [E421]	18
	Using a variable of an incompatible type [T422]	19
4.3	Messages	24
	Unused message [W423]	24
	Using an undefined message [E424]	24
	Incorrect parameters in message text [E425]	24
	Incorrect number of arguments supplied to message [W426]	25
4.4	Sorting	25
	Comparison conditional without signature <code>:nnTF</code> [E427]	25
5	Flow analysis	26
5.1	Functions and conditional functions	26
	Multiply defined function [E500]	26
	Multiply defined function variant [W501]	27
	Unused private function [W502]	28
	Unused private function variant [W503]	29
	Function variant for an undefined function [E505]	29
	Calling an undefined function [E506]	30
	Indirect function definition from an undefined function [E507]	31
	Setting a function before definition [W508]	32
	Unexpandable or restricted-expandable boolean expression [E509]	33
	Expanding an unexpandable function [E510]	34
	Fully-expanding a restricted-expandable function [E511]	34
	Defined an expandable function as protected [W512]	35
	Defined an unexpandable function as unprotected [W513]	35
	Conditional function with no return value [E514]	36
	Conditional function with no return value [E515]	36
	Comparison code with no return value [E516]	36
	Paragraph token in the parameter of a "nopar" function [E517]	37
5.2	Variables and constants	37
	Unused variable or constant [W518]	37
	Setting an undeclared variable [E519]	38
	Using an undeclared variable or constant [W520]	38

	Multiply declared variable or constant [E521]	39
5.3	Messages	39
	Unused message [W522]	39
	Using an undefined message [E523]	40
	Incorrect number of arguments supplied to message [W524]	40
	Multiply defined message [E525]	41
5.4	Input-output streams	41
	Using an unopened or closed stream [E526]	41
	Multiply opened stream [E527]	41
	Unclosed stream [W528]	42
5.5	Piecewise token list construction	42
	Building on a regular token list [T529]	42
	Using a semi-built token list [T530]	43
	Multiply started building a token list [E531]	43
	Unfinished semi-built token list [W532]	44
	Caveats	44
	References	46
	Index	48

Introduction

In this document, I list the warnings and errors for the different processing steps of the `expl3` linter [1]:

Preprocessing Determine which parts of the input files contain `expl3` code.

Lexical analysis Convert `expl3` parts of the input files into `TEX` tokens.

Syntactic analysis Convert `TEX` tokens into a tree of function calls.

Semantic analysis Determine the meaning of the different function calls.

Flow analysis Determine additional emergent properties of the code.

For each warning and error, I specify a unique identifier that can be used to disable the warning or error, a description of the condition for the warning or error, and a code example that demonstrates the condition and serves as a test case for the linter.

Warnings and errors have different types that decides the prefix of their identifiers:

- Warnings:
 - S : Style warnings
 - W : Other warnings
- Errors:
 - T : Type errors
 - E : Other errors

Issues that are planned but not yet implemented are grayed out.

1 Preprocessing

In the preprocessing step, the `expl3` analysis tool determines which parts of the input files contain `expl3` code. Inline `TEX` comments that disable warnings and errors are also analyzed in this step.

No standard delimiters [W100]

An input file contains no delimiters such as `\ExplSyntaxOn`, `\ExplSyntaxOff`, `\ProvidesExplPackage`, `\ProvidesExplClass`, and `\ProvidesExplFile` [2, Section 2.1]. The analysis tool should assume that the whole input file is in `expl3`.

```

1  % file-wide warning
2  \tl_new:N
3    \g_example_tl
4  \tl_gset:Nn
5    \g_example_tl
6    { Hello,~ }
7  \tl_gput_right:Nn
8    \g_example_tl
9    { world! }
10 \tl_use:N
11  \g_example_tl

```

Unexpected delimiters [W101]

An input file contains extraneous `\ExplSyntaxOn` delimiters [2, Section 2.1] in `expl3` parts or extraneous `\ExplSyntaxOff` delimiters in non-`expl3` parts.

```

1  \input expl3-generic
2  \ExplSyntaxOff % warning on this line
3  \ExplSyntaxOn
4  \tl_new:N
5    \g_example_tl
6  \tl_gset:Nn
7    \g_example_tl
8    { Hello,~ }
9  \ExplSyntaxOn % warning on this line
10 \tl_gput_right:Nn
11  \g_example_tl
12  { world! }
13 \tl_use:N
14  \g_example_tl

```

Expl3 material in non-`expl3` parts [E102]

An input file contains what looks like `expl3` material [2, Section 1.1] in non-`expl3` parts.

```

1  \ProvidesExplFile{example.tex}{2024-04-09}{1.0.0}{An example
   → file}
2  \tl_new:N
3    \g_example_tl
4  \tl_gset:Nn
5    \g_example_tl
6    { Hello,~ }
7  \tl_gput_right:Nn
8    \g_example_tl
9    { world! }

```

```

10 \ExplSyntaxOff
11 \tl_use:N % error on this line
12 \g_example_tl % error on this line

```

Line too long [S103]

Some lines in expl3 parts are longer than 80 characters [3, Section 2].

```

1 This line is not very long, because it is 80 characters long, not
  → 81 characters.
2 This line is overly long, because it is 81 characters long
  → excluding the comment. % warning on this line
3 This line is not very long, because it is 80 characters long,
  → comments excluded. % no warning on this line

```

The maximum line length can be configured using the command-line option `--max-line-length` or with the Lua option `max_line_length`.

Multiple delimiters \ProvidesExpl* in a single file [E104]

An input file contains multiple delimiters `\ProvidesExplPackage`, `\ProvidesExplClass`, and `\ProvidesExplFile`.

```

1 \ProvidesExplPackage
2   {example.sty}{2024-04-09}{1.0.0}{An example package}
3 \ExplSyntaxOff
4 \ProvidesExplClass % error on this line
5   {example.cls}{2024-04-09}{1.0.0}{An example class}

```

Needlessly ignored issue [S105]

An input file contains `% noqa` comments with needlessly ignored issues.

```

1 % warning on the following line
2 There is no issue here. % noqa: e123

```

2 Lexical analysis

In the lexical analysis step, the expl3 analysis tool converts the expl3 parts of the input files into a list of \TeX tokens.

“Do not use” argument specifiers [W200]

Some control sequence tokens correspond to functions with D (do not use) argument specifiers.

```

1 \tex_space:D % warning on this line
2 \tex_italiccor^^3aD % warning on this line
3 \tex_hyphen^^zD % warning on this line
4 \tex_let:^^44 % warning on this line

```

The above example has been taken from The L^AT_EX Project [2, Chapter 24].

Unknown argument specifiers [E201]

Some control sequence tokens correspond to functions with unknown argument specifiers. [2, Section 1.1]

```

1 \cs_new:Nn
2   \example:bar % error on this line
3   { foo }
4   { bar }
5   { baz }

```

Deprecated control sequences [W202]

Some control sequence tokens correspond to deprecated expl3 control sequences from l3obsolete.txt [5].

```

1 \str_lower_case:n % warning on this line
2   { FOO BAR }

```

Missing stylistic whitespaces [S204]

Some control sequences and curly braces are not surrounded by whitespaces [4, Section 6, 3, Section 3].

```

1 \cs_new:Npn \foo_bar:Nn #1#2
2 {
3   \cs_if_exist:NTF#1 % warning on this line
4     { \__foo_bar:n {#2} }
5     { \__foo_bar:nn{#2}{literal} } % warning on this line
6 }

```

Too many closing braces [E208]

An expl3 part of the input file contains too many closing braces.

```

1 \tl_new:N
2   \g_example_tl
3 \tl_gset:Nn
4   \g_example_tl
5   { Hello,~ } } % error on this line

```

Invalid characters [E209]

An expl3 part of the input file contains invalid characters.

```
1 ^^7f % error on this line
2 \fo^^?o % error on this line
```

3 Syntactic analysis

In the syntactic analysis step, the expl3 analysis tool converts the list of T_EX tokens into a tree of function calls.

Unexpected function call argument [E300]

A function is called with an unexpected argument.

```
1 \cs_new:Nn
2   { unexpected } % error on this line
3   \l_tmpa_tl
```

Partial applications are detected by analysing closing braces (}) and do not produce an error:

```
1 \cs_new:Nn
2   \example_foo:n
3   { foo~#1 }
4 \cs_new:Nn
5   \example_bar:
6   { \example_foo:n }
7 \cs_new:Nn
8   \example_baz:
9   {
10    \example_bar:
11    { bar }
12 }
```

End of expl3 part within function call [E301]

A function call is cut off by the end of a file or an expl3 part of a file:

```
1 \cs_new:Nn % error on this line
2   \example_foo:n
```

Unbraced n-type function call argument [W302]

An n-type function call argument is unbraced:

```

1 \tl_set:No
2   \l_tmpa_tl
3   \l_tmpb_tl % warning on this line

```

Depending on the specific function, this may or may not be an error.

Braced N-type function call argument [W303]

An N-type function call argument is braced:

```

1 \cs_new:Nn
2   { \example_foo:n } % warning on this line
3   { bar }

```

Depending on the specific function, this may or may not be an error.

Unexpected parameter number [E304]

A parameter or replacement text contains parameter tokens (#) followed by unexpected numbers:

```

1 \cs_new:Npn
2   \example_foo:nnn
3   #1#2#9 % error on this line
4   { foo~#1 }

1 \cs_new:Npn
2   \example_foo:nnn
3   #1#2#3
4   { foo~#4 } % error on this line

```

Expanding an unexpandable variable or constant [T305]

A function with a V-type argument is called with a variable or constant that does not support V-type expansion [2, Section 1.1].

```

1 \cs_new:Nn
2   \module_foo:n
3   { #1 }
4 \cs_generate_variant:Nn
5   \module_foo:n
6   { V, v }
7 \module_foo:V
8   \c_false_bool % error on this line
9 \module_foo:v
10  { c_false_bool } % error on this line

```

4 Semantic analysis

In the semantic analysis step, the expl3 analysis tool determines the meaning of the different function calls.

4.1 Functions and conditional functions

Unused private function [W401]

A private function or conditional function name is defined but unused.

```
1 \cs_new:Nn % warning on this line
2   \__module_foo:
3   { bar }

1 \prg_new_conditional:Nnn % warning on this line
2   \__module_foo:
3   { p, T, F, TF }
4   { \prg_return_true: }
```

Unused private function variant [W402]

A private function or conditional function variant name is defined but unused.

```
1 \cs_new:Nn
2   \__module_foo:n
3   { bar~#1 }
4 \cs_generate_variant:Nn % warning on this line
5   \__module_foo:n
6   { V }
7 \__module_foo:n
8   { baz }

1 \prg_new_conditional:Nnn
2   \__module_foo:n
3   { p, T, F, TF }
4   { \prg_return_true: }
5 \prg_generate_conditional_variant:Nnn % warning on this line
6   \__module_foo:n
7   { V }
8   { TF }
9 \__module_foo:nTF
10  { foo }
11  { bar }
12  { baz }
```

Function variant of incompatible type [T403]

A function or conditional function variant is generated from an incompatible argument type [2, Section 5.2, documentation of function `\cs_generate_variant:Nn`].

```
1 \cs_new:Nn
2   \module_foo:Nn
3   { bar }
4 \cs_generate_variant:Nn
5   \module_foo:Nn
6   { Nnn } % error on this line
```

Higher-order variants can be created from existing variants as long as only `n` and `N` arguments are changed to other types:

```
1 \cs_new:Nn
2   \module_foo:Nn
3   { bar }
4 \cs_generate_variant:Nn
5   \module_foo:Nn
6   { cn }
7 \cs_generate_variant:Nn
8   \module_foo:cn
9   { cx }
10 \cs_generate_variant:Nn
11  \module_foo:cx
12  { Ne } % error on this line
```

Protected predicate function [E404]

A protected predicate function is defined.

```
1 \prg_new_protected_conditional:Nnn
2   \module_foo:
3   { p }
4   { \prg_return_true: }
```

Function variant for an undefined function [E405]

A function or conditional function variant is defined for an undefined function.

```
1 \cs_new:Nn
2   \module_foo:n
3   { bar }
4 \cs_generate_variant:Nn
5   \module_bar:n % error on this line
6   { V }
```

```

1 \prg_new_conditional:Nnn
2   \module_foo:n
3     { p, T, F }
4     { \prg_return_true: }
5 \prg_generate_conditional_variant:Nnn
6   \module_bar:n % error on this line
7     { V }
8     { T }
9 \prg_generate_conditional_variant:Nnn
10  \module_foo:n % error on this line
11    { V }
12    { TF }

```

Calling an undefined function [E408]

A function or conditional function (variant) is called but undefined.

```

1 \module_foo: % error on this line

1 \cs_new:Nn
2   \module_foo:n
3     { bar~#1 }
4 \tl_set:Nn
5   \l_tmpa_tl
6     { baz }
7 \module_foo:V % error on this line
8   \l_tmpa_tl

1 \prg_new_conditional:Nnn
2   \module_foo:n
3     { p, T, F, TF }
4     { \prg_return_true: }
5 \prg_generate_conditional_variant:Nnn
6   \module_foo:n
7     { V }
8     { T }
9 \module_foo:VTF % error on this line
10  \l_tmpa_tl
11    { foo }
12    { bar }

```

Function variant of deprecated type [W410]

A function or conditional function variant is generated from a deprecated argument type [2, Section 5.2, documentation of function `\cs_generate_variant:Nn`].

```

1 \cs_new:Nn

```

```

2   \module_foo:Nn
3   { bar }
4   \cs_generate_variant:Nn
5   \module_foo:Nn
6   { nn } % warning on this line
7   \cs_generate_variant:Nn
8   \module_foo:Nn
9   { NN } % warning on this line
10  \cs_generate_variant:Nn
11  \module_foo:Nn
12  { vc } % warning on this line

```

Indirect function definition from an undefined function [E411]

A function or conditional function is indirectly defined from an undefined function.

```

1   \cs_new:Nn
2   \module_foo:n
3   { bar~#1 }
4   \cs_new_eq:NN
5   \module_bar:n
6   \module_foo:n
7   \cs_new_eq:NN
8   \module_baz:n
9   \module_bar:n
10  \module_baz:n
11  { foo }

1   \prg_new_conditional:Nnn
2   \module_foo:n
3   { p, T, F, TF }
4   { \prg_return_true: }
5   \cs_new_eq:NN
6   \module_bar:nTF
7   \module_foo:nTF
8   \cs_new_eq:NN
9   \module_baz:nTF
10  \module_bar:nTF
11  \module_baz:nTF
12  { foo }
13  { bar }
14  { baz }

1   \cs_new:Nn
2   \module_foo:n
3   { bar~#1 }
4   \cs_new_eq:NN % error on this line

```

```

5   \module_baz:n
6   \module_bar:n
7   \module_baz:n
8   { foo }

1  \prg_new_conditional:Nnn
2  \module_foo:n
3  { p, T, F, TF }
4  { \prg_return_true: }
5  \cs_new_eq:NN % error on this line
6  \module_baz:nTF
7  \module_bar:nTF
8  \module_baz:nTF
9  { foo }
10 { bar }
11 { baz }

```

Malformed function name [S412]

Some function have names that are not in the format $\backslash\langle module \rangle_ \langle description \rangle : \langle arg-spec \rangle$ [4, Section 3.2].

```

1  \cs_new:Nn
2  \description: % warning on this line
3  { foo }

1  \cs_gset:Npn
2  \module__description: % warning on this line
3  { foo }

1  \cs_new:Nn
2  \example_foo:
3  { bar }
4  \cs_set_eq:NN
5  \module_description: % warning on this line
6  \example_foo:

1  \cs_generate_from_arg_count:NNnn
2  \__module_description:
3  \cs_new:Npn
4  { 0 }
5  { foo }

```

This also extends to conditional functions:

```

1  \prg_new_conditional:Nn
2  \description: % warning on this line
3  { p, T, F, TF }
4  { foo }

```

```

1 \prg_gset_conditional:Npn
2   \module_description: % warning on this line
3   { p, T, F, TF }
4   { foo }

```

```

1 \prg_new_conditional:Nnn
2   \example_foo:
3   { p, T, F, TF }
4   { \prg_return_true: }
5 \prg_set_eq_conditional:NNn
6   \module_description: % warning on this line
7   \example_foo:
8   { p, T, F, TF }

```

Furthermore, this also extends to function variants:

```

1 \cs_new:Nn
2   \example:n % warning on this line
3   { #1 }
4 \cs_generate_variant:Nn
5   \example:n % warning on this line
6   { e, x, V }

```

4.2 Variables and constants

Malformed variable or constant name [S413]

Some expl3 variables and constants have names that are not in the format $\langle scope \rangle_ \langle module \rangle_ \langle description \rangle_ \langle type \rangle$ [4, Section 3.2].

```

1 \tl_new:N
2   \g_description_tl % warning on this line
3 \box_use:N
4   \l_description_box % warning on this line
5 \int_const:Nn
6   \c_description % warning on this line
7   { 123 }

```

```

1 \regex_new:N
2   \g_module_description_regex
3 \coffin_new:N
4   \l_module_description_coffin
5 \str_const:Nn
6   \c_module_description_str
7   { foo }

```

```

1 \tl_use:N
2   \l_tmpa_tl

```

```

3 \int_gset:Nn
4   \g_tmpb_int
5   { 1 + 2 }
6 \str_show:N
7   \g_tmpa_str
8 \bool_set_true:N
9   \l_tmpa_bool

```

Malformed quark or scan mark name [S414]

Some expl3 quarks and scan marks have names that do not start with `\q_` and `\s_`, respectively [4, Chapter 19].

```

1 \quark_new:N
2   \foo_bar % warning on this line

```

```

1 \quark_new:N
2   \q_foo_bar

```

```

1 \scan_new:N
2   \foo_bar % warning on this line

```

```

1 \scan_new:N
2   \s_foo_bar

```

Unused variable or constant [W415]

A variable or a constant name is declared and perhaps defined but unused.

```

1 \tl_new:N
2   \g_declared_but_undefined_tl % warning on this line

```

```

1 \tl_new:N
2   \g_defined_but_unused_tl % warning on this line
3 \tl_gset:Nn
4   \g_defined_but_unused_tl
5   { foo }

```

```

1 \tl_new:N
2   \g_defined_and_used_tl
3 \tl_gset:Nn
4   \g_defined_and_used_tl
5   { foo }
6 \tl_use:N
7   \g_defined_and_used_tl

```

```

1 \tl_const:Nn
2   \c_defined_but_unused_tl % warning on this line
3   { foo }

```

```

1 \tl_const:Nn
2   \c_defined_and_used_tl
3   { foo }
4 \tl_use:N
5   \c_defined_and_used_tl

```

Setting an undeclared variable [W416]

An undeclared variable is set.

```

1 \tl_gset:Nn
2   \g_example_tl % warning on this line
3   { bar }

```

Setting a variable as a constant [E417]

A variable is set as though it were a constant.

```

1 \tl_const:Nn
2   \g_example_tl % error on this line
3   { bar }

```

Setting a constant [E418]

A constant is set.

```

1 \tl_gset:Nn
2   \c_example_tl % error on this line
3   { bar }

```

Using an undeclared variable or constant [W419]

A variable or constant is used but undeclared or undefined.

```

1 \tl_use:N
2   \g_undeclared_tl % warning on this line

1 \tl_new:N
2   \g_declared_but_undefined_tl
3 \tl_use:N
4   \g_declared_but_undefined_tl

1 \tl_new:N
2   \g_defined_tl
3 \tl_gset:Nn
4   \g_defined_tl
5   { foo }
6 \tl_use:N
7   \g_defined_tl

```

```

1 \tl_use:N
2   \c_undefined_tl % warning on this line

1 \tl_const:Nn
2   \c_defined_tl
3   { foo }
4 \tl_use:N
5   \c_defined_tl

```

This also includes usage in V- and v-type arguments:

```

1 \cs_new:Nn
2   \example_foo:n
3   { foo }
4 \cs_generate_variant:Nn
5   \example_foo:n
6   { V, v }
7 \example_foo:V
8   \g_undeclared_tl % warning on this line
9 \example_foo:v
10  { c_undefined_tl } % warning on this line
11 \tl_new:N
12   \g_defined_tl
13 \tl_gset:Nn
14   \g_defined_tl
15   { bar }
16 \tl_const:Nn
17   \c_defined_tl
18   { baz }
19 \example_foo:V
20   \g_defined_tl
21 \example_foo:v
22   { c_defined_tl }

```

Locally setting a global variable [E420]

A global variable is locally set.

```

1 \tl_new:N
2   \g_example_tl
3 \tl_set:Nn % error on this line
4   \g_example_tl
5   { foo }

```

Globally setting a local variable [E421]

A local variable is globally set.

```

1 \tl_new:N
2   \l_example_tl
3 \tl_gset:Nn % error on this line
4   \l_example_tl
5   { foo }

```

Using a variable of an incompatible type [T422]

A variable of one type is used where a variable of a different type should be used.

```

1 \tl_new:N % error on this line
2   \l_example_str
3 \str_new:N % error on this line
4   \l_example_tl

1 \tl_new:N
2   \l_example_tl
3 \tl_count:N
4   \l_example_tl
5 \str_count:N
6   \l_example_tl
7 \seq_count:N % error on this line
8   \l_example_tl
9 \clist_count:N
10  \l_example_tl
11 \prop_count:N % error on this line
12  \l_example_tl
13 \intarray_count:N % error on this line
14  \l_example_tl
15 \fparray_count:N % error on this line
16  \l_example_tl

1 \str_new:N
2   \l_example_str
3 \tl_count:N
4   \l_example_str
5 \str_count:N
6   \l_example_str
7 \seq_count:N % error on this line
8   \l_example_str
9 \clist_count:N % error on this line
10  \l_example_str
11 \prop_count:N % error on this line
12  \l_example_str
13 \intarray_count:N % error on this line
14  \l_example_str
15 \fparray_count:N % error on this line
16  \l_example_str

```

```

1 \int_new:N
2   \l_example_int
3 \tl_count:N % error on this line
4   \l_example_int
5 \str_count:N % error on this line
6   \l_example_int
7 \seq_count:N % error on this line
8   \l_example_int
9 \clist_count:N % error on this line
10  \l_example_int
11 \prop_count:N % error on this line
12  \l_example_int
13 \intarray_count:N % error on this line
14  \l_example_int
15 \fparray_count:N % error on this line
16  \l_example_int

1 \seq_new:N
2   \l_example_seq
3 \tl_count:N % error on this line
4   \l_example_seq
5 \str_count:N % error on this line
6   \l_example_seq
7 \seq_count:N
8   \l_example_seq
9 \clist_count:N % error on this line
10  \l_example_seq
11 \prop_count:N % error on this line
12  \l_example_seq
13 \intarray_count:N % error on this line
14  \l_example_seq
15 \fparray_count:N % error on this line
16  \l_example_seq

1 \clist_new:N
2   \l_example_clist
3 \tl_count:N
4   \l_example_clist
5 \str_count:N % error on this line
6   \l_example_clist
7 \seq_count:N % error on this line
8   \l_example_clist
9 \clist_count:N
10  \l_example_clist
11 \prop_count:N % error on this line
12  \l_example_clist
13 \intarray_count:N % error on this line

```

```

14   \l_example_clist
15   \fparray_count:N % error on this line
16   \l_example_clist

1   \clist_new:N % error on this line
2   \l_example_prop
3   \tl_count:N % error on this line
4   \l_example_prop
5   \str_count:N % error on this line
6   \l_example_prop
7   \seq_count:N % error on this line
8   \l_example_prop
9   \clist_count:N % error on this line
10  \l_example_prop
11  \prop_count:N
12  \l_example_prop
13  \intarray_count:N % error on this line
14  \l_example_prop
15  \fparray_count:N % error on this line
16  \l_example_prop

1   \intarray_new:Nn
2   \g_example_intarray
3   { 5 }
4   \tl_count:N % error on this line
5   \g_example_intarray
6   \str_count:N % error on this line
7   \g_example_intarray
8   \seq_count:N % error on this line
9   \g_example_intarray
10  \clist_count:N % error on this line
11  \g_example_intarray
12  \prop_count:N % error on this line
13  \g_example_intarray
14  \intarray_count:N
15  \g_example_intarray
16  \fparray_count:N % error on this line
17  \g_example_intarray

1   \fparray_new:Nn
2   \g_example_fparray
3   { 5 }
4   \tl_count:N % error on this line
5   \g_example_fparray
6   \str_count:N % error on this line
7   \g_example_fparray
8   \seq_count:N % error on this line

```

```

9   \g_example_farray
10  \clist_count:N % error on this line
11  \g_example_farray
12  \prop_count:N % error on this line
13  \g_example_farray
14  \intarray_count:N % error on this line
15  \g_example_farray
16  \farray_count:N
17  \g_example_farray

1   \ior_new:N
2   \l_example_ior
3   \iow_open:Nn % error on this line
4   \l_example_ior
5   { example }

1   \clist_new:N
2   \l_example_clist
3   \tl_set:Nn
4   \l_tmpa_tl
5   { foo }
6   \clist_set_eq:NN % error on this line
7   \l_example_clist
8   \l_tmpa_tl

1   \tl_set:Nn
2   \l_tmpa_tl
3   { foo }
4   \seq_set_from_clist:NN % error on this line
5   \l_tmpa_seq
6   \l_tmpa_tl

1   \tl_set:Nn
2   \l_tmpa_tl
3   { foo }
4   \regex_set:Nn
5   \l_tmpa_regex
6   { foo }
7   \int_set:Nn
8   \l_tmpa_int
9   { 1 + 2 }
10  \regex_show:N % error on this line
11  \l_tmpa_tl
12  \regex_show:N
13  \l_tmpa_regex
14  \regex_show:N % error on this line
15  \l_tmpa_int

```

```

1 \tl_set:Nn
2   \l_tmpa_tl
3   { foo }
4 \int_set_eq:NN % error on this line
5   \l_tmpa_int
6   \l_tmpa_tl

1 \str_new:N
2   \l_example_str
3 \tl_const:Nn
4   \c_example_tl
5   { foo }
6 \str_set_eq:NN % error on this line
7   \l_example_str
8   \c_example_tl

1 \tl_new:N
2   \l_example_tl
3 \str_new:N
4   \l_example_str
5 \str_set:Nn
6   \l_example_str
7   { foo }
8 \tl_set_eq:NN
9   \l_example_tl
10  \l_example_str
11 \str_set_eq:NN
12  \l_example_tl
13  \l_example_str
14 \tl_set_eq:NN % error on this line
15  \l_example_str
16  \l_example_tl
17 \str_set_eq:NN % error on this line
18  \l_example_str
19  \l_example_tl

1 \str_set_eq:NN % error on this line
2   \l_example_tl
3   \l_example_tl
4 \tl_set_eq:NN % error on this line
5   \l_example_str
6   \l_example_str

1 \seq_set_from_clist:NN
2   \l_tmpa_seq
3   \l_tmpa_clist
4 \seq_set_from_clist:NN % error on this line
5   \l_tmpa_seq

```

```

6   \l_tmpa_tl
7   \seq_set_from_clist:NN % error on this line
8   \l_tmpa_seq
9   \l_tmpa_seq
10  \clist_set_from_seq:NN
11  \l_tmpa_clist
12  \l_tmpa_seq
13  \clist_set_from_seq:NN % error on this line
14  \l_tmpa_clist
15  \l_tmpa_int

```

4.3 Messages

Unused message [W423]

A message name is defined but unused.

```

1   \msg_new:nnn % warning on this line
2   { foo }
3   { bar }
4   { baz }

1   \msg_new:nnn
2   { bar }
3   { bar }
4   { baz }
5   \msg_info:nn
6   { bar }
7   { bar }

```

Using an undefined message [E424]

A message is used but undefined.

```

1   \msg_info:nnn % error on this line
2   { foo }
3   { bar }
4   { baz }

```

Incorrect parameters in message text [E425]

Parameter tokens other than #1, #2, #3, and #4 are specified in a message text.

```

1   \msg_new:nnn
2   { foo }
3   { bar }
4   { #5 } % error on this line

```

```

1 \msg_new:nnnn
2   { foo }
3   { bar }
4   { #5~#6 } % two errors on this line
5   { #6~#7 } % two errors on this line

1 \msg_new:nnnn
2   { foo }
3   { bar }
4   { #1~#2 }
5   { #3~#4 }

```

Incorrect number of arguments supplied to message [W426]

A message was supplied fewer or more arguments than there are parameters in the message text.

```

1 \msg_new:nnn
2   { foo }
3   { bar }
4   { #1~#2 }
5 \msg_info:nn % warning on this line
6   { foo }
7   { bar }
8 \msg_info:nnn % warning on this line
9   { foo }
10  { bar }
11  { foo }
12 \msg_info:nnnn
13  { foo }
14  { bar }
15  { foo }
16  { bar }
17 \msg_info:nnnnn % warning on this line
18  { foo }
19  { bar }
20  { foo }
21  { bar }
22  { baz }

```

4.4 Sorting

Comparison conditional without signature :nnTF [E427]

A sorting function is called with a conditional that has a signature different than :nnTF [2, Section 15.5.4].

```

1 \cs_new:Nn

```

```

2   \example_foo:
3   { \prg_return_true: }
4   \tl_sort:nN
5   { { foo } { bar } }
6   \example_foo:nnT % error on this line

1   \cs_new:Nn
2   \example_foo:
3   { \prg_return_true: }
4   \tl_sort:nN
5   { { foo } { bar } }
6   \example_foo:nnTF

```

5 Flow analysis

In the flow analysis step, the `expl3` analysis tool determines compiler-theoretic properties of functions, such as expandability, and variables, such as reaching definitions.

5.1 Functions and conditional functions

Multiply defined function [E500]

A function or conditional function is defined multiple times.

```

1   \cs_new:Nn
2   \module_foo:
3   { bar }
4   \cs_new:Nn % error on this line
5   \module_foo:
6   { bar }

1   \cs_new:Nn
2   \module_foo:
3   { bar }
4   \cs_undefine:N
5   \module_foo:
6   \cs_new:Nn
7   \module_foo:
8   { bar }

1   \cs_new:Nn
2   \module_foo:
3   { bar }
4   \cs_gset:Nn
5   \module_foo:
6   { bar }

```

```

1 \prg_new_conditional:Nnn
2   \module_foo:
3     { p, T, F, TF }
4     { \prg_return_true: }
5 \prg_new_conditional:Nnn % error on this line
6   \module_foo:
7     { p, T, F, TF }
8     { \prg_return_true: }

1 \prg_new_conditional:Nnn
2   \module_foo:
3     { p, T, F, TF }
4     { \prg_return_true: }
5 \cs_undefine:N
6   \module_foo_p:
7 \cs_undefine:N
8   \module_foo:T
9 \cs_undefine:N
10  \module_foo:F
11 \cs_undefine:N
12  \module_foo:TF
13 \prg_new_conditional:Nnn
14   \module_foo:
15     { p, T, F, TF }
16     { \prg_return_true: }

1 \prg_new_conditional:Nnn
2   \module_foo:
3     { p, T, F, TF }
4     { \prg_return_true: }
5 \prg_gset_conditional:Nnn
6   \module_foo:
7     { p, T, F, TF }
8     { \prg_return_true: }

```

Multiply defined function variant [W501]

A function or conditional function variant is defined multiple times.

```

1 \cs_new:Nn
2   \module_foo:n
3     { bar~#1 }
4 \cs_generate_variant:Nn
5   \module_foo:n
6     { V }
7 \cs_generate_variant:Nn % warning on this line
8   \module_foo:n
9     { o, V }

```

```

1 \cs_new:Nn
2   \module_foo:n
3   { bar~#1 }
4 \cs_generate_variant:Nn
5   \module_foo:n
6   { V }
7 \cs_undefine:N
8   \module_foo:V
9 \cs_generate_variant:Nn
10  \module_foo:n
11  { o, V }

1 \prg_new_conditional:Nnn
2   \module_foo:n
3   { p, T, F, TF }
4   { \prg_return_true: }
5 \prg_generate_conditional_variant:Nnn
6   \module_foo:n
7   { V }
8   { TF }
9 \prg_generate_conditional_variant:Nnn % warning on this line
10  \module_foo:n
11  { o, V }
12  { TF }

1 \prg_new_conditional:Nnn
2   \module_foo:n
3   { p, T, F, TF }
4   { \prg_return_true: }
5 \prg_generate_conditional_variant:Nnn
6   \module_foo:n
7   { V }
8   { TF }
9 \cs_undefine:N
10  \module_foo:VTF
11 \prg_generate_conditional_variant:Nnn
12  \module_foo:n
13  { o, V }
14  { TF }

```

Unused private function [W502]

A private function or conditional function is defined but unused.

```

1 \cs_new:Nn % warning on this line
2   \__module_foo:
3   { bar }

```

```

4 \cs_new:Nn
5   \__module_baz:
6   { \__module_foo: }

```

This check is a stronger version of W401 and should only be emitted if W401 has not previously been emitted for this function.

Unused private function variant [W503]

A private function or conditional function variant is defined but unused.

```

1 \cs_new:Nn
2   \__module_foo:n
3   { bar~#1 }
4 \cs_new:Nn
5   \__module_baz:
6   {
7     \tl_set:Nn
8       \l_tmpa_tl
9       { baz }
10    \__module_foo:V
11    \l_tmpa_tl
12  }
13 \cs_generate_variant:Nn % warning on this line
14   \__module_foo:n
15   { V }
16 \__module_foo:n
17   { baz }

```

This check is a stronger version of W402 and should only be emitted if W402 has not previously been emitted for this function variant.

Function variant for an undefined function [E505]

A function or conditional function variant is defined before the base function has been defined or after it has been undefined.

```

1 \cs_new:Nn
2   \module_foo:n
3   { bar }
4 \cs_generate_variant:Nn
5   \module_foo:n
6   { V }
7
8 \cs_generate_variant:Nn % error on this line
9   \module_foo:n
10  { V }
11 \cs_new:Nn

```

```

5   \module_foo:n
6   { bar }

1  \cs_new:Nn
2   \module_foo:n
3   { bar }
4  \cs_undefine:N
5   \module_foo:n
6  \cs_generate_variant:Nn % error on this line
7   \module_foo:n
8   { V }

1  \prg_new_conditional:Nnn
2   \module_foo:n
3   { p, T, F, TF }
4   { \prg_return_true: }
5  \prg_generate_conditional_variant:Nnn
6   \module_foo:n
7   { V }
8   { TF }

1  \prg_generate_conditional_variant:Nnn % error on this line
2   \module_foo:n
3   { V }
4   { TF }
5  \prg_new_conditional:Nnn
6   \module_foo:n
7   { p, T, F, TF }
8   { \prg_return_true: }

1  \prg_new_conditional:Nnn
2   \module_foo:n
3   { p, T, F, TF }
4   { \prg_return_true: }
5  \cs_undefine:N
6   \module_foo:nTF
7  \prg_generate_conditional_variant:Nnn % error on this line
8   \module_foo:n
9   { V }
10  { TF }

```

This check is a stronger version of E405 and should only be emitted if E405 has not previously been emitted for this function variant.

Calling an undefined function [E506]

A function or conditional function (variant) is called before it has been defined or after it has been undefined.

```

1 \module_foo: % error on this line
2 \cs_new:Nn
3   \module_foo:
4   { bar }

1 \cs_new:Nn
2   \module_foo:
3   { bar }
4 \cs_undefine:N
5   \module_foo:
6 \module_foo: % error on this line

1 \cs_new:Nn
2   \module_foo:n
3   { bar }
4 \tl_set:Nn
5   \l_tmpa_tl
6   { baz }
7 \module_foo:V % error on this line
8   \l_tmpa_tl
9 \cs_generate_variant:Nn
10  \module_foo:n
11  { V }

```

This check is a stronger version of E408 and should only be emitted if E408 has not previously been emitted for this function.

Indirect function definition from an undefined function [E507]

A function or conditional function is indirectly defined from a function that has yet to be defined or after it has been undefined.

```

1 \cs_new:Nn
2   \module_foo:n
3   { bar~#1 }
4 \cs_new_eq:NN % error on this line
5   \module_baz:n
6   \module_bar:n
7 \cs_new_eq:NN
8   \module_bar:n
9   \module_foo:n
10 \module_baz:n
11  { foo }

1 \cs_new:Nn
2   \module_foo:n
3   { bar~#1 }
4 \cs_new_eq:NN

```

```

5   \module_bar:n
6   \module_foo:n
7   \cs_undefine:N
8   \module_bar:n
9   \cs_new_eq:NN % error on this line
10  \module_baz:n
11  \module_bar:n
12  \module_baz:n
13  { foo }

1   \prg_new_conditional:Nnn
2   \module_foo:n
3   { p, T, F, TF }
4   { \prg_return_true: }
5   \cs_new_eq:NN % error on this line
6   \module_baz:nTF
7   \module_bar:nTF
8   \cs_new_eq:NN
9   \module_bar:nTF
10  \module_foo:nTF
11  \module_baz:nTF
12  { foo }
13  { bar }
14  { baz }

1   \prg_new_conditional:Nnn
2   \module_foo:n
3   { p, T, F, TF }
4   { \prg_return_true: }
5   \cs_new_eq:NN
6   \module_bar:nTF
7   \module_foo:nTF
8   \cs_undefine:N
9   \module_bar:nTF
10  \cs_new_eq:NN % error on this line
11  \module_baz:nTF
12  \module_bar:nTF
13  \module_baz:nTF
14  { foo }
15  { bar }
16  { baz }

```

This check is a stronger version of E411 and should only be emitted if E411 has not previously been emitted for this function.

Setting a function before definition [W508]

A function is set before it has been defined or after it has been undefined.

```

1 \cs_gset:N % warning on this line
2   \module_foo:
3   { foo }
4 \cs_new:Nn
5   \module_foo:
6   { bar }

1 \cs_new:Nn
2   \module_foo:
3   { bar }
4 \cs_undefine:N
5   \module_foo:
6 \cs_gset:N % warning on this line
7   \module_foo:
8   { foo }

```

Unexpandable or restricted-expandable boolean expression [E509]

A boolean expression [2, Section 9.2] is not fully-expandable.

```

1 \cs_new_protected:N
2   \example_unexpandable:
3   {
4     \tl_set:Nn
5       \l_tmpa_tl
6       { bar }
7     \c_true_bool
8   }
9 \cs_new:N
10  \example_restricted_expandable:
11  {
12    \bool_do_while:Nn
13      \c_false_bool
14      { }
15    \c_true_bool
16  }
17 \cs_new_protected:N
18  \example_expandable:
19  { \c_true_bool }
20 \bool_set:Nn
21  \l_tmpa_bool
22  { \example_unexpandable: } % error on this line
23 \bool_set:Nn
24  \l_tmpa_bool
25  { \example_restricted_expandable: } % error on this line
26 \bool_set:Nn

```

```

27 \l_tmpa_bool
28 { \example_expandable: }

```

Expanding an unexpandable function [E510]

An unexpandable function or conditional function is called within an x-type, e-type, or f-type argument.

```

1 \cs_new_protected:N
2 \example_unexpandable:
3 {
4 \tl_set:Nn
5 \l_tmpa_tl
6 { bar }
7 }
8 \cs_new:N
9 \module_foo:n
10 { #1 }
11 \cs_generate_variant:Nn
12 \module_foo:n
13 { x, e, f }
14 \module_foo:n
15 { \example_unexpandable: }
16 \module_foo:x
17 { \example_unexpandable: } % error on this line
18 \module_foo:e
19 { \example_unexpandable: } % error on this line
20 \module_foo:f
21 { \example_unexpandable: } % error on this line

```

Fully-expanding a restricted-expandable function [E511]

An restricted-expandable function or conditional function is called within an f-type argument.

```

1 \cs_new:N
2 \example_restricted_expandable:
3 {
4 \int_to_roman:n
5 { 1 + 2 }
6 }
7 \cs_new:N
8 \module_foo:n
9 { #1 }
10 \cs_generate_variant:Nn
11 \module_foo:n

```

```

12 { x, e, f }
13 \module_foo:n
14 { \example_restricted_expandable: }
15 \module_foo:x
16 { \example_restricted_expandable: }
17 \module_foo:e
18 { \example_restricted_expandable: }
19 \module_foo:f
20 { \example_restricted_expandable: } % error on this line

```

Defined an expandable function as protected [W512]

A fully-expandable function or conditional function is defined using a creator function `\cs_new_protected:*` or `\prg_new_protected_conditional:*`. [3, Section 4]

```

1 \cs_new_protected:Nn % warning on this line
2   \example_expandable:
3   { foo }

1 \prg_new_protected_conditional:Nnn % warning on this line
2   \example_expandable:
3   { T, F, TF }
4   { \prg_return_true: }

```

Defined an unexpandable function as unprotected [W513]

An unexpandable or restricted-expandable function or conditional function is defined using a creator function `\cs_new:*` or `\prg_new_conditional:*`. [3, Section 4]

```

1 \cs_new:Nn % warning on this line
2   \example_unexpandable:
3   {
4     \tl_set:Nn
5       \l_tmpa_tl
6       { bar }
7   }

1 \prg_new_conditional:Nnn % warning on this line
2   \example_unexpandable:
3   { p, T, F, TF }
4   {
5     \tl_set:Nn
6       \l_tmpa_tl
7       { bar }
8     \prg_return_true:
9   }

```

Conditional function with no return value [E514]

A conditional functions has no return value.

```
1 \prg_new_conditional:Nnn % error on this line
2   \example_no_return_value:
3   { p, T, F, TF }
4   { foo }
```

```
1 \prg_new_conditional:Nnn
2   \example_has_return_value:
3   { p, T, F, TF }
4   { \example_foo: }
5 \cs_new:Nn
6   \example_foo:
7   { \prg_return_true: }
```

Conditional function with no return value [E515]

A conditional functions has no return value.

```
1 \prg_new_conditional:Nnn % error on this line
2   \example_no_return_value:
3   { p, T, F, TF }
4   { foo }
```

```
1 \prg_new_conditional:Nnn
2   \example_has_return_value:
3   { p, T, F, TF }
4   { \example_foo: }
5 \cs_new:Nn
6   \example_foo:
7   { \prg_return_true: }
```

Comparison code with no return value [E516]

A comparison code [2, Section 6.1] has no return value.

```
1 \clist_set:Nn
2   \l_foo_clist
3   { 3 , 01 , -2 , 5 , +1 }
4 \clist_sort:Nn % error on this line
5   \l_foo_clist
6   { foo }
```

```
1 \clist_set:Nn
2   \l_foo_clist
```

```

3   { 3 , 01 , -2 , 5 , +1 }
4   \clist_sort:Nn
5     \l_foo_clist
6     { \example_foo: }
7   \cs_new:Nn
8     \example_foo:
9     {
10      \int_compare:nNnTF { #1 } > { #2 }
11        { \sort_return_swapped: }
12        { \sort_return_same: }
13    }

```

The above example has been taken from The L^AT_EX Project [2, Chapter 6].

Paragraph token in the parameter of a "nopar" function [E517]

An argument that contains \par tokens may reach a function with the "nopar" restriction.

```

1   \cs_new_nopar:Nn
2     \example_foo:n
3     { #1 }
4   \cs_new:nn
5     \example_bar:n
6     {
7       \example_foo:n
8       { #1 }
9     }
10  \example_bar:n
11  {
12    foo
13    \par % error on this line
14    bar
15  }

```

5.2 Variables and constants

Unused variable or constant [W518]

A variable or a constant is declared and perhaps defined but unused.

```

1   \tl_new:N % warning on this line
2     \g_defined_but_unreachable_tl
3   \tl_gset:Nn
4     \g_defined_but_unreachable_tl
5     { foo }
6   \cs_new:Nn

```

```

7  \__module_baz:
8  {
9    \tl_use:N
10   \g_defined_but_unreachable_tl
11 }

```

This check is a stronger version of W415 and should only be emitted if W415 has not previously been emitted for this variable or constant.

Setting an undeclared variable [E519]

A variable is set before it has been declared.

```

1  \tl_gset:Nn % error on this line
2  \g_example_tl
3  { bar }
4  \tl_new:N
5  \g_example_tl

```

This check is a stronger version of W416 and should prevent W416 from being emitted for this variable.

Using an undeclared variable or constant [W520]

A variable or constant is used before it has been declared.

```

1  \tl_use:N
2  \g_example_tl % warning on this line
3  \tl_new:N
4  \g_example_tl

1  \tl_use:N
2  \c_example_tl % warning on this line
3  \tl_const:N
4  \c_example_tl
5  { foo }

```

This also includes usage in V- and v-type arguments:

```

1  \cs_new:Nn
2  \example_foo:n
3  { foo }
4  \cs_generate_variant:Nn
5  \example_foo:n
6  { V, v }
7  \example_foo:V
8  \g_example_tl % warning on this line
9  \example_foo:v

```

```

10 { c_example_tl } % warning on this line
11 \tl_new:N
12   \g_example_tl
13 \tl_const:N
14   \c_example_tl
15   { foo }

```

This check is a stronger version of W419 and should only be emitted if W419 has not previously been emitted for this variable or constant.

Multiply declared variable or constant [E521]

A variable or constant is declared multiple times.

```

1 \tl_new:N
2   \g_example_tl
3 \tl_new:N % error on this line
4   \g_example_tl

1 \tl_const:Nn
2   \c_example_tl
3   { foo }
4 \tl_const:Nn % error on this line
5   \c_example_tl
6   { bar }

```

5.3 Messages

Unused message [W522]

A message is defined but unused.

```

1 \msg_new:nnn % warning on this line
2   { foo }
3   { bar }
4   { baz }
5 \cs_new:Nn
6   \__module_baz:
7   {
8     \msg_info:nn
9       { foo }
10      { bar }
11  }

```

This check is a stronger version of W423 and should only be emitted if W423 has not previously been emitted for this message.

Using an undefined message [E523]

A message is used before it has been defined.

```
1 \msg_info:nn % error on this line
2 { foo }
3 { bar }
4 \msg_new:nnn
5 { foo }
6 { bar }
7 { baz }
```

This check is a stronger version of E424 and should only be emitted if E424 has not previously been emitted for this message.

Incorrect number of arguments supplied to message [W524]

A message was supplied fewer or more arguments than there are parameters in the message text.

```
1 \msg_new:nnn
2 { foo }
3 { bar }
4 { #1 }
5 \msg_set:nnn
6 { foo }
7 { bar }
8 { baz }
9 \msg_info:nnn % error on this line
10 { foo }
11 { bar }
12 { baz }

1 \msg_new:nnn
2 { foo }
3 { bar }
4 { #1 }
5 \msg_info:nnn
6 { foo }
7 { bar }
8 { baz }
9 \msg_set:nnn
10 { foo }
11 { bar }
12 { baz }
```

This check is a stronger version of W426 and should only be emitted if W426 has not previously been emitted for this message.

Multiply defined message [E525]

A message is defined multiple times.

```
1 \msg_new:nnn
2   { foo }
3   { bar }
4   { baz }
5 \msg_new:nnn % error on this line
6   { foo }
7   { bar }
8   { baz }
```

5.4 Input–output streams

Using an unopened or closed stream [E526]

A stream is used before it has been opened or after it has been closed.

```
1 \ior_new:N
2   \l_example_ior
3 \ior_str_get:NN % error on this line
4   \l_example_ior
5   \l_tmpa_tl
6 \ior_open:Nn
7   \l_example_ior
8   { example }

1 \ior_new:N
2   \l_example_ior
3 \ior_open:Nn
4   \l_example_ior
5   { example }
6 \ior_close:N
7   \l_example_ior
8 \ior_str_get:NN % error on this line
9   \l_example_ior
10  \l_tmpa_tl
```

Multiply opened stream [E527]

A stream is opened a second time without closing the stream first.

```
1 \iow_new:N
2   \l_example_iow
3 \iow_open:Nn
4   \l_example_iow
```

```

5   { foo }
6 \iow_open:Nn % error on this line
7   \l_example_iow
8   { bar }
9 \iow_close:N
10  \l_example_iow

```

Unclosed stream [W528]

A stream is opened but not closed.

```

1 % file-wide warning
2 \ior_new:N
3   \l_example_ior
4 \ior_open:Nn
5   \l_example_ior
6   { example }

```

5.5 Piecewise token list construction

Building on a regular token list [T529]

A token list variable is used with `\tl_build_*` functions before a function `\tl_build_*begin:N` has been called or after a function `\tl_build_*end:N` has been called.

```

1 \tl_new:N
2   \l_example_tl
3 \tl_build_put_right:Nn % error on this line
4   \l_example_tl
5   { foo }
6 \tl_build_begin:N
7   \l_example_tl
8 \tl_build_end:N
9   \l_example_tl

1 \tl_new:N
2   \l_example_tl
3 \tl_build_begin:N
4   \l_example_tl
5 \tl_build_put_right:Nn
6   \l_example_tl
7   { foo }
8 \tl_build_end:N
9   \l_example_tl

```

```

1 \tl_new:N
2   \l_example_tl
3 \tl_build_begin:N
4   \l_example_tl
5 \tl_build_end:N
6   \l_example_tl
7 \tl_build_put_right:Nn % error on this line
8   \l_example_tl
9   { foo }

```

Using a semi-built token list [T530]

A token list variable is used where a regular token list is expected after a function `\tl_build_*begin:N` has been called and before a function `\tl_build_*end:N` has been called.

```

1 \tl_new:N
2   \l_example_tl
3 \tl_use:N
4   \l_example_tl
5 \tl_build_begin:N
6   \l_example_tl
7 \tl_build_end:N
8   \l_example_tl

1 \tl_new:N
2   \l_example_tl
3 \tl_build_begin:N
4   \l_example_tl
5 \tl_use:N
6   \l_example_tl % error on this line
7 \tl_build_end:N
8   \l_example_tl

1 \tl_new:N
2   \l_example_tl
3 \tl_build_begin:N
4   \l_example_tl
5 \tl_build_end:N
6   \l_example_tl
7 \tl_use:N
8   \l_example_tl

```

Multiply started building a token list [E531]

A function `\tl_build_*begin:N` is called on a token list variable a second time without calling a function `\tl_build_*end:N` first.

```

1 \tl_new:N
2   \l_example_tl
3 \tl_build_begin:N
4   \l_example_tl
5 \tl_build_begin:N % error on this line
6   \l_example_tl
7 \tl_build_end:N
8   \l_example_tl

```

Unfinished semi-built token list [W532]

A function `\tl_build_*begin:N` is called on a token list variable without calling a function `\tl_build_*end:N` later.

```

1 % file-wide warning
2 \tl_new:N
3   \l_example_tl
4 \tl_build_begin:N
5   \l_example_tl

```

Caveats

The warnings and errors in this documents do not cover the complete `expl3` language. The caveats currently include the following areas, among others:

- Functions with “weird” (w) argument specifiers
- Symbolic evaluation of expansion functions [2, sections 5.4–5.10]
- Validation of parameters in (inline) functions (c.f. E425 and W426)
- Shorthands such as `\~` and `\|` in message texts [2, sections 11.4 and 12.1.3]
- Quotes in shell commands and file names [2, Section 10.7 and Chapter 12]
- Functions used outside their intended context:
 - `\sort_return_*`: outside comparison code [2, Section 6.1]
 - `\prg_return_*`: outside conditional functions [2, Section 9.1]
 - Predicates (`*_p:*`) outside boolean expressions [2, Section 9.3]
 - `*_map_break:*` outside a corresponding mapping [2, sections 9.8]
 - `\msg_line_*`, `\iow_char:N`, and `\iow_newline:` outside message text [2, sections 11.3 and 12.1.3]
 - `\iow_wrap_allow_break:` and `\iow_indent:n` outside wrapped message text [2, Section 12.1.4]
 - Token list and string variables without accessor functions `\tl_use:N` and `\str_use:N`
 - Boolean variable without an accessor function `\bool_to_str:N` outside boolean expressions [2, Section 21.4]

- Integer variable without an accessor function `\int_use:N` outside integer or floating point expressions [2, Section 21.4]
 - Dimension variable without an accessor function `\dim_use:N` outside dimension or floating point expressions [2, Section 26.7]
 - Skip variable without an accessor function `\skip_use:N` outside skip or floating point expressions [2, Section 26.14]
 - Muskip variable without an accessor function `\muskip_use:N` outside muskip or floating point expressions [2, Section 26.21]
 - Floating point variable without an accessor function `\fp_use:N` outside floating point expressions [2, Section 29.3]
 - Box variable without accessor functions `\box_use(_drop)?:N` or `\[hv]box_unpack(_drop)?:N`, or without a measuring function `\box_(dp|ht|wd|ht_plus_dp):*` outside dimension or floating point expressions [2, sections 35.2 and 35.3]
 - Coffin variable without accessor function `\coffin_typeset:Nnnnn` outside dimension or floating point expressions [2, Section 36.4]
 - Lonely variables of other types that may or may not have accessor functions
- Validation of literal expressions:
 - Comparison expressions in functions `*_compare(_p:n|:nT?F?)`
 - Regular expressions and replacement text [2, sections 8.1 and 8.2]
 - Boolean expressions [2, Section 9.3]
 - Integer expressions and bases [2, sections 21.1 and 21.8]
 - Dimension, skip, and muskip expressions [2, Chapter 26]
 - Floating point expressions [2, Section 29.12]
 - Color expressions [2, Chapter 37.3]
 - Validation of naming schemes and member access:
 - String encoding and escaping [2, Section 18.1]
 - Key–value interfaces [2, Chapter 27]:
 - * Are keys defined at the point of use or is the module or its subdivision set up to accept unknown keys? [2, sections 27.2, 27.5, and 27.6]
 - * Are inheritance parents, choices, multi-choices, and groups used in a key definition defined at points of use? [2, sections 27.1, 27.3, and 27.7]
 - Floating-point symbolic expressions and user-defined functions [2, sections 29.6 and 29.7]
 - Names of bitset indexes [2, Section 31.1]
 - BCP-47 language tags [2, Section 34.2]
 - Color support [2, Chapter 37]:
 - * Named colors [2, Section 37.4]

- * Color export targets [2, Section 37.8]
- * Color models and their families and params [2, sections 37.2 and 37.9]
- Function `\file_input_stop`: not used on its own line [2, Section 12.2.3]
- Exhaustively or fully expanding quarks and scan marks [2, Chapter 19]
- Bounds checking for accessing constant sequences and other sequences where the number of items can be easily bounded such as integer and floating point arrays [2, chapters 28 and 30]:
 - Index checking functions `*_range*:*` and `*_item*:*`
 - Endless loop checking in functions `*_step*:*` [2, Section 21.7]
 - Number of symbols in a value-to-symbol mapping [2, Section 21.8]
- Applying functions `\clist_remove_duplicates:N` and `\clist_if_in:*` to comma lists that contain `{`, `}`, or `*` [2, sections 23.3 and 23.4]
- Incorrect parameters to function `\char_generate:nn` [2, Section 24.1]
- Incorrect parameters to functions `\char_set_*code:nn` [2, Section 24.2]
- Using implicit tokens `\c_catcode_(letter|other)_token` or the token list `\c_catcode_active_tl` [2, Section 24.3]
- Validation of key–value interfaces [2, Chapter 27]:
 - Setting a key with some properties `.*_g?(set|put)*:*` should be validated similarly to calling the corresponding functions directly: Have the variables been declared, do they have the correct type, does the value have the correct type?
 - Do points of use always set keys with property `.value_required:n` and never set keys with property `.value_forbidden:n`?
- Horizontal box operation on a vertical box or vice versa [2, Chapter 35], similarly for coffins

References

- [1] Vít Starý Novotný. *Static analysis of expl3 programs (3). Design Specification*. To be released. URL: <https://witiko.github.io/Expl3-Linter-3/>.
- [2] The L^AT_EX Project. *The L^AT_EX3 interfaces*. The referenced version of the document is attached to this document. May 8, 2024. URL: <http://mirrors.ctan.org/macros/latex/required/l3kernel/interface3.pdf> (visited on 05/15/2024).
- [3] The L^AT_EX Project. *The L^AT_EX3 kernel. Style guide for code authors*. The referenced version of the document is attached to this document. Apr. 11, 2024. URL: <http://mirrors.ctan.org/macros/latex/required/l3kernel/l3styleguide.pdf> (visited on 05/08/2024).
- [4] The L^AT_EX Project. *The expl3 package and L^AT_EX3 programming*. The referenced version of the document is attached to this document. Apr. 11, 2024. URL: <http://mirrors.ctan.org/macros/latex/required/l3kernel/exp13.pdf> (visited on 05/08/2024).

- [5] Joseph Wright. Apr. 29, 2024. URL: <https://github.com/latex3/latex3/pull/1542#issuecomment-2082352499> (visited on 05/15/2024).

Index

E

E102, 5
E104, 6
E201, 7
E203, 7
E208, 7
E209, 8
E300, 8
E301, 8
E304, 9
E404, 11
E405, 11
E408, 12
E411, 13
E417, 17
E418, 17
E420, 18
E421, 18
E424, 24
E425, 24
E427, 25
E500, 26
E505, 29
E506, 30
E507, 31
E509, 33
E510, 34

E511, 34
E514, 36
E515, 36
E516, 36
E517, 37
E519, 38
E521, 39
E523, 40
E525, 41
E526, 41
E527, 41
E531, 43

S

S103, 6
S105, 6
S204, 7
S412, 14
S413, 15
S414, 16

T

T305, 9
T403, 11
T422, 19
T529, 42
T530, 43

W

W100, 4
W101, 5
W200, 6
W202, 7
W302, 8
W303, 9
W401, 10
W402, 10
W410, 12
W415, 16
W416, 17
W419, 17
W423, 24
W426, 25
W501, 27
W502, 28
W503, 29
W508, 32
W512, 35
W513, 35
W518, 37
W520, 38
W522, 39
W524, 40
W528, 42
W532, 44