Warnings and errors for the expl3 analysis tool

Vít Starý Novotný

2025-06-24

Contents

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 idenfitiers:

- 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.

```
% file-wide warning
\tl_new:N
  \g_example_tl
\tl_gset:Nn
  \g_example_tl
  { Hello,~ }
\tl_gput_right:Nn
  \g_example_tl
  { world! }
\tl_use:N
  \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.

```
\input expl3-generic
\ExplSyntaxOff  % warning on this line
\ExplSyntaxOn
\tl_new:N
  \g_example_tl
\tl_gset:Nn
  \g_example_tl
  { Hello,~ }
\ExplSyntaxOn  % warning on this line
\tl_gput_right:Nn
  \g_example_tl
  { world! }
\tl_use:N
  \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.

```
\ProvidesExplFile{example.tex}{2024-04-09}{1.0.0}{An example
↪  file}
\tl_new:N
  \g_example_tl
\tl_gset:Nn
  \g_example_tl
  { Hello,~ }
\tl_gput_right:Nn
  \g_example_tl
  { world! }
```

```
\ExplSyntaxOff
\tl_use:N  % error on this line
  \g_example_tl  % error on this line
```

Line too long [S103]

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

```
This line is not very long, because it is 80 characters long, not
↪  81 characters.
This line is overly long, because it is 81 characters long
↪  excluding the comment.  % warning on this line
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`.

```
\ProvidesExplPackage
  {example.sty}{2024-04-09}{1.0.0}{An example package}
\ExplSyntaxOff
\ProvidesExplClass  % error on this line
  {example.cls}{2024-04-09}{1.0.0}{An example class}
```

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.

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

The above example has been taken from The LATEX Project [2, Chapter 24].

Unknown argument specifiers [E201]

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

```
\cs_new:Nn
  \example:bar  % error on this line
  { foo }
  { bar }
  { baz }
```

Deprecated control sequences [W202]

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

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

Missing stylistic whitespaces [S204]

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

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

Malformed function name [S205]

Some function have names that are not in the format \⟨*module*⟩_⟨*description*⟩:⟨*arg-spec*⟩ [4, Section 3.2].

```
\cs_new:Nn
  \description:  % warning on this line
  { foo }
```

```
\cs_gset:Npn
  \module__description:  % warning on this line
  { foo }
```

```
\cs_set_eq:NN
  \_module_description:  % warning on this line
  \example_foo:
```

```
\cs_generate_from_arg_count:NNnn
  \__module_description:
  \cs_new:Npn
  { 0 }
  { foo }
```

This also extends to conditional functions:

```
\prg_new_conditional:Nn
  \description:  % warning on this line
  { p, T, F, TF }
  { foo }
```

```
\prg_gset_conditional:Npn
  \module__description:  % warning on this line
  { p, T, F, TF }
  { foo }
```

```
\prg_set_eq_conditional:NNn
  \_module_description:  % warning on this line
  \example_foo:
  { p, T, F, TF }
```

Malformed variable or constant name [S206]

Some expl3 variables and constants have names that are not in the format \⟨*scope*⟩_⟨*module*⟩_⟨*description*⟩_⟨*type*⟩ [4, Section 3.2], where the ⟨*module*⟩ part is optional.

```
\tl_new:N
  \g_description_tl  % warning on this line
\box_new:N
  \l__description_box  % warning on this line
\int_const:Nn
  \c_description  % warning on this line
  { 123 }
```

```
\regex_new:N
  \g_module_description_regex
\coffin_new:N
  \l_module_description_coffin
\str_const:Nn
  \c__module_description_str
  { foo }
```

An exception is made for scratch variables [2, Section 1.1.1]:

```
\tl_use:N
  \l_tmpa_tl
\int_gset:Nn
  \g_tmpb_int
  { 1 + 2 }
\str_show:N
  \g_tmpa_str
\bool_set_true:N
  \l_tmpa_bool
```

Malformed quark or scan mark name [S207]

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

```
\quark_new:N
  \foo_bar  % error on this line
```

```
\quark_new:N
  \q_foo_bar
```

```
\scan_new:N
  \foo_bar  % error on this line
```

```
\scan_new:N
  \s_foo_bar
```

Too many closing braces [E208]

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

```
\tl_new:N
  \g_example_tl
\tl_gset:Nn
  \g_example_tl
  { Hello,~ } }  % error on this line
```

Invalid characters [E209]

An expl3 part of the input file contains invalid characters.

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

3 Syntactic analysis

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

Unexpected function call argument [E300]

A function is called with an unexpected argument.

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

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

```
\cs_new:Nn
  \example_foo:n
  { foo~#1 }
\cs_new:Nn
  \example_bar:
  { \example_foo:n }
\cs_new:Nn
  \example_baz:
  {
    \example_bar:
      { bar }
  }
```

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:

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

Unbraced n-type function call argument [W302]

An n-type function call argument is unbraced:

```
\tl_set:No
  \l_tmpa_tl
  \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:

```
\cs_new:Nn
  { \example_foo:n }  % warning on this line
  { 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:

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

```
\cs_new:Npn
  \example_foo:nnn
  #1#2#3
  { 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].

```
\cs_new:Nn
  \module_foo:n
  { #1 }
\cs_generate_variant:Nn
  \module_foo:n
  { V, v }
\module_foo:V
  \c_false_bool  % error on this line
\module_foo:v
  { 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 is defined but unused.

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

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

Unused private function variant [W402]

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

```
\cs_new:Nn
  \__module_foo:n
  { bar~#1 }
\cs_generate_variant:Nn  % warning on this line
  \__module_foo:n
  { V }
\__module_foo:n
  { baz }
```

```
\prg_new_conditional:Nnn
  \__module_foo:n
  { p, T, F, TF }
  { \prg_return_true: }
\prg_generate_conditional_variant:Nnn  % warning on this line
  \__module_foo:n
  { V }
  { TF }
\__module_foo:nTF
  { foo }
  { bar }
  { 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`].

```
\cs_new:Nn
  \module_foo:Nn
  { bar }
\cs_generate_variant:Nn  % error on this line
  \module_foo:Nn
  { Nnn }
```

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

```
\cs_new:Nn
  \module_foo:Nn
  { bar }
\cs_generate_variant:Nn
  \module_foo:Nn
  { cn }
```

```
\cs_generate_variant:Nn
  \module_foo:cn
  { cx }
\cs_generate_variant:Nn  % error on this line
  \module_foo:cx
  { Ne }
```

Protected predicate function [E404]

A protected predicate function is defined.

```
\prg_new_protected_conditional:Nnn
  \module_foo:
  { p }
  { \prg_return_true: }
```

Function variant for an undefined function [E405]

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

```
\cs_new:Nn
  \module_foo:n
  { bar }
\cs_generate_variant:Nn  % error on this line
  \module_bar:n
  { V }
```

```
\prg_new_conditional:Nnn
  \module_foo:n
  { p, T, F }
  { \prg_return_true: }
\prg_generate_conditional_variant:Nnn  % error on this line
  \module_bar:n
  { V }
  { T }
\prg_generate_conditional_variant:Nnn  % error on this line
  \module_foo:n
  { V }
  { TF }
```

Calling an undefined function [E408]

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

```
\module_foo:  % error on this line
```

```
\cs_new:Nn
  \module_foo:n
```

```
  { bar~#1 }
\tl_set:Nn
  \l_tmpa_tl
  { baz }
\module_foo:V  % error on this line
  \l_tmpa_tl
```

```
\prg_new_conditional:Nnn
  \module_foo:n
  { p, T, F, TF }
  { \prg_return_true: }
\prg_generate_conditional_variant:Nnn
  \module_foo:n
  { V }
  { T }
\module_foo:VTF  % error on this line
  \l_tmpa_tl
  { foo }
  { 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`].

```
\cs_new:Nn
  \module_foo:Nn
  { bar }
\cs_generate_variant:Nn  % warning on this line
  \module_foo:Nn
  { nn }
\cs_generate_variant:Nn  % warning on this line
  \module_foo:Nn
  { NN }
\cs_generate_variant:Nn  % warning on this line
  \module_foo:Nn
  { vc }
```

Indirect function definition from an undefined function [E411]

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

```
\cs_new:Nn
  \module_foo:n
  { bar~#1 }
\cs_new_eq:NN
  \module_bar:n
```

```
  \module_foo:n
\cs_new_eq:NN
  \module_baz:n
  \module_bar:n
\module_baz:n
  { foo }
```

```
\prg_new_conditional:Nnn
  \module_foo:n
  { p, T, F, TF }
  { \prg_return_true: }
\cs_new_eq:NN
  \module_bar:nTF
  \module_foo:nTF
\cs_new_eq:NN
  \module_baz:nTF
  \module_bar:nTF
\module_baz:nTF
  { foo }
  { bar }
  { baz }
```

```
\cs_new:Nn
  \module_foo:n
  { bar~#1 }
\cs_new_eq:NN  % error on this line
  \module_baz:n
  \module_bar:n
\module_baz:n
  { foo }
```

```
\prg_new_conditional:Nnn
  \module_foo:n
  { p, T, F, TF }
  { \prg_return_true: }
\cs_new_eq:NN  % error on this line
  \module_baz:nTF
  \module_bar:nTF
\module_baz:nTF
  { foo }
  { bar }
  { baz }
```

4.2 Variables and constants

Unused variable or constant [W412]

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

```
\tl_new:N  % warning on this line
  \g_declared_but_undefined_tl
```

```
\tl_new:N  % warning on this line
  \g_defined_but_unused_tl
\tl_gset:Nn
  \g_defined_but_unused_tl
  { foo }
```

```
\tl_new:N
  \g_defined_but_unused_tl
\tl_gset:Nn
  \g_defined_but_unused_tl
  { foo }
\tl_use:N
  \g_defined_but_unused_tl
```

```
\tl_const:Nn  % warning on this line
  \c_defined_but_unused_tl
  { foo }
```

```
\tl_const:Nn
  \c_defined_but_unused_tl
  { foo }
\tl_use:N
  \c_defined_but_unused_tl
```

Setting an undeclared variable [W413]

An undeclared variable is set.

```
\tl_gset:Nn  % warning on this line
  \g_example_tl
  { bar }
```

Setting a constant [E414]

A constant is set.

```
\tl_gset:Nn  % error on this line
  \c_example_tl
  { bar }
```

Using a token list variable or constant without an accessor [W415]

A token list variable or constant is used without an accessor function.

```
\tl_set:Nn
  \l_tmpa_tl
  { world }
Hello,~\l_tmpa_tl!  % warning on this line
Hello,~\tl_use:N \l_tmpa_tl !
```

This also applies to subtypes of token lists such as strings and comma-lists:

```
\str_set:Nn
  \l_tmpa_str
  { world }
Hello,~\l_tmpa_str!  % warning on this line
Hello,~\str_use:N \l_tmpa_str !
```

```
\clist_set:Nn
  \l_tmpa_clist
  { world }
Hello,~\l_tmpa_clist!  % warning on this line
Hello,~\clist_use:Nn \l_tmpa_clist { and } !
```

Using non-token-list variable or constant without an accessor [E416]

A non-token-list variable or constant is used without an accessor function.

```
Hello,~\l_tmpa_seq!  % error on this line
Hello,~\seq_use:Nn \l_tmpa_seq { and } !
```

Note that boolean and integer variables may be used without accessor functions in boolean and integer expressions, respectively. Therefore, we may want to initially exclude them from this check to prevent false positives.

Multiply declared variable or constant [E417]

A variable or constant is declared multiple times.

```
\tl_new:N
  \g_example_tl
\tl_new:N  % error on this line
  \g_example_tl
```

```
\tl_const:Nn
  \c_example_tl
  { foo }
\tl_const:Nn  % error on this line
  \c_example_tl
  { bar }
```

Using an undefined variable or constant [E418]

A variable or constant is used but undeclared or undefined.

```
\tl_use:N  % error on this line
  \g_undeclared_tl
```

```
\tl_new:N
  \g_declared_but_undefined_tl
\tl_use:N  % error on this line
  \g_declared_but_undefined_tl
```

```
\tl_new:N
  \g_defined_tl
\tl_gset:Nn
  \g_defined_tl
  { foo }
\tl_use:N
  \g_defined_tl
```

```
\tl_use:N  % error on this line
  \c_undefined_tl
```

```
\tl_const:Nn
  \c_defined_tl
  { foo }
\tl_use:N
  \c_defined_tl
```

Locally setting a global variable [E419]

A global variable is locally set.

```
\tl_new:N
  \g_example_tl
\tl_set:Nn  % error on this line
  \g_example_tl
  { foo }
```

Globally setting a local variable [E420]

A local variable is globally set.

```
\tl_new:N
  \l_example_tl
\tl_gset:Nn  % error on this line
  \l_example_tl
  { foo }
```

Using a variable of an incompatible type [T421]

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

```
\tl_new:N
  \l_example_str  % error on this line
```

```
\tl_new:N
  \l_example_tl
\tl_count:N
  \l_example_tl
\str_count:N
  \l_example_tl
\seq_count:N
  \l_example_tl  % error on this line
\clist_count:N
  \l_example_tl  % error on this line
\prop_count:N
  \l_example_tl  % error on this line
\intarray_count:N
  \l_example_tl  % error on this line
\fparray_count:N
  \l_example_tl  % error on this line
```

```
\str_new:N
  \l_example_str
\tl_count:N
  \l_example_str
\str_count:N
  \l_example_str
\seq_count:N
  \l_example_str  % error on this line
\clist_count:N
  \l_example_str  % error on this line
\prop_count:N
  \l_example_str  % error on this line
\intarray_count:N
  \l_example_str  % error on this line
\fparray_count:N
  \l_example_str  % error on this line
```

```
\int_new:N
  \l_example_int
\tl_count:N
  \l_example_int  % error on this line
\str_count:N
  \l_example_int  % error on this line
\seq_count:N
```

```
  \l_example_int  % error on this line
\clist_count:N
  \l_example_int  % error on this line
\prop_count:N
  \l_example_int  % error on this line
\intarray_count:N
  \l_example_int  % error on this line
\fparray_count:N
  \l_example_int  % error on this line
```

```
\seq_new:N
  \l_example_seq
\tl_count:N
  \l_example_seq  % error on this line
\str_count:N
  \l_example_seq  % error on this line
\seq_count:N
  \l_example_seq
\clist_count:N
  \l_example_seq  % error on this line
\prop_count:N
  \l_example_seq  % error on this line
\intarray_count:N
  \l_example_seq  % error on this line
\fparray_count:N
  \l_example_seq  % error on this line
```

```
\clist_new:N
  \l_example_clist
\tl_count:N
  \l_example_clist  % error on this line
\str_count:N
  \l_example_clist  % error on this line
\seq_count:N
  \l_example_clist  % error on this line
\clist_count:N
  \l_example_clist
\prop_count:N
  \l_example_clist  % error on this line
\intarray_count:N
  \l_example_clist  % error on this line
\fparray_count:N
  \l_example_clist  % error on this line
```

```
\clist_new:N
  \l_example_prop
\tl_count:N
```

```
  \l_example_prop  % error on this line
\str_count:N
  \l_example_prop  % error on this line
\seq_count:N
  \l_example_prop  % error on this line
\clist_count:N
  \l_example_prop  % error on this line
\prop_count:N
  \l_example_prop
\intarray_count:N
  \l_example_prop  % error on this line
\fparray_count:N
  \l_example_prop  % error on this line
```

```
\intarray_new:Nn
  \g_example_intarray
  { 5 }
\tl_count:N
  \g_example_intarray  % error on this line
\str_count:N
  \g_example_intarray  % error on this line
\seq_count:N
  \g_example_intarray  % error on this line
\clist_count:N
  \g_example_intarray  % error on this line
\prop_count:N
  \g_example_intarray  % error on this line
\intarray_count:N
  \g_example_intarray
\fparray_count:N
  \g_example_intarray  % error on this line
```

```
\fparray_new:Nn
  \g_example_fparray
  { 5 }
\tl_count:N
  \g_example_fparray  % error on this line
\str_count:N
  \g_example_fparray  % error on this line
\seq_count:N
  \g_example_fparray  % error on this line
\clist_count:N
  \g_example_fparray  % error on this line
\prop_count:N
  \g_example_fparray  % error on this line
\intarray_count:N
  \g_example_fparray  % error on this line
```

```
\fparray_count:N
  \g_example_fparray
```

```
\ior_new:N
  \l_example_ior
\iow_open:Nn
  \l_example_ior  % error on this line
  { example }
```

```
\clist_new:N
  \l_example_clist
\tl_set:Nn
  \l_tmpa_tl
  { foo }
\clist_set_eq:NN
  \l_example_clist
  \l_tmpa_tl  % error on this line
```

```
\tl_set:Nn
  \l_tmpa_tl
  { foo }
\seq_set_from_clist:NN
  \l_tmpa_seq
  \l_tmpa_tl  % error on this line
```

```
\tl_set:Nn
  \l_tmpa_tl
  { foo }
\regex_set:Nn
  \l_tmpa_regex
  { foo }
\int_set:Nn
  \l_tmpa_int
  { 1 + 2 }
\regex_show:N
  \l_tmpa_tl
\regex_show:N
  \l_tmpa_regex
\regex_show:N
  \l_tmpa_int  % error on this line
```

```
\tl_set:Nn
  \l_tmpa_tl
  { foo }
\int_set_eq:NN
  \l_tmpa_int
  \l_tmpa_tl  % error on this line
```

4.3 Messages

Unused message [W422]

A message is defined but unused.

```
\msg_new:nnn  % warning on this line
  { foo }
  { bar }
  { baz }
```

```
\msg_new:nnn
  { bar }
  { bar }
  { baz }
\msg_info:nn
  { bar }
  { bar }
```

Setting an undefined message [W423]

A message is set but undefined.

```
\msg_set:nnn  % error on this line
  { foo }
  { bar }
  { baz }
```

```
\msg_new:nnn
  { foo }
  { bar }
  { baz }
\msg_set:nnn
  { foo }
  { bar }
  { baz }
```

Multiply defined message [E424]

A message is defined multiple times.

```
\msg_new:nnn
  { foo }
  { bar }
  { baz }
\msg_new:nnn  % error on this line
  { foo }
  { bar }
  { baz }
```

Using an undefined message [E425]

A message is used but undefined.

```
\msg_info:nn
  { foo }
  { bar }
```

Incorrect parameters in message text [E426]

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

```
\msg_new:nnn
  { foo }
  { bar }
  { #5 }  % error on this line
```

```
\msg_new:nnnn
  { foo }
  { bar }
  { #4 }
  { #5 }  % error on this line
```

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].

```
\cs_new:Nn
  \example_foo:
  { \prg_return_true: }
\tl_sort:nN
  { { foo } { bar } }
  \example_foo:TF
```

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.

```
\cs_new:Nn
  \module_foo:
  { bar }
\cs_new:Nn  % error on this line
  \module_foo:
  { bar }
```

```
\cs_new:Nn
  \module_foo:
  { bar }
\cs_undefine:N
  \module_foo:
\cs_new:Nn
  \module_foo:
  { bar }
```

```
\cs_new:Nn
  \module_foo:
  { bar }
\cs_gset:Nn
  \module_foo:
  { bar }
```

```
\prg_new_conditional:Nnn
  \module_foo:
  { p, T, F, TF }
  { \prg_return_true: }
\prg_new_conditional:Nnn  % error on this line
  \module_foo:
  { p, T, F, TF }
  { \prg_return_true: }
```

```
\prg_new_conditional:Nnn
  \module_foo:
  { p, T, F, TF }
  { \prg_return_true: }
\cs_undefine:N
  \module_foo_p:
\cs_undefine:N
  \module_foo:T
\cs_undefine:N
  \module_foo:F
```

```
\cs_undefine:N
  \module_foo:TF
\prg_new_conditional:Nnn
  \module_foo:
  { p, T, F, TF }
  { \prg_return_true: }
```

```
\prg_new_conditional:Nnn
  \module_foo:
  { p, T, F, TF }
  { \prg_return_true: }
\prg_gset_conditional:Nnn
  \module_foo:
  { p, T, F, TF }
  { \prg_return_true: }
```

Multiply defined function variant [W501]

A function or conditional function variant is defined multiple times.

```
\cs_new:Nn
  \module_foo:n
  { bar~#1 }
\cs_generate_variant:Nn
  \module_foo:n
  { V }
\cs_generate_variant:Nn  % warning on this line
  \module_foo:n
  { o, V }
```

```
\cs_new:Nn
  \module_foo:n
  { bar~#1 }
\cs_generate_variant:Nn
  \module_foo:n
  { V }
\cs_undefine:N
  \module_foo:V
\cs_generate_variant:Nn
  \module_foo:n
  { o, V }
```

```
\prg_new_conditional:Nnn
  \module_foo:n
  { p, T, F, TF }
  { \prg_return_true: }
\prg_generate_conditional_variant:Nnn
```

```
  \module_foo:n
  { V }
  { TF }
\prg_generate_conditional_variant:Nnn  % warning on this line
  \module_foo:n
  { o, V }
  { TF }
```

```
\prg_new_conditional:Nnn
  \module_foo:n
  { p, T, F, TF }
  { \prg_return_true: }
\prg_generate_conditional_variant:Nnn
  \module_foo:n
  { V }
  { TF }
\cs_undefine:N
  \module_foo:VTF
\prg_generate_conditional_variant:Nnn
  \module_foo:n
  { o, V }
  { TF }
```

Unused private function [W502]

A private function or conditional function is defined but all its calls are unreachable.[1]

```
\cs_new:Nn  % warning on this line
  \__module_foo:
  { bar }
\cs_new:Nn
  \__module_baz:
  { \__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 all its calls are unreachable.

```
\cs_new:Nn
  \__module_foo:n
```

[1] Code is unreachable if it is only reachable through private functions which that are either unused or also unreachable.

```
  { bar~#1 }
\cs_new:Nn
  \__module_baz:
  {
    \tl_set:Nn
      \l_tmpa_tl
      { baz }
    \__module_foo:V
      \l_tmpa_tl
  }
\cs_generate_variant:Nn  % warning on this line
  \__module_foo:n
  { V }
\__module_foo:n
  { 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 [E504]

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

```
\cs_new:Nn
  \module_foo:n
  { bar }
\cs_generate_variant:Nn
  \module_foo:n
  { V }
```

```
\cs_generate_variant:Nn  % error on this line
  \module_foo:n
  { V }
\cs_new:Nn
  \module_foo:n
  { bar }
```

```
\cs_new:Nn
  \module_foo:n
  { bar }
\cs_undefine:N
  \module_foo:n
\cs_generate_variant:Nn  % error on this line
  \module_foo:n
  { V }
```

```
\prg_new_conditional:Nnn
```

```
  \module_foo:n
  { p, T, F, TF }
  { \prg_return_true: }
\prg_generate_conditional_variant:Nnn
  \module_foo:n
  { V }
  { TF }
```

```
\prg_generate_conditional_variant:Nnn  % error on this line
  \module_foo:n
  { V }
  { TF }
\prg_new_conditional:Nnn
  \module_foo:n
  { p, T, F, TF }
  { \prg_return_true: }
```

```
\prg_new_conditional:Nnn
  \module_foo:n
  { p, T, F, TF }
  { \prg_return_true: }
\cs_undefine:N
  \module_foo:nTF
\prg_generate_conditional_variant:Nnn  % error on this line
  \module_foo:n
  { V }
  { 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 [E505]

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

```
\module_foo:  % error on this line
\cs_new:Nn
  \module_foo:
  { bar }
```

```
\cs_new:Nn
  \module_foo:
  { bar }
\cs_undefine:N
  \module_foo:
\module_foo:  % error on this line
```

```
\cs_new:Nn
  \module_foo:n
  { bar~#1 }
\tl_set:Nn
  \l_tmpa_tl
  { baz }
\module_foo:V  % error on this line
  \l_tmpa_tl
\cs_generate_variant:Nn
  \module_foo:n
  { 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 [E506]

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

```
\cs_new:Nn
  \module_foo:n
  { bar~#1 }
\cs_new_eq:NN  % error on this line
  \module_baz:n
  \module_bar:n
\cs_new_eq:NN
  \module_bar:n
  \module_foo:n
\module_baz:n
  { foo }
```

```
\cs_new:Nn
  \module_foo:n
  { bar~#1 }
\cs_new_eq:NN
  \module_bar:n
  \module_foo:n
\cs_undefine:N
  \module_bar:n
\cs_new_eq:NN  % error on this line
  \module_baz:n
  \module_bar:n
\module_baz:n
  { foo }
```

```
\prg_new_conditional:Nnn
  \module_foo:n
```

```
  { p, T, F, TF }
  { \prg_return_true: }
\cs_new_eq:NN  % error on this line
  \module_baz:nTF
  \module_bar:nTF
\cs_new_eq:NN
  \module_bar:nTF
  \module_foo:nTF
\module_baz:nTF
  { foo }
  { bar }
  { baz }
```

```
\prg_new_conditional:Nnn
  \module_foo:n
  { p, T, F, TF }
  { \prg_return_true: }
\cs_new_eq:NN
  \module_bar:nTF
  \module_foo:nTF
\cs_undefine:N
  \module_bar:nTF
\cs_new_eq:NN  % error on this line
  \module_baz:nTF
  \module_bar:nTF
\module_baz:nTF
  { foo }
  { bar }
  { 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 [W507]

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

```
\cs_gset:N  % warning on this line
  \module_foo:
  { foo }
\cs_new:Nn
  \module_foo:
  { bar }
```

```
\cs_new:Nn
  \module_foo:
  { bar }
\cs_undefine:N
```

```
  \module_foo:
\cs_gset:N  % warning on this line
  \module_foo:
  { foo }
```

Unexpandable or restricted-expandable boolean expression [E508]

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

```
\cs_new_protected:N
  \example_unexpandable:
  {
    \tl_set:Nn
      \l_tmpa_tl
      { bar }
    \c_true_bool
  }
\cs_new:N
  \example_restricted_expandable:
  {
    \bool_do_while:Nn
      \c_false_bool
      { }
    \c_true_bool
  }
\cs_new_protected:N
  \example_expandable:
  { \c_true_bool }
\bool_set:Nn
  \l_tmpa_bool
  { \example_unexpandable: }  % error on this line
\bool_set:Nn
  \l_tmpa_bool
  { \example_restricted_expandable: }  % error on this line
\bool_set:Nn
  \l_tmpa_bool
  { \example_expandable: }
```

Expanding an unexpandable function [E509]

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

```
\cs_new_protected:N
  \example_unexpandable:
  {
```

```
    \tl_set:Nn
      \l_tmpa_tl
      { bar }
  }
\cs_new:N
  \module_foo:n
  { #1 }
\cs_generate_variant:Nn
  \module_foo:n
  { x, e, f }
\module_foo:n
  { \example_unexpandable: }
\module_foo:x
  { \example_unexpandable: }  % error on this line
\module_foo:e
  { \example_unexpandable: }  % error on this line
\module_foo:f
  { \example_unexpandable: }  % error on this line
```

Fully-expanding a restricted-expandable function [E510]

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

```
\cs_new:N
  \example_restricted_expandable:
  {
    \int_to_roman:n
      { 1 + 2 }
  }
\cs_new:N
  \module_foo:n
  { #1 }
\cs_generate_variant:Nn
  \module_foo:n
  { x, e, f }
\module_foo:n
  { \example_restricted_expandable: }
\module_foo:x
  { \example_restricted_expandable: }
\module_foo:e
  { \example_restricted_expandable: }
\module_foo:f
  { \example_restricted_expandable: }  % error on this line
```

Defined an expandable function as protected [W511]

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

```
\cs_new_protected:Nn  % warning on this line
  \example_expandable:
  { foo }
```

```
\prg_new_protected_conditional:Nnn  % warning on this line
  \example_expandable:
  { T, F, TF }
  { \prg_return_true: }
```

Defined an unexpandable function as unprotected [W512]

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

```
\cs_new:Nn  % warning on this line
  \example_unexpandable:
  {
    \tl_set:Nn
      \l_tmpa_tl
      { bar }
  }
```

```
\prg_new_conditional:Nnn  % warning on this line
  \example_unexpandable:
  { p, T, F, TF }
  {
    \tl_set:Nn
      \l_tmpa_tl
      { bar }
    \prg_return_true:
  }
```

Conditional function with no return value [E513]

A conditional functions has no return value.

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

```
\prg_new_conditional:Nnn
  \example_has_return_value:
  { p, T, F, TF }
  { \example_foo: }
\cs_new:Nn
  \example_foo:
  { \prg_return_true: }
```

Comparison code with no return value [E514]

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

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

```
\clist_set:Nn
  \l_foo_clist
  { 3 , 01 , -2 , 5 , +1 }
\clist_sort:Nn
  \l_foo_clist
  { \example_foo: }
\cs_new:Nn
  \example_foo:
  {
    \int_compare:nNnTF { #1 } > { #2 }
      { \sort_return_swapped: }
      { \sort_return_same: }
  }
```

The above example has been taken from The LaTeX Project [2, Chapter 6].

Paragraph token in the parameter of a "nopar" function [E515]

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

```
\cs_new_nopar:Nn
  \example_foo:n
  { #1 }
\cs_new:nn
  \example_bar:n
  {
    \example_foo:n
```

```
      { #1 }
  }
\example_bar:n
  {
    foo
    \par  % error on this line
    bar
  }
```

5.2 Variables and constants

Unused variable or constant [W516]

A variable or a constant is declared and perhaps defined but all its uses are unreachable.

```
\tl_new:N  % warning on this line
  \g_defined_but_unreachable_tl
\tl_gset:Nn
  \g_defined_but_unreachable_tl
  { foo }
\cs_new:Nn
  \__module_baz:
  {
    \tl_use:N
      \g_defined_but_unreachable_tl
  }
```

This check is a stronger version of W412 and should only be emitted if W412 has not previously been emitted for this variable or constant.

Setting an undeclared variable [E517]

A variable is set before it has been declared.

```
\tl_gset:Nn  % error on this line
  \g_example_tl
  { bar }
\tl_new:N
  \g_example_tl
```

This check is a stronger version of W413 and should prevent W413 from being emitted for this variable.

Using an undefined variable or constant [E518]

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

```
\tl_new:N
  \g_example_tl
\tl_use:N  % error on this line
  \g_example_tl
\tl_gset:Nn
  \g_example_tl
  { foo }
```

This check is a stronger version of E418 and should only be emitted if E418 has not previously been emitted for this variable or constant.

5.3 Messages

Unused message [W519]

A message is defined but all its uses are unreachable.

```
\msg_new:nnn  % warning on this line
  { foo }
  { bar }
  { baz }
\cs_new:Nn
  \__module_baz:
  {
    \msg_info:nn
      { foo }
      { bar }
  }
```

This check is a stronger version of W422 and should only be emitted if W422 has not previously been emitted for this message.

Setting an undefined message [E520]

A message is set before it has been defined.

```
\msg_set:nnn  % error on this line
  { foo }
  { bar }
  { baz }
\msg_new:nnn
  { foo }
  { bar }
  { baz }
```

This check is a stronger version of W423 and should prevent W423 from being emitted for this message.

Using an undefined message [E521]

A message is used before it has been defined.

```
\msg_info:nn  % error on this line
  { foo }
  { bar }
\msg_new:nnn
  { foo }
  { bar }
  { baz }
```

This check is a stronger version of E425 and should only be emitted if E425 has not previously been emitted for this message.

Too few arguments supplied to message [E522]

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

```
\msg_new:nnn
  { foo }
  { bar }
  { #1~#2 }
\msg_info:nn  % error on this line
  { foo }
  { bar }
\msg_info:nnn  % error on this line
  { foo }
  { bar }
  { baz }
\msg_info:nnnn
  { foo }
  { bar }
  { baz }
  { baz }
```

Since a message can be redefined, we need to track the (possibly many) definitions that can be active when we display a message.

```
\msg_new:nnn
  { foo }
  { bar }
  { #1 }
\msg_set:nnn
  { foo }
  { bar }
  { baz }
```

```
\msg_info:nnn  % error on this line
  { foo }
  { bar }
  { baz }
```

```
\msg_new:nnn
  { foo }
  { bar }
  { #1 }
\msg_info:nnn
  { foo }
  { bar }
  { baz }
\msg_set:nnn
  { foo }
  { bar }
  { baz }
```

5.4 Input–output streams

Using an unopened or closed stream [E523]

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

```
\ior_new:N
  \l_example_ior
\ior_str_get:NN  % error on this line
  \l_example_ior
  \l_tmpa_tl
\ior_open:Nn
  \l_example_ior
  { example }
```

```
\ior_new:N
  \l_example_ior
\ior_open:Nn
  \l_example_ior
  { example }
\ior_close:N
  \l_example_ior
\ior_str_get:NN  % error on this line
  \l_example_ior
  \l_tmpa_tl
```

Multiply opened stream [E524]

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

```
\iow_new:N
  \l_example_iow
\iow_open:Nn
  \l_example_iow
  { foo }
\iow_open:Nn  % error on this line
  \l_example_iow
  { bar }
\iow_close:N
  \l_example_iow
```

Unclosed stream [W525]

A stream is opened but not closed.

```
% file-wide warning
\ior_new:N
  \l_example_ior
\ior_open:Nn
  \l_example_ior
  { example }
```

5.5 Piecewise token list construction

Building on a regular token list [T526]

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.

```
\tl_new:N
  \l_example_tl
\tl_build_put_right:Nn  % error on this line
  \l_example_tl
  { foo }
\tl_build_begin:N
  \l_example_tl
\tl_build_end:N
  \l_example_tl
```

```
\tl_new:N
  \l_example_tl
\tl_build_begin:N
```

```
  \l_example_tl
\tl_build_put_right:Nn
  \l_example_tl
  { foo }
\tl_build_end:N
  \l_example_tl
```

```
\tl_new:N
  \l_example_tl
\tl_build_begin:N
  \l_example_tl
\tl_build_end:N
  \l_example_tl
\tl_build_put_right:Nn  % error on this line
  \l_example_tl
  { foo }
```

Using a semi-built token list [T527]

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.

```
\tl_new:N
  \l_example_tl
\tl_use:N
  \l_example_tl
\tl_build_begin:N
  \l_example_tl
\tl_build_end:N
  \l_example_tl
```

```
\tl_new:N
  \l_example_tl
\tl_build_begin:N
  \l_example_tl
\tl_use:N
  \l_example_tl  % error on this line
\tl_build_end:N
  \l_example_tl
```

```
\tl_new:N
  \l_example_tl
\tl_build_begin:N
  \l_example_tl
\tl_build_end:N
  \l_example_tl
```

```
\tl_use:N
  \l_example_tl
```

Multiply started building a token list [E528]

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.

```
\tl_new:N
  \l_example_tl
\tl_build_begin:N
  \l_example_tl
\tl_build_begin:N  % error on this line
  \l_example_tl
\tl_build_end:N
  \l_example_tl
```

Unfinished semi-built token list [W529]

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

```
% file-wide warning
\tl_new:N
  \l_example_tl
\tl_build_begin:N
  \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. E426 and E522)
- 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]
  - Boolean variable without an accessor function `\bool_to_str:N` outside boolean expressions [2, Section 21.4] (see E416)
  - Integer variable without an accessor function `\int_use:N` outside integer or floating point expressions [2, Section 21.4] (see E416)
  - Dimension variable without an accessor function `\dim_use:N` outside dimension or floating point expressions [2, Section 26.7] (see E416)
  - Skip variable without an accessor function `\skip_use:N` outside skip or floating point expressions [2, Section 26.14] (see E416)
  - Muskip variable without an accessor function `\muskip_use:N` outside muskip or floating point expressions [2, Section 26.21] (see E416)
  - Floating point variable without an accessor function `\fp_use:N` outside floating point expressions [2, Section 29.3] (see E416)
  - 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]

- 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]

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 LaTeX Project. *The LaTeX3 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 LaTeX Project. *The LaTeX3 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 LaTeX Project. *The expl3 package and LaTeX3 programming*. The referenced version of the document is attached to this document. Apr. 11, 2024. URL: `http://mirrors.ctan.org/macros/latex/required/l3kernel/expl3.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