

The grading-scheme package

Maximilian Kessler*

Released 2022/02/22

version 0.1

Abstract

This package aims at an easy-to-use interface to typeset grading schemes, in particular grading-schemes of exercises of mathematical olympiads where multiple solutions have to be graded and might offer mutual exclusive ways of receiving points.

Contents

1	General notions	2
2	Usage	3
3	L^AT_EX3 interface	4
4	grading-scheme implementation	5
4.1	Dependencies and options	5
4.2	A simple logging module	5
4.3	Wrappers	6
4.4	Customizable backends	7
4.5	Resources	7
4.6	Pointers	8
4.7	Structs	15
4.8	Entries	15
4.9	Blocks	19
4.10	Elements	28
4.11	Facilities for populating data structures	34
4.12	Grading scheme environment	36
4.13	Implementing the pipe syntax	38
	Index	39

*ctan@maximilian-kessler.de

SUM	MAX	MAX	At least one correct solution		1 P.
			All correct solutions		1 P.
			Uniqueness proofs		
			SUM	First solution	
				Show that either $4 \mid p - 2$ or $3 \mid p - 4$.	3 P.
				Show that $m = 5$	2 P.
				Show that $n = 4$	1 P.
				Conclude solutions	1 P.
			SUM	Second solution	
				Show that p is even	2 P.
				Reduce to at most 30 candidates	3 P.
				Test all candidates	2 P.
		MIN(2)	SUM	Plausible Arguments	
				plausible argument that there are finitely many solutions	1 P.
				suggestion that divisibility by 4 is useful	1 P.
				suggesting that not both of m, n can be 'large'.	1 P.

Table 1: Example output of a grading scheme.

1 General notions

Probably, an example is fitting best at this point. If you want to set up a grading scheme, with the `grading-scheme` package all you have to do to get the output of [Table 1](#) is type the following:

```
\begin{gradingscheme}{sum}
| At least one correct solution & 1
| All correct solutions & 1
\begin{block}{max}
\begin{block}[Uniqueness proofs]{max}
\begin{block}[First solution]{sum}
| Show that either  $4 \mid p - 2$  or  $3 \mid p - 4$ . & 3
| Show that  $m = 5$  & 2
| Show that  $n = 4$  & 1
| Conclude solutions & 1
\end{block}
\end{block}
\begin{block}[Second solution]{sum}
| Show that  $p$  is even & 2
| Reduce to at most 30 candidates & 3
| Test all candidates & 2
\end{block}
\end{block}
\end{gradingscheme}
```

```

\begin{block}{min(2)}
  \begin{block}[Plausible Arguments]{sum}
    | plausible argument that there are finitely many solutions & 1
    | suggestion that divisibility by 4 is useful & 1
    | suggesting that not both of  $m$ ,  $n$  can be 'large'. & 1
  \end{block}
\end{block}
\end{block}
\end{gradingscheme}

```

Note in particular that the correct width of the rows and columns are automatically adjusted.

For us, a grading scheme consists of several statements, each worth some specific number of points, that will be combined into a final number of points using a nested expression of operations, typically sums, maximums or minimums.

An abstract grammar defining our notions is defined in [Table 2](#). The meanings of $\langle \textit{balanced text} \rangle$ and $\langle \textit{number} \rangle$ are as usual.

```

grading scheme := block
  block := [ $\langle \textit{block description} \rangle$ ],  $\langle \textit{block operation} \rangle$ ,  $\langle \textit{element list} \rangle$ 
  element list :=  $\langle \textit{element} \rangle$  |  $\langle \textit{element} \rangle$  :  $\langle \textit{element list} \rangle$ 
  element :=  $\langle \textit{block} \rangle$  |  $\langle \textit{entry} \rangle$ 
  entry :=  $\langle \textit{entry description} \rangle$ ,  $\langle \textit{entry points} \rangle$ 
block description :=  $\langle \textit{balanced text} \rangle$ 
block operation :=  $\langle \textit{balanced text} \rangle$ 
entry description :=  $\langle \textit{balanced text} \rangle$ 
entry points :=  $\langle \textit{number} \rangle$ 

```

Table 2: Abstract grammar for a grading scheme

The package facilitates specifying a grading scheme according to this grammar and typesets these.

2 Usage

Load the package as usual by

```
\usepackage[pipe]{grading-scheme}
```

The `pipe` option is of course optional and the only currently supported option. It activates the pipe syntax explained later.

```

gradingscheme \begin{gradingscheme}[\langle \textit{description text} \rangle]{\langle \textit{operator} \rangle}
\end{gradingscheme}

```

The main environment provided by this package. Inside the grading scheme, any number of blocks and entries can be specified. The grading scheme environment will act like the outer-most block and typeset all its contents.

Grading schemes cannot contain each other. As they are implemented as `tabulars`, you can (and maybe should) put a grading scheme into a `table` environment as if it were a regular `tabular`.

T_EXhackers note: Actually, the gradingscheme environment just sets up some hooks that further block environments and entries will fill and typesets these at the end of the environment.

You could technically put any regular text inside the grading scheme and this will be typeset as if it had been specified before the grading scheme. However, this is definitely not recommended.

block `\begin{block}[\langle description text \rangle]{\langle operator \rangle}`
 `\end{block}`

The $\langle description text \rangle$ will be shown in a separate row on top of the block. If not specified, no such row is inserted. The $\langle operator \rangle$ is printed in small capitals (and rotated) in a column that spans the left side of the block, indicating that this operation is applied to the block.

The block will be an element of the outer block or grading scheme it is contained in.

Inside the block, any number of blocks or entries can be specified (in order) that will be typeset as members of this block.

A block environment can only be used inside a gradingscheme. Block environments can contain each other in balanced form.

\entry `\entry{\langle entry description \rangle}{\langle points \rangle}`

Specifies an entry with given description and points. The entry is typeset as part of the block/grading scheme containing it.

—
 | `| \langle entry description \rangle & \langle points \rangle \n`
 —

This is an alternative form to specify an entry. The `\n` is to be interpreted as a newline in the source file, so `|` reads until the end of the line.

This is equivalent to calling `\entry{\langle entry description \rangle}{\langle points \rangle}`

The package option `pipe` has to be specified to activate this syntax.

T_EXhackers note: The `|` character (pipe) is made an active character for the scope of each gradingscheme environment. Thus, this syntax only works if the gradingscheme is read in expansion mode by L^AT_EX, since otherwise the category code will be messed up.

3 L^AT_EX3 interface

There is also a L^AT_EX3 interface for dealing with this package that can deal with all of the mentioned notions separately, technically allowing you to combine them in other ways or formatting them into your custom tables.

These functions are for now not documented separately, since the author thinks that such use cases would be rather rare.

The L^AT_EX3 interface is, however, *not* considered to be stable yet, so if you really have to rely on this, feel free to contact the author.

4 grading-scheme implementation

```

1 <*package>
2 <@@=grading_scheme>

```

4.1 Dependencies and options

Currently, there is only one option

`\g__grading_scheme_pipe_syntax_bool` This will toggle the pipe syntax option of the package.

```

3 \bool_new:N \g__grading_scheme_pipe_syntax_bool
4
5 (End definition for \g__grading_scheme_pipe_syntax_bool.)
6
7 \keys_define:nn { grading-scheme }
8 {
9   pipe      .bool_gset:N = \g__grading_scheme_pipe_syntax_bool,
10  pipe      .default:n = { true },
11  unknown .code:n =
12    {
13      \msg_error:nnn { grading-scheme } { unknown-option } { \l_keys_key_str }
14    }
15 }
16 \msg_new:nnn { grading-scheme } { unknown-option }
17 {
18   Unknown ~ option ~ '#1'.
19 }
20 \RequirePackage { l3keys2e }
21 \ProcessKeysOptions { grading-scheme }
22 \RequirePackage{multirow}
23 \RequirePackage{rotating}

```

4.2 A simple logging module

```

21 <*log>

```

We provide a simple log/debugging facility for this package.

`\g__grading_scheme_log_depth_int` Stores the log depth for indentation in the log file.

```

22 \int_new:N \g__grading_scheme_log_depth_int

```

(End definition for `\g__grading_scheme_log_depth_int`.)

`__grading_scheme_log_incr:` Increments the log depth

```

23 \cs_new:Npn \__grading_scheme_log_incr:
24 {
25   \int_gincr:N \g__grading_scheme_log_depth_int
26 }

```

(End definition for `__grading_scheme_log_incr:.`)

`__grading_scheme_log_decr:` Decrements the log depth

```

27 \cs_new:Npn \__grading_scheme_log_decr:
28 {
29   \int_gdecr:N \g__grading_scheme_log_depth_int
30 }

```

(End definition for `_grading_scheme_log_decr:.`)

```

\_grading_scheme_log:n Logs to the terminal and log file using the current depth.
\_grading_scheme_log:x
31 \cs_new:Npn \_grading_scheme_log:n #1
32 {
33   \iow_term:x
34   {
35     [gs] \prg_replicate:nn \g\_grading_scheme_log_depth_int { . . }
36     \exp_not:n { #1 }
37   }
38   \iow_log:x
39   {
40     [gs] \prg_replicate:nn \g\_grading_scheme_log_depth_int { . . }
41     \exp_not:n { #1 }
42   }
43 }
44 \cs_generate_variant:Nn \_grading_scheme_log:n { x }
(End definition for \_grading_scheme_log:n.)
45 \</log>

```

4.3 Wrappers

We provide some L^AT_EX3 wrappers for private usage.

```

\_grading_scheme_multicolumn:nnn Just a wrapper around \multicolumn from LATEX 2ε.
46 \cs_set_eq:NN \_grading_scheme_multicolumn:nnn \multicolumn
(End definition for \_grading_scheme_multicolumn:nnn.)

```

```

\_grading_scheme_multirow:nnn Just a wrapper around \multirow from LATEX 2ε.
47 \cs_set_eq:NN \_grading_scheme_multirow:nnn \multirow
(End definition for \_grading_scheme_multirow:nnn.)

```

```

\_grading_scheme_cline:n A wrapper around \cline.
\_grading_scheme_cline:nn The second form takes the beginning and ending column as separate arguments.
48 \cs_set_eq:NN \_grading_scheme_cline:n \cline
49 \cs_new:Npn \_grading_scheme_cline:nn #1 #2
50 {
51   \_grading_scheme_cline:n { #1 - #2 }
52 }

```

(End definition for `_grading_scheme_cline:n` and `_grading_scheme_cline:nn`.)

```

\_grading_scheme_hline: Wrapper for a \hline
53 \cs_set_eq:NN \_grading_scheme_hline: \hline
(End definition for \_grading_scheme_hline:.)

```

```

\_grading_scheme_rotatebox:nnn Wrapper around rotatebox
54 \cs_new:Npn \_grading_scheme_rotatebox:nnn #1 %implicit #2, #3
55 {
56   \rotatebox [ #1 ]
57 }
(End definition for \_grading_scheme_rotatebox:nnn.)

```

4.4 Customizable backends

We set up some functions that will be used internally, but could technically be customized at some point. For now, these are just constants.

`__grading_scheme_points:n` Prints the number of points. Used in the last column of the grading scheme.

```
58 \cs_new:Npn \__grading_scheme_points:n #1
59 {
60   \textbf{ #1 ~ P. }
61 }
```

(End definition for `__grading_scheme_points:n`.)

`__grading_scheme_operation:n` Prints an operation. Used in the leftmost column of a block.

```
62 \cs_new:Npn \__grading_scheme_operation:n #1
63 {
64   { \sc #1 }
65 }
```

(End definition for `__grading_scheme_operation:n`.)

4.5 Resources

4.5.1 Some general thoughts

This is some general reasoning about why we model data structures as we do in this package, comparing parts of L^AT_EX3 to plain C code.

We have to model elements, blocks and entries here. L^AT_EX3 provides some data structures for this, and we will use property-lists to model our data structures for this package.

When thinking about what a property list is, a call to `\prop_new:N` essentially allocates some memory for us and makes our given argument a *pointer* to a data structure that we refer to as a “property list”.

We want to think of these as pointers since modifying a property list at some place really also modifies this property list at another, so these behave like pointers.

Considering the grouping mechanism of T_EX, this is of course not quite right for local variables (we assume that in the L^AT_EX3 spirit, all variables are either local or global once and for all, to avoid confusion) since these rather correspond to one pointer at each grouping level, where upon entering such a grouping level, the (new) pointer is initialized with a copy of the outer instance of the pointer and destroyed again when leaving the current grouping level.

In this spirit, we really want to think of grouping levels as *scopes* like in C.

Considering functions now, a typical L^AT_EX3 function has no return value (these would be functions with the `_p` predicate or `_use:` functions that leave contents in the input stream), since this is in general not desired. Rather, we pass pointers as a function argument and receive the desired “return value” as written into our given pointer (overwriting any prior data), which is also a common behaviour in large parts of C.

Now, as mentioned earlier, data structures such as property queues are just pointers that refer to some tokens with specific internal structure. All functions dealing with such pointers have to ensure that these invariants remain satisfied, and this is what functions like `\prop_get:NnN` or `\prop_put:Nn` do. The key point here is that we refer to these structures only with pointers and do not deal with them directly. This is what gives us

some very nice abstract model, hiding the class invariants of a property queue from the user and providing high-level semantic access to such data structures.

Just for completeness, we mention that this is not the case for *all* data structures. For example, a `clist` has some structure that is well-known and can be dealt with by the user directly. That is why there are also `clist` commands accepting `n`-type arguments for dealing with a `clist`, but no such commands for property-lists, as they only use `N` and thus essentially pointers.

If we want to model data structures now, especially ours from this package, we even want data structures containing other data structures. Although technically, storing all data of a grading scheme in a single macro would be possible, due to its recursive structure, parsing would be quite a challenge and possibly also rather slow. Also, this would require *direct* access to the representation of blocks contained in this block, which is undesired.

Compare this to the question of putting a property-list inside a property-list. Since there is no (semantic/provided) possibility of actually putting a property-list anywhere, we cannot store property-lists in property-lists directly. We can, however, store the name of a property list inside of another one, this amounts to actually storing the *pointer* to the first list inside the second one.

Now, back to our package, we essentially want to model blocks and alike as C-style `structs` that contain *pointers* to their elements and similar. The classic question of ownership arises in this context, so we would have to consider shallow or deep copies and a user has to watch out when dealing with our data structures.

Since only a limited set of operations is needed in our case, we take a simple approach by saying that each struct owns all its contents, that is, owns everything its data member pointers point to, also recursively. This essentially forbids shallow-copying (violating our assumption) but we do not need this here, so this is okay.

Still, this has some undesired consequences: Since in \LaTeX 3 each pointer has some name, we need to construct a lot of pointers on the fly where we are not able to actually specify their names directly, so we have to indirectly deal with pointers. That means that we actually need double-pointers, i.e. local variables *containing* pointers to the data structures we are actually interested in. This is what a `void**` would correspond to in C. Also, note that once we store a pointer to some element in a property queue (e.g. the name of the list of members when dealing with a block), we actually also need a `void**` when retrieving this value again from the property queue, since we return by pointer.

Consequently, this leads to quite a lot of expansion-control when dealing with such pointers, and if not taking precautions, also to easy confusion about what is a pointer and what not.

Our approach to tackle these problems is described in [subsection 4.6](#).

4.6 Pointers

This module should/will be reworked into an own package in the future. Also, there is no proper documentation for it apart from the comments in the implementation, since this is not meant to be used in other code than this package currently, nor regarded stable.

We introduce a new data type of `pointer` and a new argument type of `P`. The argument type `P` shall indicate a single expansion *with the assumption that this expansion will yield a single token*. So, from an implementation point of view, this is the same as an `o`-type argument, just with further restricted assumptions.

This also enables us to generate P variant arguments from N variant macros. Compare this to the usual deprecation warning that L^AT_EX3 gives when generating an o variant from an N-type argument:

```
! LaTeX3 Error: Variant form 'o' deprecated for base form '\foo:N'. One
(LaTeX3)          should not change an argument from type 'N' to type 'o':
(LaTeX3)          base form only accepts a single token argument.
```

Since basically anything in L^AT_EX3 is in fact a pointer, we will introduce the `pointer` structure with the `ptr` type (suffix) and actually mean that this is a `void**`.

Thus, we introduce the following conventions:

1. A T_EX-pointer is a T_EXcontrol sequence that represents some data.
2. A `pointer` is a control sequence that expands to a single T_EX control sequence that is a T_EX-pointer.
3. The *name* of a `pointer` is the name of the control sequence representing it. Thus, such a name always ends with `_ptr` by convention.
4. A `ptr` is said to be NULL if its expansion yields an undefined T_EX control sequence. We also refer to this as a `nullptr`.
5. A *typed pointer* is a pointer where the underlying type is specified. By this, we mean that expanding the token exactly once yields a token representing a data structure of the type we said this pointer to have or that the pointer is NULL.
6. A *void pointer* shall be a pointer whose actual type is unknown.
7. When a `ptr` is of type `type`, we denote these by the suffix `_type_ptr`.
8. *Dereferencing* a pointer amounts to expanding it exactly once.
9. A P-type argument accepts a single token that has to be of type `ptr`. The token is expanded exactly once and then treated as an N-type argument in the corresponding function.

`\g_ptr_counter_int` Counts the number of pointers given by `\ptr_new:N`. This ensures that each pointer can have a unique name regardless of grouping.

```
66 \int_new:N \g_ptr_counter_int
```

(End definition for `\g_ptr_counter_int`.)

`\l_ptr_var_prefix_str` Stores the prefix of new pointer variables.

```
67 \str_new:N \l_ptr_var_prefix_str
```

(End definition for `\l_ptr_var_prefix_str`.)

`\ptr_new:N` `\ptr_new:N⟨pointer⟩`

Gives a new unique pointer that is NULL. Gives an error when the `⟨pointer⟩` already exists. The pointer variable is created globally.

```
68 \cs_new:Npn \ptr_new:N #1
```

```
69 {
```

```
70   \__ptr_get_var_prefix:NN #1 \l_ptr_var_prefix_str
```

```

71     \ptr_new:NVn #1 \l_ptr_var_prefix_str { any }
72 }
73 \cs_new:Npn \ptr_new:Nn #1 #2
74 {
75     \ptr_new:Nnn { #1 } { #2 } { any }
76 }
77 % ptr var, prefix, type
78 \cs_new:Npn \ptr_new:Nnn #1 #2 %i #3
79 {
80     \str_case:nnT
81     { #2 }
82     {
83         { l } { }
84         { g } { }
85     }
86     {
87         \tl_new:N #1
88     }
89     \__ptr_init:Nnn #1 { #2 } %i { #3 }
90 }
91 \cs_generate_variant:Nn \ptr_new:Nnn { N V n }
92 % ptr var, prefix, type
93 % assumes that the pointer name exists already in case of l or g prefix
94 \cs_new:Npn \__ptr_init:Nnn #1 #2 % implicit #3
95 {
96     \__ptr_get_var_prefix:NN #1 \l_ptr_var_prefix_str
97     \str_case:VnF
98     \l_ptr_var_prefix_str
99     {
100         { l }
101         {
102             \cs_set_eq:NN \__ptr_poly_tl_set:Nx \tl_set:Nx
103         }
104         { g }
105         {
106             \cs_set_eq:NN \__ptr_poly_tl_set:Nx \tl_gset:Nx
107         }
108         { c }
109         {
110             \cs_set_eq:NN \__ptr_poly_tl_set:Nx \tl_const:Nx
111         }
112     }
113     {
114         \str_show:N \l_ptr_var_prefix_str % TODO: show error
115     }
116     \__ptr_init_aux:Nnn #1 { #2 }
117 }
118 \cs_generate_variant:Nn \__ptr_init:Nnn { N V n }
119 % ptr var, prefix, type. assumes poly_tl_set:Nx has been set
120 \cs_new:Npn \__ptr_init_aux:Nnn #1 #2 #3
121 {
122     \__ptr_poly_tl_set:Nx #1
123     {
124         \exp_not:c

```

```

125         {
126             #2
127             __ptr_unique__
128             \int_use:N \g_ptr_counter_int
129             -
130             #3
131         }
132     }
133     \int_gincr:N \g_ptr_counter_int
134 }
135 \cs_generate_variant:Nn \__ptr_init:Nnn { N V n }

```

(End definition for \ptr_new:N. This function is documented on page ??.)

`__ptr_get_var_prefix:NN` Gets the first character of the name of the variable given as #1. The first character is assumed to be one of c, l or g by usual naming conventions.

```

136 \cs_new:Npn \__ptr_get_var_prefix:NN #1 #2
137 {
138     \tl_if_head_eq_charcode:fNTF { \cs_to_str:N #1 } l
139     {
140         \str_set:Nn #2 { l }
141     }
142     {
143         \tl_if_head_eq_charcode:fNTF { \cs_to_str:N #1 } g
144         {
145             \str_set:Nn #2 { g }
146         }
147         {
148             \tl_if_head_eq_charcode:fNTF { \cs_to_str:N #1 } c
149             {
150                 \str_set:Nn #2 { c }
151             }
152             {
153                 \msg_error:nxx { ptr } { variable-name-illegal-prefix }
154                 {
155                     \token_to_str:N #1
156                 }
157             }
158         }
159     }
160 }
161 \msg_new:nnnn { ptr } { variable-name-illegal-prefix }
162 {
163     Requested ~ new ~ pointer ~ '#1' ~ has ~ illegal ~ prefix.
164 }
165 {
166     Prefix ~ should ~ be ~ one ~ of ~ 'l', ~ 'g' ~ or ~ 'c'.
167 }

```

(End definition for __ptr_get_var_prefix:NN.)

`\ptr_clear:N` Clears this pointer. This makes the pointer equivalent to a new nullptr.

```

168 \cs_new:Npn \ptr_clear:N #1
169 {

```

```

170     \__ptr_get_var_prefix:NN #1 \l__ptr_var_prefix_str
171     \__ptr_init:NVn #1 \l__ptr_var_prefix_str { any }
172   }
173   \cs_new:Npn \ptr_clear:Nn #1 #2
174   {
175     \__ptr_init:Nnn #1 { #2 } { any }
176   }

```

(End definition for `\ptr_clear:N`. This function is documented on page ??.)

`\ptr_set:NN` Sets the pointer to represent the given argument.

```

177   \cs_set_eq:NN \ptr_set:NN \tl_set:Nn

```

(End definition for `\ptr_set:NN`. This function is documented on page ??.)

`\ptr_set_eq:NN` Accepts two pointers. The first is set to be equivalent to the second.

```

178   \cs_set_eq:NN \ptr_set_eq:NN \tl_set_eq:NN

```

(End definition for `\ptr_set_eq:NN`. This function is documented on page ??.)

`\ptr_use:N` Uses the pointer, i.e. expands to the stored `TeX`-pointer.

```

179   \cs_set_eq:NN \ptr_use:N \tl_use:N

```

(End definition for `\ptr_use:N`. This function is documented on page ??.)

`__ptr_check:N` Checks that the given argument is a pointer.

```

180   \cs_new:Npn \__ptr_check:N #1
181   {
182     \tl_if_single:NF #1
183     {
184       \msg_error:nnx { ptr } { is-not-a-pointer }
185       {
186         \token_to_str:N #1
187       }
188     }
189   }
190   \msg_new:nnn { ptr } { is-not-a-pointer }
191   {
192     The ~ control ~ sequence ~ '#1' ~ is ~ not ~ a ~ valid ~ pointer.
193   }

```

(End definition for `__ptr_check:N`.)

`__ptr_check:NN` Checks that the second argument is a pointer.

```

194   \cs_new:Npn \__ptr_check:NN #1 #2
195   {
196     \__ptr_check:N #2
197     #1 #2
198   }

```

(End definition for `__ptr_check:NN`.)

`__ptr_check:nN` Checks that the second argument is a pointer.

```

199 \cs_new:Npn \__ptr_check:nN #1 #2
200 {
201   \__ptr_check:N #2
202   { #1 } #2
203 }

```

(End definition for __ptr_check:nN.)

`\l__ptr_content_tl` Stores the (internal) T_EX-pointer of a pointer.

```

204 \tl_new:N \l__ptr_content_tl

```

(End definition for \l__ptr_content_tl.)

`\exp_args:NP` Expands a pointer. This leaves the first tokens as a single token in the input stream,
`\exp_args:NNP` followed by the value of the pointer as a single token.

`\exp_args:NNNP` If the last argument does not expand to a single token, an error is given.

```

\exp_args:NNNNP
\exp_args:NPP
205 \cs_new:Npn \exp_args:NP #1 #2
206 {
207   \__ptr_check:N #2
208   \exp_last_unbraced:No #1 #2
209 }
210 \cs_new:Npn \exp_args:NNP #1 #2 #3
211 {
212   \__ptr_check:N #3
213   \exp_last_unbraced:NNo #1 #2 #3
214 }
215 \cs_new:Npn \exp_args:NNNP #1 #2 #3 #4
216 {
217   \__ptr_check:N #4
218   \exp_last_unbraced:NNNo #1 #2 #3 #4
219 }
220 \cs_new:Npn \exp_args:NNNNP #1 #2 #3 #4 #5
221 {
222   \__ptr_check:N #5
223   \exp_last_unbraced:NNNNo #1 #2 #3 #4 #5
224 }
225 \cs_new:Npn \exp_args:NPP #1 #2 #3
226 {
227   \__ptr_check:N #2
228   \__ptr_check:N #3
229   \exp_last_two_unbraced:Noo #1 #2 #3
230 }

```

(End definition for \exp_args:NP and others. These functions are documented on page ??.)

`\ptr_if_null:N \underline{T} \underline{F}` Checks if the pointer is NULL.

```

231 \cs_new:Npn \ptr_if_null:NT
232 {
233   \exp_args:NP \cs_if_exist:NF
234 }
235 \cs_new:Npn \ptr_if_null:NF
236 {
237   \exp_args:NP \cs_if_exist:NT

```

```

238 }
239 \cs_new:Npn \ptr_if_null:NTF #1 #2 #3
240 {
241   \exp_args:NP \cs_if_exist:NTF #1 { #3 } { #2 }
242 }

```

(End definition for \ptr_if_null:NTF. This function is documented on page ??.)

```

\tl_set_eq:NP Just variants of \tl_set_eq:NN with indicated signature.
\tl_set_eq:PN
\tl_set_eq:PP
  \tl_use:P
  \exp_not:P
  \clist_new:P
  \clist_put_right:Pn
  \clist_gput_right:Pn
\clist_map_function:PN
  \prop_show:P
  \clist_show:P
243 \cs_new:Npn \tl_set_eq:NP
244 {
245   \exp_args:NMP \tl_set_eq:NN
246 }
247 \cs_new:Npn \tl_set_eq:PN
248 {
249   \exp_args:NP \tl_set_eq:NN
250 }
251 \cs_new:Npn \tl_set_eq:PP
252 {
253   \exp_args:NPP \tl_set_eq:NN
254 }
255 \cs_new:Npn \tl_use:P
256 {
257   \exp_args:NP \tl_use:N
258 }
259 \cs_new:Npn \exp_not:P
260 {
261   \exp_args:NP \exp_not:N
262 }
263 \cs_new:Npn \clist_new:P
264 {
265   \exp_args:NP \clist_new:N
266 }
267 \cs_new:Npn \clist_show:P
268 {
269   \exp_args:NP \clist_show:N
270 }
271 \cs_new:Npn \clist_put_right:Pn
272 {
273   \exp_args:NP \clist_put_right:Nn
274 }
275 \cs_new:Npn \clist_gput_right:Pn
276 {
277   \exp_args:NP \clist_gput_right:Nn
278 }
279 \cs_new:Npn \clist_map_function:PN
280 {
281   \exp_args:NP \clist_map_function:NN
282 }
283 \cs_new:Npn \prop_show:P
284 {
285   \exp_args:NP \prop_show:N
286 }

```

(End definition for \tl_set_eq:NP and others. These functions are documented on page ??.)

`\ptr_show:N` Shows information about this pointer, namely:

If it is `NULL`, then it indicates this. If it is not `NULL`, then the `TEX`-pointer and the contained contents of the `TEX`-pointer (in unexpanded form) are shown.

```

287 \cs_new:Npn \ptr_show:N #1
288 {
289   \__ptr_check:N #1
290   \ptr_if_null:NTF #1
291   {
292     \tl_show:x
293     {
294       \token_to_str:N #1 -> \exp_not:P #1 = NULL
295     }
296   }
297   {
298     \tl_show:x
299     {
300       \token_to_str:N #1 -> \exp_not:P #1 -> \exp_args:NP \exp_not:V #1
301     }
302   }
303 }
```

(End definition for `\ptr_show:N`. This function is documented on page ??.)

4.7 Structs

We will model C-style `structs` as property-lists in this package. Each struct member is put into the property list, using its 'name' as a key, and we store the corresponding contents there.

We will also have a local variable named correspondingly for each `struct` member. In case we deal with a struct and want to access its data, we load the values from the property-list into these local variables. Thus, the full information about a `struct` is contained in the property-list, and we can work with them quite conveniently when implementing functions.

4.8 Entries

An entry is for us the following:

```

struct Entry {
    tl description;
    int points;
}
```

`\grading_scheme_entry_new:N` Do what their signature suggests. We just forward them to the property lists.

```

\grading_scheme_entry_clear:N
\grading_scheme_entry_set_eq:NN
\grading_scheme_entry_gclear:N
\grading_scheme_entry_gset_eq:NN
304 \cs_set_eq:NN \grading_scheme_entry_new:N \prop_new:N
305 \cs_set_eq:NN \grading_scheme_entry_clear:N \prop_clear:N
306 \cs_set_eq:NN \grading_scheme_entry_set_eq:NN \prop_set_eq:NN
307 \cs_set_eq:NN \grading_scheme_entry_gclear:N \prop_gclear:N
308 \cs_set_eq:NN \grading_scheme_entry_gset_eq:NN \prop_gset_eq:NN
```

(End definition for `\grading_scheme_entry_new:N` and others. These functions are documented on page ??.)

$\backslash l_grading_scheme_entry_points_int$ $\backslash l_grading_scheme_entry_description_tl$	<p>The mentioned local variables where we retrieve values.</p> <pre> 309 \int_new:N \l__grading_scheme_entry_points_int 310 \tl_new:N \l__grading_scheme_entry_description_tl (End definition for \l__grading_scheme_entry_points_int and \l__grading_scheme_entry_description_ tl.) </pre>
$\backslash grading_scheme_entry_set_description:Nn$ $\backslash grading_scheme_entry_gset_description:Nn$	<p>Sets the description.</p> <pre> 311 \cs_new:Npn \grading_scheme_entry_set_description:Nn #1 % implicit description 312 { 313 \prop_put:Nnn #1 { description } 314 } 315 \cs_new:Npn \grading_scheme_entry_gset_description:Nn #1 % implicit description 316 { 317 \prop_gput:Nnn #1 { description } 318 } (End definition for \grading_scheme_entry_set_description:Nn and \grading_scheme_entry_gset_ description:Nn. These functions are documented on page ??.) </pre>
$\backslash grading_scheme_entry_set_points:Nn$ $\backslash grading_scheme_entry_gset_points:Nn$	<p>Sets the points.</p> <pre> 319 \cs_new:Npn \grading_scheme_entry_set_points:Nn #1 #2 320 { 321 \prop_put:Nnx #1 { points } { \int_eval:n { #2 } } 322 } 323 \cs_new:Npn \grading_scheme_entry_gset_points:Nn #1 #2 324 { 325 \prop_gput:Nnn #1 { points } { \int_eval:n { #2 } } 326 } (End definition for \grading_scheme_entry_set_points:Nn and \grading_scheme_entry_gset_points:Nn. These functions are documented on page ??.) </pre>
$\backslash grading_scheme_entry_get_description:NN$	<p>Gets the description of the entry and stores it in #2. The assignment is local.</p> <pre> 327 \cs_new:Npn \grading_scheme_entry_get_description:NN #1 %implicit #2 328 { 329 \prop_get:NnN #1 { description } 330 } (End definition for \grading_scheme_entry_get_description:NN. This function is documented on page ??.) </pre>
$\backslash grading_scheme_entry_get_points:NN$	<p>Gets the points of the entry and stores it in #2. The assignment is local.</p> <pre> 331 \cs_new:Npn \grading_scheme_entry_get_points:NN #1 %implicit #2 332 { 333 \prop_get:NnN #1 { points } 334 } (End definition for \grading_scheme_entry_get_points:NN. This function is documented on page ??.) </pre>
$\backslash_grading_scheme_entry_load_points:N$	<p>Loads the points into the local variable</p> <pre> 335 \cs_new:Npn __grading_scheme_entry_load_points:N #1 336 { 337 \grading_scheme_entry_get_points:NN #1 \l__grading_scheme_entry_points_int 338 } </pre>

(End definition for `_grading_scheme_entry_load_points:N`.)

`_grading_scheme_entry_load_description:N`

Loads the description of an entry

```

339 \cs_new:Npn \_grading_scheme_entry_load_description:N #1
340 {
341   \grading_scheme_entry_get_description:NN #1 \l__grading_scheme_entry_description_tl
342 }

```

(End definition for `_grading_scheme_entry_load_description:N`.)

`\grading_scheme_entry_format:NnnN`

`\grading_scheme_entry_put:NnnN{<entry>}{<indent>}{<width>}{<tl var>}`

`\grading_scheme_entry_gformat:NnnN`

Puts the formatted contents of the entry into the `<tl var>`. The `<indent>` specify how many tabs will be inserted before adding the contents. The `<width>` specifies how many columns this entry will occupy. This `<width>` has to be at least 2.

`\grading_scheme_entry_format:PnnN`

An `\` is inserted at the end of the entry.

`\grading_scheme_entry_gformat:PnnN`

```

343 \cs_new:Npn \grading_scheme_entry_format:NnnN
344 {
345   \cs_set_eq:NN \_grading_scheme_tl_put_right:Nx \tl_put_right:Nx
346   \_grading_scheme_entry_format:NnnN
347 }
348 \cs_new:Npn \grading_scheme_entry_gformat:NnnN
349 {
350   \cs_set_eq:NN \_grading_scheme_tl_put_right:Nx \tl_gput_right:Nx
351   \_grading_scheme_entry_format:NnnN
352 }
353 \cs_new:Npn \grading_scheme_entry_format:PnnN
354 {
355   \exp_args:NP \grading_scheme_entry_format:NnnN
356 }
357 \cs_new:Npn \grading_scheme_entry_gformat:PnnN
358 {
359   \exp_args:NP \grading_scheme_entry_gformat:NnnN
360 }

```

(End definition for `\grading_scheme_entry_format:NnnN` and others. These functions are documented on page ??.)

`_grading_scheme_entry_format:NnnN`

Aux function that assumes that `_grading_scheme_tl_put_right:Nx` has been so to globally or locally putting into the token list.

`_grading_scheme_entry_format:PnnN`

```

361 \cs_new:Npn \_grading_scheme_entry_format:NnnN #1 #2 #3 #4
362 {
363   <*log>
364   \_grading_scheme_log:x
365   { Formatting ~ entry ~ '\token_to_str:N #1' ~ into ~ '\token_to_str:N #4' with }
366   \_grading_scheme_log:n { indent = '#2' ~ and ~ width ~ '#3' }
367   \prop_log:N #1
368   \_grading_scheme_log_incr:
369   </log>
370   \_grading_scheme_entry_assert_description:N #1 % implicitly loads value
371   \_grading_scheme_entry_assert_points:N #1 % implicitly loads value
372   \_grading_scheme_tl_put_right:Nx #4
373   {
374     \prg_replicate:nn { #2 } { & }

```

```

375 \exp_not:N \__grading_scheme_multicolumn:nnn
376 {
377   \int_eval:n { #3 -1 }
378 }
379 { 1 }
380 {
381   \exp_not:V \l__grading_scheme_entry_description_tl
382 }
383 &
384 \exp_not:N \__grading_scheme_points:n
385 {
386   \tl_use:N \l__grading_scheme_entry_points_int
387 }
388 \\
389 }
390 <*log>
391   \__grading_scheme_log_decr:
392   \__grading_scheme_log:x { / Formatting ~ entry ~ '\token_to_str:N #1' }
393 </log>
394 }
395 \cs_new:Npn \__grading_scheme_entry_format:PnnN % implicit #1-4
396 {
397   \exp_args:NP \__grading_scheme_entry_format:NnnN
398 }

```

(End definition for __grading_scheme_entry_format:NnnN and __grading_scheme_entry_format:PnnN.)

__grading_scheme_entry_assert_description:N
 __grading_scheme_entry_assert_points:N

These functions check the presence of values of an entry. If an entry is absent, an error message is omitted.

```

399 \cs_new:Npn \__grading_scheme_entry_assert_description:N #1
400 {
401   \__grading_scheme_entry_load_description:N #1
402   \quark_if_no_value:NT \l__grading_scheme_entry_description_tl
403   {
404     \msg_error:nnxxx { grading-scheme } { missing-value }
405     { entry } { \token_to_str:N #1 } { description }
406   }
407 }
408 \cs_new:Npn \__grading_scheme_entry_assert_points:N #1
409 {
410   \__grading_scheme_entry_load_points:N #1
411   \quark_if_no_value:NT \l__grading_scheme_entry_points_int
412   {
413     \msg_error:nnxxx { grading-scheme } { missing-value }
414     { entry } { \token_to_str:N #1 } { points }
415   }
416 }
417 \msg_new:nnn { grading-scheme } { missing-value }
418 {
419   #1 ~ '#2' ~ has ~ no ~ #3.
420 }

```

(End definition for __grading_scheme_entry_assert_description:N and __grading_scheme_entry_assert_points:N.)

4.9 Blocks

4.9.1 Struct setting / reading

A $\langle block \rangle$ is for us the following:

```
struct Block {
    clist* elements;
    tl      text;
    tl      operation;
}
```

Do what thear names suggest. We just forward these to the property lists.

```
\grading_scheme_block_new:N
    \grading_scheme_block_clear:N
    \grading_scheme_block_set_eq:NN
    \grading_scheme_block_gclear:N
    \grading_scheme_block_gset_eq:NN
421 \cs_set_eq:NN \grading_scheme_block_new:N      \prop_new:N
422 \cs_set_eq:NN \grading_scheme_block_clear:N    \prop_clear:N
423 \cs_set_eq:NN \grading_scheme_block_set_eq:NN  \prop_gset_eq:NN
424 \cs_set_eq:NN \grading_scheme_block_gclear:N   \prop_gclear:N
425 \cs_set_eq:NN \grading_scheme_block_gset_eq:NN \prop_gset_eq:NN
```

(End definition for $\backslash grading_scheme_block_new:N$ and others. These functions are documented on page ??.)

The mentioned local variables where we retrieve values.

```
\l__grading_scheme_block_elements_clist_ptr
\l__grading_scheme_block_description_tl
\l__grading_scheme_block_operation_tl
426 \ptr_new:N \l__grading_scheme_block_elements_clist_ptr
427 \tl_new:N \l__grading_scheme_block_description_tl
428 \tl_new:N \l__grading_scheme_block_operation_tl
```

(End definition for $\backslash l_grading_scheme_block_elements_clist_ptr$, $\backslash l_grading_scheme_block_description_tl$, and $\backslash l_grading_scheme_block_operation_tl$.)

Set the description / operation or elements of the block.

When setting elements, a $\langle clist var \rangle$ is expected.

```
\grading_scheme_block_set_description:Nn
\grading_scheme_block_gset_description:Nn
\grading_scheme_block_set_operation_tl:Nn
\grading_scheme_block_gset_operation_tl:Nn
\grading_scheme_block_set_elements:NN
\grading_scheme_block_gset_elements:NN
\grading_scheme_block_set_elements:NP
\grading_scheme_block_gset_elements:NP
429 \cs_new:Npn \grading_scheme_block_set_description:Nn #1 % implicit description
430 {
431     \prop_put:Nnn #1 { description }
432 }
433 \cs_new:Npn \grading_scheme_block_gset_description:Nn #1 % implicit description
434 {
435     \prop_gput:Nnn #1 { description }
436 }
437 \cs_new:Npn \grading_scheme_block_set_operation:Nn #1 % implicit operation
438 {
439     \prop_put:Nnn #1 { operation }
440 }
441 \cs_new:Npn \grading_scheme_block_gset_operation:Nn #1 % implicit operation
442 {
443     \prop_gput:Nnn #1 { operation }
444 }
445 \cs_new:Npn \grading_scheme_block_set_elements:NN #1 % implicit elements clist
446 {
447     \prop_put:Nnn #1 { elements }
448 }
449 \cs_new:Npn \grading_scheme_block_gset_elements:NN #1 % implicit elements clist
450 {
```

```

451     \prop_gput:Nnn #1 { elements }
452   }
453   \cs_new:Npn \grading_scheme_block_set_elements:NP
454     {
455       \exp_args:NMP \grading_scheme_block_set_elements:NN
456     }
457   \cs_new:Npn \grading_scheme_block_gset_elements:NP
458     {
459       \exp_args:NMP \grading_scheme_block_gset_elements:NN
460     }

```

(End definition for \grading_scheme_block_set_description:Nn and others. These functions are documented on page ??.)

```

\grading_scheme_block_get_description:NN \grading_scheme_block_get_operation:NN \grading_scheme_block_get_elements:NN
\grading_scheme_block_get_operation:NN \grading_scheme_block_get_elements:NN
\grading_scheme_block_get_elements:NN

```

Get access to the members of the block. The assignment is local. The returned values might be \q_no_value if no value is present.

```

461   \cs_new:Npn \grading_scheme_block_get_description:NN #1 % implicit #2
462     {
463       \prop_get:NnN #1 { description }
464     }
465   \cs_new:Npn \grading_scheme_block_get_operation:NN #1 % implicit #2
466     {
467       \prop_get:NnN #1 { operation }
468     }
469   \cs_new:Npn \grading_scheme_block_get_elements:NN #1 % implicit #2
470     {
471       \prop_get:NnN #1 { elements }
472     }

```

(End definition for \grading_scheme_block_get_description:NN, \grading_scheme_block_get_operation:NN, and \grading_scheme_block_get_elements:NN. These functions are documented on page ??.)

_grading_scheme_block_load_description:N Loads the members into the corresponding local variables.

```

\_grading_scheme_block_load_operation:N
\_grading_scheme_block_load_elements:NN
473   \cs_new:Npn \_grading_scheme_block_load_description:N #1
474     {
475       \grading_scheme_block_get_description:NN #1 \l__grading_scheme_block_description_tl
476     }
477   \cs_new:Npn \_grading_scheme_block_load_operation:N #1
478     {
479       \grading_scheme_block_get_operation:NN #1 \l__grading_scheme_block_operation_tl
480     }
481   \cs_new:Npn \_grading_scheme_block_load_elements:N #1
482     {
483       \grading_scheme_block_get_elements:NN #1 \l__grading_scheme_block_elements_clist_ptr
484     }

```

(End definition for _grading_scheme_block_load_description:N, _grading_scheme_block_load_operation:N, and _grading_scheme_block_load_elements:NN.)

_grading_scheme_block_ensure_elements:N Ensures that this block has an elements attribute that is a valid clist pointer.

If no value is present, or the pointer is NULL, a pointer and/or the clist variable are created.

```

485 \cs_new:Npn \__grading_scheme_block_ensure_elements:N % implicit #1
486 {
487   \cs_set_eq:NN
488     \__grading_scheme_block_set_elements_optg:NP
489     \grading_scheme_block_set_elements:NP
490     \__grading_scheme_block_ensure_elements_aux:nN { 1 }
491 }
492 \cs_new:Npn \__grading_scheme_block_gensure_elements:N % implicit #1
493 {
494   \cs_set_eq:NN
495     \__grading_scheme_block_set_elements_optg:NP
496     \grading_scheme_block_gset_elements:NP
497     \__grading_scheme_block_ensure_elements_aux:nN { g } %i #1
498 }
499 % prefix, block
500 % assumes that \__grading_scheme_block_set_elements_optg:NP has been
501 \cs_new:Npn \__grading_scheme_block_ensure_elements_aux:nN #1 #2
502 {
503   \__grading_scheme_block_load_elements:N #2
504   \quark_if_no_value:NT \l__grading_scheme_block_elements_clist_ptr
505   {
506     \ptr_clear:Nn \l__grading_scheme_block_elements_clist_ptr { #1 }
507     \__grading_scheme_block_set_elements_optg:NP
508     #2
509     \l__grading_scheme_block_elements_clist_ptr
510   }
511   \ptr_if_null:NT \l__grading_scheme_block_elements_clist_ptr
512   {
513     \clist_new:P \l__grading_scheme_block_elements_clist_ptr
514   }
515 }

```

(End definition for __grading_scheme_block_ensure_elements:N.)

```

\grading_scheme_block_add_element:NN      \grading_scheme_block_add_element:NN⟨block var⟩⟨element var⟩
\grading_scheme_block_gadd_element:NN     Add an element to the block.
\grading_scheme_block_add_element:PP
\grading_scheme_block_gadd_element:PP
\grading_scheme_block_add_element:PN
\grading_scheme_block_gadd_element:PN
516 \cs_new:Npn \grading_scheme_block_add_element:NN #1 % implicit element
517 {
518   \__grading_scheme_block_ensure_elements:N #1 % also loads local variable
519   \clist_put_right:Pn \l__grading_scheme_block_elements_clist_ptr
520 }
521 \cs_new:Npn \grading_scheme_block_gadd_element:NN #1 % implicit element
522 {
523   \__grading_scheme_block_gensure_elements:N #1 % also loads local variable
524   \clist_gput_right:Pn \l__grading_scheme_block_elements_clist_ptr
525 }
526 \cs_new:Npn \grading_scheme_block_add_element:PP
527 {
528   \exp_args:NPP \grading_scheme_block_add_element:NN
529 }
530 \cs_new:Npn \grading_scheme_block_gadd_element:PP
531 {
532   \exp_args:NPP \grading_scheme_block_gadd_element:NN
533 }

```

```

534 \cs_new:Npn \grading_scheme_block_add_element:PN
535 {
536   \exp_args:NP \grading_scheme_block_add_element:NN
537 }
538 \cs_new:Npn \grading_scheme_block_gadd_element:PN
539 {
540   \exp_args:NP \grading_scheme_block_gadd_element:NN
541 }

```

(End definition for `\grading_scheme_block_add_element:NN` and others. These functions are documented on page ??.)

4.9.2 Formatting blocks

`\grading_scheme_block_format:NnnnN` `\grading_scheme_block_format:NnnnN⟨block var⟩{⟨indent⟩}{⟨width⟩}{⟨indent first row⟩}{⟨tl var⟩}`
`\grading_scheme_block_gformat:NnnnN` `\grading_scheme_block_gformat:NnnnN`
`\grading_scheme_block_format:PnnnN` `\grading_scheme_block_format:PnnnN`
`\grading_scheme_block_gformat:PnnnN` `\grading_scheme_block_gformat:PnnnN`

Formats this block and puts the control sequence into `⟨tl var⟩`.
The `⟨indent⟩` and `⟨width⟩` work as in `\grading_scheme_entry_format:NnnN`.
`⟨indent first row⟩` can be any `⟨boolean expression⟩` and indicates if the first row is also indented. Set this to false to start the block in a row of the tabular that already has contents.

```

542 \cs_new:Npn \grading_scheme_block_format:NnnnN %implicit #1-5
543 {
544   \cs_set_eq:NN \__grading_scheme_tl_put_right:Nx \tl_put_right:Nx
545   \cs_set_eq:NN \__grading_scheme_element_format_optg:NnnnN \grading_scheme_element_format
546   \cs_set_eq:NN \__grading_scheme_tl_set:Nn \tl_set:Nn
547   \__grading_scheme_block_format:NnnnN
548 }
549 \cs_new:Npn \grading_scheme_block_gformat:NnnnN %implicit #1-5
550 {
551   \cs_set_eq:NN \__grading_scheme_tl_put_right:Nx \tl_gput_right:Nx
552   \cs_set_eq:NN \__grading_scheme_element_format_optg:NnnnN \grading_scheme_element_gformat
553   \cs_set_eq:NN \__grading_scheme_tl_set:Nn \tl_gset:Nn
554   \__grading_scheme_block_format:NnnnN
555 }
556 \cs_new:Npn \grading_scheme_block_format:PnnnN % implicit #1-5
557 {
558   \exp_args:NP
559   \grading_scheme_block_format:NnnnN
560 }
561 \cs_new:Npn \grading_scheme_block_gformat:PnnnN % implicit #1-5
562 {
563   \exp_args:NP
564   \grading_scheme_block_gformat:NnnnN
565 }

```

(End definition for `\grading_scheme_block_format:NnnnN` and others. These functions are documented on page ??.)

For formatting the block, we need some local variables:

`\l_grading_scheme_block_indent_bool` Controls whether the `__grading_scheme_block_indent:nN` macro will actually perform an indent.

```

566 \bool_new:N \l_grading_scheme_block_indent_bool

```

(End definition for \l__grading_scheme_block_indent_bool.)

__grading_scheme_block_height_int Locally stores the height of the block to be typeset.

```
567 \int_new:N \l__grading_scheme_block_height_int
```

(End definition for __grading_scheme_block_height_int.)

__grading_scheme_block_format:NnnnN Aux function. Assumes that __grading_scheme__tl_put_right:Nx has been set properly.
__grading_scheme_block_format:PnnnN

```
568 \cs_new:Npn \__grading_scheme_block_format:NnnnN #1 #2 #3 #4 #5
569 {
570   \*log
571   \__grading_scheme_log:x
572   {
573     Formatting ~ block ~ '\token_to_str:N #1' ~ into ~ '\token_to_str:N #5'
574     ~ with ~ indent = '#2', ~ width ~ '#3' ~ and ~ first ~
575     row = '\bool_to_str:n { #4 }'
576   }
577   \prop_log:N #1
578   \__grading_scheme_block_load_elements:N #1
579   \exp_args:NP \clist_log:N \l__grading_scheme_block_elements_clist_ptr
580   \__grading_scheme_log_incr:
581   \*log
```

We need grouping here so that our indentation boolean is not messed up by recursive calls:

```
582   \bool_set:Nn \l__grading_scheme_block_indent_bool { #4 && \c_true_bool }
583   \group_begin:
584   \__grading_scheme_block_load_description:N #1
585   \grading_scheme_block_get_height:NN #1 \l__grading_scheme_block_height_int
```

We now format the description of the block if a value is present.

```
586   \quark_if_no_value:NF \l__grading_scheme_block_description_tl
587   {
588     \__grading_scheme_block_indent:nN { #2 } #5
589     \__grading_scheme__tl_put_right:Nx #5
590     {
591       \exp_not:N \__grading_scheme_multicolumn:nnn
592       {
593         \int_eval:n { #3 }
594       }
595       { |l| }
596       {
597         \exp_not:N \l__grading_scheme_block_description_tl
598       }
599       \\
600       \exp_not:N \__grading_scheme_cline:nn
601       {
602         \int_eval:n { #2 + 2 }
603       }
604       {
605         \int_eval:n { #2 + #3 }
606       }
607     }
608   }
```

Now, we have to format the operation of this block. This is a multirow with rotated description

```

609 \__grading_scheme_block_indent:nN { #2 } #5
610 \__grading_scheme_block_assert_operation:N #1
611 \__grading_scheme_tl_put_right:Nx #5
612 {
613   \exp_not:N \__grading_scheme_multirow:nnn
614   {
615     \int_eval:n { \int_use:N \l__grading_scheme_block_height_int - 1 }
616   }
617   { * }
618   {
619     \exp_not:N \__grading_scheme_rotatebox:nnn
620     { origin = c }
621     { 90 }
622     {
623       \exp_not:N \__grading_scheme_operation:n
624       {
625         \exp_not:N \l__grading_scheme_block_operation_tl
626       }
627     }
628   }
629   &
630 }

```

The first element of our block must not be indented:

```

631 \bool_set_false:N \l__grading_scheme_block_indent_bool

```

Now, we want to recursively typeset all elements of this block. We need a customized function for this to map over the elements.

```

632 \cs_set:Npn \__grading_scheme_block_format_element:N ##1
633 {
634   \__grading_scheme_element_format_optg:NnnnN
635   ##1
636   { #2 + 1 } % indent is one more than the current block
637   { #3 - 1 } % width is one less than current block
638   { \l__grading_scheme_block_indent_bool }
639   #5

```

This ensures that the second and all further elements will get indented:

```

640 \bool_set_true:N \l__grading_scheme_block_indent_bool
641 }
642 \__grading_scheme_block_ensure_elements:N #1 % load + sanitize elements
643 \clist_map_function:PN
644 \l__grading_scheme_block_elements_clist_ptr
645 \__grading_scheme_block_format_element:N

```

Now, we need to 'smuggle out' the output token list of the current group.

```

646 \exp_args:NNNV
647 \group_end:
648 \__grading_scheme_tl_set:Nn
649 #5
650 #5
651 <*log>
652 \__grading_scheme_log_decr:

```



```

653 \__grading_scheme_log:x { / Formatting ~ block ~ '\token_to_str:N #1' }
654 </log>
655 }
656 \cs_new:Npn \__grading_scheme_block_format:PnnnN
657 {
658 \exp_args:NP \__grading_scheme_block_format:NnnnN
659 }

```

(End definition for __grading_scheme_block_format:NnnnN and __grading_scheme_block_format:PnnnN.)

__grading_scheme_block_assert_operation:N Asserts that the block has an operation and loads it. If no operation is set, an error is emitted.

```

660 \cs_new:Npn \__grading_scheme_block_assert_operation:N #1
661 {
662 \__grading_scheme_block_load_operation:N #1
663 \quark_if_no_value:NT \l__grading_scheme_block_operation_tl
664 {
665 \msg_error:nnxxx { grading-scheme } { missing-value}
666 { block }
667 { \token_to_str:N #1 }
668 { operation }
669 }
670 }

```

(End definition for __grading_scheme_block_assert_operation:N.)

__grading_scheme_block_indent:nN __grading_scheme_block_indent:nN{<indent>}<tl var>
Performs the indent into <tl var> iff \l__grading_scheme_block_indent_bool is true.
Sets \l__grading_scheme_block_indent_bool to true afterwards.

```

671 \cs_new:Npn \__grading_scheme_block_indent:nN #1 #2
672 {
673 \bool_if:NT \l__grading_scheme_block_indent_bool
674 {
675 \__grading_scheme_tl_put_right:Nx #2
676 {
677 \prg_replicate:nn { #1 } { & }
678 }
679 }
680 \bool_set_true:N \l__grading_scheme_block_indent_bool
681 }

```

(End definition for __grading_scheme_block_indent:nN.)

4.9.3 Width and height of a block

\l__grading_scheme_height_sum_int Some temporary int values

```

\l__grading_scheme_height_acc_int
682 \int_new:N \l__grading_scheme_height_sum_int
683 \int_new:N \l__grading_scheme_height_acc_int

```

(End definition for \l__grading_scheme_height_sum_int and \l__grading_scheme_height_acc_int.)

```

\grading_scheme_block_get_height:NN      \grading_scheme_block_get_height:NN<block var><int var>
\grading_scheme_block_get_height:PN      Gets the height of a block and stores it into the <int var>.

684 \cs_new:Npn \grading_scheme_block_get_height:NN #1 #2
685 {
686   <*log>
687   \__grading_scheme_log:x { Getting ~ height ~ of ~ block ~ '\token_to_str:N #1' }
688   \__grading_scheme_log_incr:
689   </log>

```

Grouping is needed to not mess up local variables in the recursion:

```

690   \group_begin:
691   \__grading_scheme_block_load_elements:N #1
692   \int_zero:N \l__grading_scheme_height_sum_int
693   \clist_map_function:PN
694     \l__grading_scheme_block_elements_clist_ptr
695     \__grading_scheme_block_height_accumulator:N
696   \grading_scheme_block_if_description:NT #1
697   {
698     \int_incr:N \l__grading_scheme_height_sum_int
699   }

```

Smuggle out this length at return it to the caller:

```

700   \exp_args:NNNV
701   \group_end:
702   \int_set:Nn #2 \l__grading_scheme_height_sum_int
703   <*log>
704   \__grading_scheme_log_decr:
705   \__grading_scheme_log:x
706   {
707     / Getting ~ height ~ of ~ block ~ '\token_to_str:N #1':
708     '\int_use:N #2'
709   }
710   </log>
711 }
712 \cs_new:Npn \grading_scheme_block_get_height:PN
713 {
714   \exp_args:NP \grading_scheme_block_get_height:NN
715 }

```

(End definition for \grading_scheme_block_get_height:NN and \grading_scheme_block_get_height:PN.
These functions are documented on page ??.)

__grading_scheme_block_height_accumulator:N This is mapped on the elements of a block and accumulates the heights.

```

716 \cs_new:Npn \__grading_scheme_block_height_accumulator:N #1
717 {
718   \grading_scheme_element_get_height:NN #1 \l__grading_scheme_height_acc_int
719   \int_add:Nn \l__grading_scheme_height_sum_int
720   { \int_use:N \l__grading_scheme_height_acc_int }
721 }

```

(End definition for __grading_scheme_block_height_accumulator:N.)

`\grading_scheme_block_if_description:NTF` Tests if the given $\langle block\ var \rangle$ has a description set or not. Also loads this description

```

722 \cs_new:Npn \grading_scheme_block_if_description:NT #1 % implicit #2
723 {
724   \__grading_scheme_block_load_description:N #1
725   \quark_if_no_value:NF \l__grading_scheme_block_description_tl
726 }
727 \cs_new:Npn \grading_scheme_block_if_description:NF #1 % implicit #2
728 {
729   \__grading_scheme_block_load_description:N #1
730   \quark_if_no_value:NT \l__grading_scheme_block_description_tl
731 }
732 \cs_new:Npn \grading_scheme_block_if_description:NTF #1 #2 #3
733 {
734   \__grading_scheme_block_load_description:N #1
735   \quark_if_no_value:NTF \l__grading_scheme_block_description_tl
736   { #3 }
737   { #2 }
738 }

```

(End definition for `\grading_scheme_block_if_description:NTF`. This function is documented on page ??.)

`\l__grading_scheme_max_width_int`

(End definition for `\l__grading_scheme_max_width_int`.)

`\l__grading_scheme_width_acc_int`

Some temporary int values

`\l__grading_scheme_max_width_int`

```

739 \int_new:N \l__grading_scheme_max_width_int
740 \int_new:N \l__grading_scheme_width_acc_int

```

(End definition for `\l__grading_scheme_width_acc_int` and `\l__grading_scheme_max_width_int`.)

`\grading_scheme_block_get_natural_width:NN`

Gets the “natural” width of a block, that is: The minimal width of this block so that it can be typeset properly.

`\grading_scheme_block_get_natural_width:PN`

```

741 \cs_new:Npn \grading_scheme_block_get_natural_width:NN #1 #2
742 {
743   <*log>
744   \__grading_scheme_log:x
745   { Getting ~ natural ~ width ~ of ~ block ~ '\token_to_str:N #1'. }
746   \__grading_scheme_log_incr:
747   </log>
748   \group_begin:
749   \__grading_scheme_block_load_elements:N #1
750   \int_zero:N \l__grading_scheme_max_width_int
751   \clist_map_function:PN
752   \l__grading_scheme_block_elements_clist_ptr
753   \__grading_scheme_block_width_accumulator:N
754   \int_incr:N \l__grading_scheme_max_width_int
755   \exp_args:NNNV
756   \group_end:
757   \int_set:Nn #2 \l__grading_scheme_max_width_int
758   <*log>
759   \__grading_scheme_log_decr:
760   \__grading_scheme_log:x

```

```

761         { / Getting ~ natural ~ width ~ of ~ block ~ '\token_to_str:N #1'. }
762     </log>
763 }
764 \cs_new:Npn \grading_scheme_block_get_natural_width:PN
765 {
766     \exp_args:NP \grading_scheme_block_get_natural_width:NN
767 }

```

(End definition for \grading_scheme_block_get_natural_width:NN and \grading_scheme_block_get_natural_width:PN. These functions are documented on page ??.)

_grading_scheme_block_width_accumulator:N Gets the natural width of subelements and accumulates the maximum of them.

```

768 \cs_new:Npn \_grading_scheme_block_width_accumulator:N #1
769 {
770     <*log>
771     \_grading_scheme_log:x { Accumulating ~ width ~ of ~ '\token_to_str:N #1' }
772     \_grading_scheme_log_incr:
773 </log>
774     \grading_scheme_element_get_natural_width:NN #1 \l__grading_scheme_width_acc_int
775     \int_set:Nn \l__grading_scheme_max_width_int
776     {
777         \int_max:nn
778         \l__grading_scheme_width_acc_int
779         \l__grading_scheme_max_width_int
780     }
781     <*log>
782     \_grading_scheme_log_decr:
783     \_grading_scheme_log:x { / Accumulationg ~ width ~ of ~ '\token_to_str:N #1' }
784 </log>
785 }

```

(End definition for _grading_scheme_block_width_accumulator:N.)

4.10 Elements

As discussed earlier, an $\langle element \rangle$ is either a $\langle block \rangle$ or an $\langle entry \rangle$. Thus, our natural representation is as follows:

```

struct Element {
    void* content;
    tl type;
}

```

which essentially just implements a union that knows of its current member.

```

\_grading_scheme_element_new:N
\_grading_scheme_element_clear:N
\_grading_scheme_element_set_eq:NN
\_grading_scheme_element_gclear:N
\_grading_scheme_element_gset_eq:NN

```

Do what these say. Forward to the underlying property lists.

```

786 \cs_set_eq:NN \grading_scheme_element_new:N \prop_new:N
787 \cs_set_eq:NN \grading_scheme_element_clear:N \prop_clear:N
788 \cs_set_eq:NN \grading_scheme_element_set_eq:NN \prop_set_eq:NN
789 \cs_set_eq:NN \grading_scheme_element_gclear:N \prop_gclear:N
790 \cs_set_eq:NN \grading_scheme_element_gset_eq:NN \prop_gset_eq:NN

```

(End definition for _grading_scheme_element_new:N and others.)

\l__grading_scheme_element_content_void_ptr
 \l__grading_scheme_element_type_str

Mentioned local variables.

```
791 \ptr_new:N \l__grading_scheme_element_content_void_ptr
792 \str_new:N \l__grading_scheme_element_type_str
```

(End definition for \l__grading_scheme_element_content_void_ptr and \l__grading_scheme_element_type_str.)

\grading_scheme_element_set_block:NN
 \grading_scheme_element_gset_block:NN
 \grading_scheme_element_set_entry:NN
 \grading_scheme_element_gset_entry:NN

```
\grading_scheme_element_set_block:NN <element var><block var>
793 \cs_new:Npn \grading_scheme_element_set_block:NN #1 % implicit #2
794 {
795   \prop_put:Nnn #1 { type } { block }
796   \prop_put:Nnn #1 { content }
797 }
798 \cs_new:Npn \grading_scheme_element_gset_block:NN #1 % implicit #2
799 {
800   \prop_gput:Nnn #1 { type } { block }
801   \prop_gput:Nnn #1 { content }
802 }
803 \cs_new:Npn \grading_scheme_element_set_entry:NN #1 % implicit #2
804 {
805   \prop_put:Nnn #1 { type } { entry }
806   \prop_put:Nnn #1 { content }
807 }
808 \cs_new:Npn \grading_scheme_element_gset_entry:NN #1 % implicit #2
809 {
810   \prop_gput:Nnn #1 { type } { entry }
811   \prop_gput:Nnn #1 { content }
812 }
```

(End definition for \grading_scheme_element_set_block:NN and others. These functions are documented on page ??.)

\grading_scheme_element_get_content:NN
 \grading_scheme_element_get_type:NN

\grading_scheme_element_get_content:NN <element var><void ptr>
 \grading_scheme_element_get_type:NN <element var><str var>
 Get the contents. The assignment is local.

```
813 \cs_new:Npn \grading_scheme_element_get_content:NN #1 % implicit #2
814 {
815   \prop_get:NnN #1 { content }
816 }
817 \cs_new:Npn \grading_scheme_element_get_type:NN #1 % implicit #2
818 {
819   \prop_get:NnN #1 { type }
820 }
```

(End definition for \grading_scheme_element_get_content:NN and \grading_scheme_element_get_type:NN. These functions are documented on page ??.)

__grading_scheme_element_load_content:N
 __grading_scheme_element_load_type:NN

Loads the element into the local variables.

```
821 \cs_new:Npn \__grading_scheme_element_load_content:N #1
822 {
823   \grading_scheme_element_get_content:NN #1 \l__grading_scheme_element_content_void_ptr
824 }
825 \cs_new:Npn \__grading_scheme_element_load_type:N #1
826 {
```

```

827 \grading_scheme_element_get_type:NN #1 \l__grading_scheme_element_type_str
828 }

```

(End definition for __grading_scheme_element_load_content:N and __grading_scheme_element_load_type:NN.)

```

\grading_scheme_element_cases:Nnn      \grading_scheme_element_cases(element var){(entry code)}{(block code)}
\grading_scheme_element_cases:NnnTF    Distinguishes between the cases of the element type.
                                       Also loads the element type.

```

```

829 \cs_new:Npn \grading_scheme_element_cases:Nnn % implicit #1-3
830 {
831   \cs_set_eq:NN \__grading_scheme_str_case:Vnw \str_case:Vn
832   \__grading_scheme_element_cases:Nnnw
833 }
834 \cs_new:Npn \grading_scheme_element_cases:NnnT % implicit #1-4
835 {
836   \cs_set_eq:NN \__grading_scheme_str_case:Vnw \str_case:VnT
837   \__grading_scheme_element_cases:Nnnw
838 }
839 \cs_new:Npn \grading_scheme_element_cases:NnnF % implicit #1-4
840 {
841   \cs_set_eq:NN \__grading_scheme_str_case:Vnw \str_case:VnF
842   \__grading_scheme_element_cases:Nnnw
843 }
844 \cs_new:Npn \grading_scheme_element_cases:NnnTF % implicit #1-5
845 {
846   \cs_set_eq:NN \__grading_scheme_str_case:Vnw \str_case:VnTF
847   \__grading_scheme_element_cases:Nnnw
848 }

```

(End definition for \grading_scheme_element_cases:NnnTF. This function is documented on page ??.)

__grading_scheme_element_cases:Nnnw This assumes that __grading_scheme_str_case:Vnw has been set to a \str_case:VnTF variant and uses this variant for switching the type cases.

```

849 \cs_new:Npn \__grading_scheme_element_cases:Nnnw #1 #2 #3 % implicit branches
850 {
851   \__grading_scheme_element_load_type:N #1
852   \__grading_scheme_str_case:Vnw \l__grading_scheme_element_type_str
853   {
854     { entry } { #2 }
855     { block } { #3 }
856   }
857   % implicit branches
858 }

```

(End definition for __grading_scheme_element_cases:Nnnw.)

\grading_scheme_element_format:NnnnN Same syntax as \grading_scheme_block_format:NnnnN.

```

859 \cs_new:Npn \grading_scheme_element_format:NnnnN % implicit #1-5
860 {
861   \cs_set_eq:NN \__grading_scheme_entry_format_aux:PnnN \grading_scheme_entry_format:PnnN
862   \cs_set_eq:NN \__grading_scheme_block_format_aux:PnnnN \grading_scheme_block_format:PnnnN
863   \cs_set_eq:NN \__grading_scheme_tl_put_right:Nx \tl_put_right:Nx
864   \__grading_scheme_element_format:NnnnN

```

```

865 }
866 \cs_new:Npn \grading_scheme_element_gformat:NnnnN % implicit #1-5
867 {
868   \cs_set_eq:NN \__grading_scheme_entry_format_aux:PnnN \grading_scheme_entry_gformat:PnnN
869   \cs_set_eq:NN \__grading_scheme_block_format_aux:PnnnN \grading_scheme_block_gformat:PnnnN
870   \cs_set_eq:NN \__grading_scheme_tl_put_right:Nx \tl_put_right:Nx
871   \__grading_scheme_element_format:NnnnN
872 }
873 \cs_new:Npn \grading_scheme_element_format:PnnnN % implicit #1-5
874 {
875   \exp_args:NP \grading_scheme_element_format:NnnnN
876 }
877 \cs_new:Npn \grading_scheme_element_gformat:PnnnN % implicit #1-5
878 {
879   \exp_args:NP \grading_scheme_element_gformat:NnnnN
880 }

```

(End definition for \grading_scheme_element_format:NnnnN and others. These functions are documented on page ??.)

Aux function.

__grading_scheme_element_format:NnnnN
 __grading_scheme_element_format:PnnnN

```

881 \cs_new:Npn \__grading_scheme_element_format:NnnnN #1 #2 #3 #4 #5
882 {
883   <*log>
884   \__grading_scheme_log:x
885   {
886     Formatting ~ element ~ '\token_to_str:N #1' ~ into ~ '\token_to_str:N #5'.
887   }
888   \__grading_scheme_log_incr:
889   \prop_log:N #1
890   </log>
891   \__grading_scheme_element_load_content:N #1
892   \grading_scheme_element_cases:NnnF #1
893   {
894     \bool_if:nTF { #4 }
895     {
896       \__grading_scheme_entry_format_aux:PnnN
897       \l__grading_scheme_element_content_void_ptr
898       { #2 }
899       { #3 }
900       #5
901     }
902     {
903       \__grading_scheme_entry_format_aux:PnnN
904       \l__grading_scheme_element_content_void_ptr
905       { 0 }
906       { #3 }
907       #5
908     }
909   }
910   {
911     \__grading_scheme_block_format_aux:PnnnN
912     \l__grading_scheme_element_content_void_ptr
913     { #2 }

```

```

914         { #3 }
915         { #4 }
916         #5
917     }
918     {
919         \msg_error:nnxxx { grading-scheme } { missing-value }
920         { element }
921         { \token_to_str:N #1 }
922         { type / content }
923     }
924     \__grading_scheme_tl_put_right:Nx #5
925     {
926         \exp_not:N \__grading_scheme_cline:nn
927         {
928             \int_eval:n { #2 +1 }
929         }
930         {
931             \int_eval:n { #2 + #3 }
932         }
933     }
934     <*log>
935     \__grading_scheme_log_decr:
936     \__grading_scheme_log:x { Done typesetting ~ element ~ '\token_to_str:N #1' }
937     </log>
938     }
939     \cs_new:Npn \__grading_scheme_element_format:PnnnN % implicit #1-5
940     {
941         \exp_args:NP \__grading_scheme_element_format:NnnnN
942     }

```

(End definition for __grading_scheme_element_format:NnnnN and __grading_scheme_element_format:PnnnN.)

\grading_scheme_element_get_height:NN
\grading_scheme_element_get_height:PN

Get the the height of an element.

```

943 \cs_new:Npn \grading_scheme_element_get_height:NN #1 #2
944 {
945     \grading_scheme_element_cases:NnnF #1
946     {
947         \int_set:Nn #2 { 1 }
948     }
949     {
950         \__grading_scheme_element_load_content:N #1
951         \grading_scheme_block_get_height:PN
952         \l__grading_scheme_element_content_void_ptr
953         #2
954     }
955     {
956         \msg_error:nnxxx { grading-scheme } { missing-value }
957         { element }
958         { \token_to_str:N #1 }
959         { type / content }
960     }
961 }
962 \cs_new:Npn \grading_scheme_element_get_height:PN
963 {

```



```

964 \exp_args:NP \grading_scheme_element_get_height:NN
965 }

```

(End definition for \grading_scheme_element_get_height:NN and \grading_scheme_element_get_height:PN. These functions are documented on page ??.)

grading_scheme_element_get_natural_width:NN
grading_scheme_element_get_natural_width:PN

Get the natural width of an element

```

966 \cs_new:Npn \grading_scheme_element_get_natural_width:NN #1 #2
967 {
968   \grading_scheme_element_cases:NnnF #1
969   {
970     \int_set:Nn { #2 } { 2 }
971   }
972   {
973     \__grading_scheme_element_load_content:N #1
974     \grading_scheme_block_get_natural_width:PN
975     \l__grading_scheme_element_content_void_ptr
976     #2
977   }
978   {
979     \msg_error:nnxxx { grading-scheme } { missing-value }
980     { element }
981     { \token_to_str:N #1 }
982     { type / content }
983   }
984 }
985 \cs_new:Npn \grading_scheme_element_get_natural_width:PN
986 {
987   \exp_args:NP \grading_scheme_element_get_natural_width:NN
988 }

```

(End definition for \grading_scheme_element_get_natural_width:NN and \grading_scheme_element_get_natural_width:PN. These functions are documented on page ??.)

\l__grading_scheme_width_int

Local int vars for typesetting.

```

989 \int_new:N \l__grading_scheme_width_int

```

(End definition for \l__grading_scheme_width_int.)

\g__grading_scheme_table_tl

Token list where we will build the table.

```

990 \tl_new:N \g__grading_scheme_table_tl

```

(End definition for \g__grading_scheme_table_tl.)

\grading_scheme_element_typeset:N

Typesets this element as a tabular and inserts this into the output stream.

\grading_scheme_element_typeset:P

```

991 \cs_new:Npn \grading_scheme_element_typeset:N #1
992 {
993   \grading_scheme_element_get_natural_width:NN #1 \l__grading_scheme_width_int
994   \tl_gclear:N \g__grading_scheme_table_tl
995   \tl_gput_right:Nn \g__grading_scheme_table_tl
996   {
997     \begin{tabular}
998   }
999   \tl_gput_right:Nx \g__grading_scheme_table_tl
1000  {

```

```

1001     {
1002         \prg_replicate:nn { \l__grading_scheme_width_int -1 } { |1 }
1003         | 1 |
1004     }
1005     \exp_not:N \__grading_scheme_hline:
1006 }
1007 \grading_scheme_element_gformat:NnnnN
1008 #1
1009 { 0 }
1010 { \l__grading_scheme_width_int }
1011 { \c_false_bool }
1012 \g__grading_scheme_table_tl
1013 \tl_gput_right:Nn \g__grading_scheme_table_tl
1014 {
1015     \end{tabular}
1016 }
1017 \group_begin:
1018 \tl_set:Nn \arraystretch { 1.5 }
1019 \tl_use:N \g__grading_scheme_table_tl
1020 \group_end:
1021 }
1022 \cs_new:Npn \grading_scheme_element_typeset:P
1023 {
1024     \exp_args:NP \grading_scheme_element_typeset:N
1025 }

```

(End definition for \grading_scheme_element_typeset:N and \grading_scheme_element_typeset:P. These functions are documented on page ??.)

4.11 Facilities for populating data structures

TODO: what about global / local assignments here?

```

\grading_scheme_entry_new:Nnn      \grading_scheme_entry_new:Nnn<entry var>{<description>}{<points>}
\grading_scheme_entry_new:Pnn
1026 \cs_new:Npn \grading_scheme_entry_new:Nnn #1 #2 % implicit #3
1027 {
1028     \grading_scheme_entry_new:N #1
1029     \grading_scheme_entry_gset_description:Nn #1 { #2 }
1030     \grading_scheme_entry_gset_points:Nn #1
1031 }
1032 \cs_new:Npn \grading_scheme_entry_new:Pnn
1033 {
1034     \exp_args:NP \grading_scheme_entry_new:Nnn
1035 }

```

(End definition for \grading_scheme_entry_new:Nnn and \grading_scheme_entry_new:Pnn. These functions are documented on page ??.)

```

\grading_scheme_block_new:Nnn      \grading_scheme_block_new:Nnn<block var>{<description>}{<operation>}
\grading_scheme_block_new:Pnn
1036 \cs_new:Npn \grading_scheme_block_new:Nnn #1 #2 % implicit #3
1037 {
1038     \grading_scheme_block_new:N #1
1039     \grading_scheme_block_gset_description:Nn #1 { #2 }
1040     \grading_scheme_block_gset_operation:Nn #1

```

```

1041 }
1042 \cs_new:Npn \grading_scheme_block_new:Pnn
1043 {
1044   \exp_args:NP \grading_scheme_block_new:Nnn
1045 }

```

(End definition for `\grading_scheme_block_new:Nnn` and `\grading_scheme_block_new:Pnn`. These functions are documented on page ??.)

```

\grading_scheme_block_new:Nn      \grading_scheme_block_new:Nn⟨block var⟩{⟨operation⟩}
\grading_scheme_block_new:Pn      Same as above, but no description provided.

1046 \cs_new:Npn \grading_scheme_block_new:Nn #1 % implicit #2
1047 {
1048   \grading_scheme_block_new:N #1
1049   \grading_scheme_block_gset_operation:Nn #1
1050 }
1051 \cs_new:Npn \grading_scheme_block_new:Pn
1052 {
1053   \exp_args:NP \grading_scheme_block_new:Nn
1054 }

```

(End definition for `\grading_scheme_block_new:Nn` and `\grading_scheme_block_new:Pn`. These functions are documented on page ??.)

```

\grading_scheme_element_from_entry_new:NN      \grading_scheme_element_from_entry_new:NN⟨element var⟩⟨entry var⟩
\grading_scheme_element_from_entry_new:NP      wraps the entry into a new element.
\grading_scheme_element_from_entry_new:PP
\grading_scheme_element_gfrom_entry_new:NN
\grading_scheme_element_gfrom_entry_new:NP
\grading_scheme_element_gfrom_entry_new:PP
1055 \cs_new:Npn \grading_scheme_element_from_entry_new:NN #1 #2
1056 {
1057   \grading_scheme_element_new:N #1
1058   \grading_scheme_element_set_entry:NN #1 #2
1059 }
1060 \cs_new:Npn \grading_scheme_element_from_entry_new:NP
1061 {
1062   \exp_args:NP \grading_scheme_element_from_entry_new:NN
1063 }
1064 \cs_new:Npn \grading_scheme_element_from_entry_new:PP
1065 {
1066   \exp_args:NPP \grading_scheme_element_from_entry_new:NN
1067 }
1068 \cs_new:Npn \grading_scheme_element_gfrom_entry_new:NN #1 #2
1069 {
1070   \grading_scheme_element_new:N #1
1071   \grading_scheme_element_gset_entry:NN #1 #2
1072 }
1073 \cs_new:Npn \grading_scheme_element_gfrom_entry_new:NP
1074 {
1075   \exp_args:NP \grading_scheme_element_gfrom_entry_new:NN
1076 }
1077 \cs_new:Npn \grading_scheme_element_gfrom_entry_new:PP
1078 {
1079   \exp_args:NPP \grading_scheme_element_gfrom_entry_new:NN
1080 }

```

(End definition for `\grading_scheme_element_from_entry_new:NN` and others. These functions are documented on page ??.)

```

\grading_scheme_element_from_block_new:NN      \grading_scheme_element_from_block_new:NN⟨element var⟩⟨block var⟩
\grading_scheme_element_from_block_new:NP      wraps the block into a new element.
\grading_scheme_element_from_block_new:NP
\grading_scheme_element_gfrom_block_new:NN
\grading_scheme_element_from_gblock_new:NP
\grading_scheme_element_gfrom_block_new:NP
1081 \cs_new:Npn \grading_scheme_element_from_block_new:NN #1 #2
1082 {
1083   \grading_scheme_element_new:N #1
1084   \grading_scheme_element_set_block:NN #1 #2
1085 }
1086 \cs_new:Npn \grading_scheme_element_from_block_new:NP
1087 {
1088   \exp_args:NP \grading_scheme_element_from_block_new:NN
1089 }
1090 \cs_new:Npn \grading_scheme_element_from_block_new:PP
1091 {
1092   \exp_args:NPP \grading_scheme_element_from_block_new:NN
1093 }
1094 \cs_new:Npn \grading_scheme_element_gfrom_block_new:NN #1 #2
1095 {
1096   \grading_scheme_element_new:N #1
1097   \grading_scheme_element_gset_block:NN #1 #2
1098 }
1099 \cs_new:Npn \grading_scheme_element_gfrom_block_new:NP
1100 {
1101   \exp_args:NP \grading_scheme_element_gfrom_block_new:NN
1102 }
1103 \cs_new:Npn \grading_scheme_element_gfrom_block_new:PP
1104 {
1105   \exp_args:NPP \grading_scheme_element_gfrom_block_new:NN
1106 }

```

(End definition for `\grading_scheme_element_from_block_new:NN` and others. These functions are documented on page ??.)

4.12 Grading scheme environment

When building a grading scheme, we introduce a `\g__grading_scheme_curr_block_ptr` that is a pointer to the current block that is being populated.

Block environments and entries will create new blocks/entries and add themselves to the block pointed out by `\g__grading_scheme_curr_block_ptr`

```

\l__grading_scheme_entry_ptr
  \l__grading_scheme_element_ptr
  \l__grading_scheme_curr_block_ptr
1107 \ptr_new:Nn \l__grading_scheme_entry_ptr { g }
1108 \ptr_new:Nn \l__grading_scheme_element_ptr { g }
1109 \ptr_new:Nn \l__grading_scheme_curr_block_ptr { g }

```

(End definition for `\l__grading_scheme_entry_ptr`, `\l__grading_scheme_element_ptr`, and `\l__grading_scheme_curr_block_ptr`.)

`__grading_scheme_entry:nn` Creates an entry in the current block.

```

1110 \cs_new:Npn \__grading_scheme_entry:nn #1 #2
1111 {
1112   \ptr_clear:Nn \l__grading_scheme_entry_ptr { g }
1113   \ptr_clear:Nn \l__grading_scheme_element_ptr { g }
1114   \grading_scheme_entry_new:Pnn
1115     \l__grading_scheme_entry_ptr

```

```

1116     { #1 }
1117     { #2 }
1118     \grading_scheme_element_gfrom_entry_new:PP
1119     \l__grading_scheme_element_ptr
1120     \l__grading_scheme_entry_ptr
1121     \grading_scheme_block_gadd_element:PP
1122     \l__grading_scheme_curr_block_ptr
1123     \l__grading_scheme_element_ptr
1124 }

```

(End definition for __grading_scheme_entry:nn.)

__grading_scheme_block_begin:nn

__grading_scheme_block_begin:n

__grading_scheme_block_end:

```

1125 \cs_new:Npn \__grading_scheme_block_begin:nn % implicit #1, #2
1126 {
1127   \ptr_clear:Nn \l__grading_scheme_curr_block_ptr { g }
1128   \grading_scheme_block_new:Pnn \l__grading_scheme_curr_block_ptr
1129 }
1130 \cs_new:Npn \__grading_scheme_block_begin:n % implicit #1
1131 {
1132   \ptr_clear:Nn \l__grading_scheme_curr_block_ptr { g }
1133   \grading_scheme_block_new:Pn \l__grading_scheme_curr_block_ptr
1134 }
1135 \cs_new:Npn \__grading_scheme_block_end:
1136 {
1137   \ptr_clear:Nn \l__grading_scheme_element_ptr { g }
1138   \grading_scheme_element_gfrom_block_new:PP
1139   \l__grading_scheme_element_ptr
1140   \l__grading_scheme_curr_block_ptr
1141   \cs_gset:Npx \__grading_scheme_after_group:
1142   {
1143     \exp_not:N \grading_scheme_block_gadd_element:PN
1144     \exp_not:N \l__grading_scheme_curr_block_ptr
1145     \exp_not:P \l__grading_scheme_element_ptr
1146   }
1147   \group_insert_after:N \__grading_scheme_after_group:
1148 }

```

(End definition for __grading_scheme_block_begin:nn, __grading_scheme_block_begin:n, and __grading_scheme_block_end:.)

\entry

```

1149 \NewDocumentCommand \entry { m m }
1150 {
1151   \__grading_scheme_entry:nn { #1 } { #2 }
1152 }

```

(End definition for \entry. This function is documented on page 4.)

block

```

1153 \NewDocumentEnvironment { block } { o m }
1154 {
1155   \IfValueTF { #1 }
1156   {
1157     \__grading_scheme_block_begin:nn { #1 } { #2 }

```

```

1158     }
1159     {
1160         \__grading_scheme_block_begin:n { #2 }
1161     }
1162 }
1163 {
1164     \__grading_scheme_block_end:
1165 }

```

(End definition for block. This function is documented on page ??.)

gradingscheme

```

1166 \NewDocumentEnvironment {gradingscheme} { o m }
1167 {
1168     \bool_if:NT \g__grading_scheme_pipe_syntax_bool
1169     {
1170         \char_set_active_eq:NN | \__grading_scheme_entry_online:w
1171         \char_set_catcode_active:N |
1172     }
1173     \IfValueTF { #1 }
1174     {
1175         \__grading_scheme_block_begin:nn { #1 } { #2 }
1176     }
1177     {
1178         \__grading_scheme_block_begin:n { #2 }
1179     }
1180 }
1181 {
1182     \ptr_clear:Nn \l__grading_scheme_element_ptr { g }
1183     \grading_scheme_element_gfrom_block_new:PP
1184     \l__grading_scheme_element_ptr
1185     \l__grading_scheme_curr_block_ptr
1186     \grading_scheme_element_typeset:P \l__grading_scheme_element_ptr
1187 }

```

(End definition for gradingscheme. This function is documented on page ??.)

4.13 Implementing the pipe syntax

Everything left is conditional on the pipe option having been specified:

```

1188 \bool_if:NF \g__grading_scheme_pipe_syntax_bool
1189 {
1190     \endinput
1191 }

```

In order to offer syntax based on |, we make | an active character that will read until the end of the line, split at the @ character, and pass this on to a __grading_scheme_entry:nn

\s__grading_scheme_endline A scan mark that will denote the end of the line for us.

```

1192 \scan_new:N \s__grading_scheme_endline

```

(End definition for \s__grading_scheme_endline.)

```

\__grading_scheme_parse_entry:w      \__grading_scheme_parse_entry:w <description> & <points> \s_endline
Creates a new entry based on our scan mark.

1193 \cs_new:Npn \__grading_scheme_parse_entry:w #1 & #2 \s_endline
1194 {
1195     \__grading_scheme_entry:nn { #1 } { #2 }
1196 }

(End definition for \__grading_scheme_parse_entry:w.)

\__grading_scheme_replace_newline:w  Calling \__grading_scheme_replace_newline:w replaces the next occurrence of a new-
\__grading_scheme_replace_newline_aux:w line character with \s__grading_scheme_endline.
This is done by briefly switching category codes.

1197 \cs_new:Npn \__grading_scheme_replace_newline:w
1198 {
1199     \group_begin:
1200     \char_set_catcode_active:N^^M
1201     \__grading_scheme_replace_newline_aux:w
1202 }

We need to briefly make the newline character an active character in this source file
so that pattern matching works correctly.

1203 \char_set_catcode_active:N^^M
1204 \cs_new:Npn \__grading_scheme_replace_newline_aux:w #1 ^^M
1205 {
1206     \group_end:
1207     #1 \s_endline
1208 }
1209 \char_set_catcode_end_line:N^^M

(End definition for \__grading_scheme_replace_newline:w and \__grading_scheme_replace_newline_
aux:w.)

\__grading_scheme_entry_online:w      Parses a one-line entry.

1210 \cs_new:Npn \__grading_scheme_entry_online:w
1211 {
1212     \__grading_scheme_replace_newline:w
1213     \__grading_scheme_parse_entry:w
1214 }

(End definition for \__grading_scheme_entry_online:w.)

1215 </package>

```

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

Symbols	
\\	388, 599
\^	1200, 1203, 1209
\arraystretch	1018

B	
<code>\begin</code>	3 , 4 , 997
<code>block (environment)</code>	1
<code>block</code>	1153
bool commands:	
<code>\bool_if:NTF</code>	673 , 1168 , 1188
<code>\bool_if:nTF</code>	894
<code>\bool_new:N</code>	3 , 566
<code>\bool_set:Nn</code>	582
<code>\bool_set_false:N</code>	631
<code>\bool_set_true:N</code>	640 , 680
<code>\bool_to_str:n</code>	575
<code>\c_false_bool</code>	1011
<code>\c_true_bool</code>	582
C	
char commands:	
<code>\char_set_active_eq:NN</code>	1170
<code>\char_set_catcode_active:N</code>	1171 , 1200 , 1203
<code>\char_set_catcode_end_line:N</code> ..	1209
<code>\cline</code>	6 , 48
clist commands:	
<code>\clist_gput_right:Nn</code>	277
<code>\clist_gput_right:Pn</code>	243 , 524
<code>\clist_log:N</code>	579
<code>\clist_map_function:NN</code>	281
<code>\clist_map_function:PN</code>	243 , 643 , 693 , 751
<code>\clist_new:N</code>	265
<code>\clist_new:P</code>	243 , 513
<code>\clist_put_right:Nn</code>	273
<code>\clist_put_right:Pn</code>	243 , 519
<code>\clist_show:N</code>	269
<code>\clist_show:P</code>	243
cs commands:	
<code>\cs_generate_variant:Nn</code>	44 , 91 , 118 , 135
<code>\cs_gset:Npx</code>	1141
<code>\cs_if_exist:NTF</code>	233 , 237 , 241
<code>\cs_new:Npn</code>	23 , 27 , 31 , 49 , 54 , 58 , 62 , 68 , 73 , 78 , 94 , 120 , 136 , 168 , 173 , 180 , 194 , 199 , 205 , 210 , 215 , 220 , 225 , 231 , 235 , 239 , 243 , 247 , 251 , 255 , 259 , 263 , 267 , 271 , 275 , 279 , 283 , 287 , 311 , 315 , 319 , 323 , 327 , 331 , 335 , 339 , 343 , 348 , 353 , 357 , 361 , 395 , 399 , 408 , 429 , 433 , 437 , 441 , 445 , 449 , 453 , 457 , 461 , 465 , 469 , 473 , 477 , 481 , 485 , 492 , 501 , 516 , 521 , 526 , 530 , 534 , 538 , 542 , 549 , 556 , 561 , 568 , 656 , 660 , 671 , 684 , 712 , 716 , 722 , 727 , 732 , 741 , 764 , 768 , 793 , 798 , 803 , 808 , 813 , 817 , 821 , 825 , 829 , 834 , 839 , 844 , 849 , 859 , 866 , 873 , 877 , 881 , 939 , 943 , 962 , 966 , 985 , 991 , 1022 , 1026 , 1032 , 1036 , 1042 , 1046 , 1051 , 1055 , 1060 , 1064 , 1068 , 1073 , 1077 , 1081 , 1086 , 1090 , 1094 , 1099 , 1103 , 1110 , 1125 , 1130 , 1135 , 1193 , 1197 , 1204 , 1210
<code>\cs_set:Npn</code>	632
<code>\cs_set_eq:NN</code>	46 , 47 , 48 , 53 , 102 , 106 , 110 , 177 , 178 , 179 , 304 , 305 , 306 , 307 , 308 , 345 , 350 , 421 , 422 , 423 , 424 , 425 , 487 , 494 , 544 , 545 , 546 , 551 , 552 , 553 , 786 , 787 , 788 , 789 , 790 , 831 , 836 , 841 , 846 , 861 , 862 , 863 , 868 , 869 , 870
<code>\cs_to_str:N</code>	138 , 143 , 148
E	
<code>\end</code>	3 , 4 , 1015
<code>\endinput</code>	1190
<code>\entry</code>	4 , 1149
environments:	
<code>block</code>	1
<code>gradingscheme</code>	1
exp commands:	
<code>\exp_args:NNNPNP</code>	205
<code>\exp_args:NNNP</code>	205
<code>\exp_args:NNNV</code>	646 , 700 , 755
<code>\exp_args:NNP</code>	205 , 245 , 455 , 459
<code>\exp_args:NP</code> ...	205 , 233 , 237 , 241 , 249 , 257 , 261 , 265 , 269 , 273 , 277 , 281 , 285 , 300 , 355 , 359 , 397 , 536 , 540 , 558 , 563 , 579 , 658 , 714 , 766 , 875 , 879 , 941 , 964 , 987 , 1024 , 1034 , 1044 , 1053 , 1062 , 1075 , 1088 , 1101
<code>\exp_args:NPP</code>	205 , 253 , 528 , 532 , 1066 , 1079 , 1092 , 1105
<code>\exp_last_two_unbraced:Nnn</code>	229
<code>\exp_last_unbraced:NNNNo</code>	223
<code>\exp_last_unbraced:NNNo</code>	218
<code>\exp_last_unbraced:NNo</code>	213
<code>\exp_last_unbraced:No</code>	208
<code>\exp_not:N</code> 124 , 261 , 375 , 384 , 591 , 600 , 613 , 619 , 623 , 926 , 1005 , 1143 , 1144
<code>\exp_not:n</code> ..	36 , 41 , 300 , 381 , 597 , 625
<code>\exp_not:P</code>	243 , 294 , 300 , 1145
G	
grading commands:	
<code>\grading_scheme_block_add_-</code> <code>element:NN</code>	21 , 516

\grading_scheme_block_add_- element:PN	516	\grading_scheme_block_new:Pnn	1036, 1128
\grading_scheme_block_add_- element:PP	516	\grading_scheme_block_set_- description:Nn	429
\grading_scheme_block_clear:N	421	\grading_scheme_block_set_- elements:NN	429
\grading_scheme_block_format:NnnnN ...	22, 30, 542	\grading_scheme_block_set_- elements:NP	429, 489
\grading_scheme_block_format:PnnnN ...	542, 862	\grading_scheme_block_set_eq:NN	421
\grading_scheme_block_gadd_- element:NN	516	\grading_scheme_block_set_- operation:Nn	437
\grading_scheme_block_gadd_- element:PN	516, 1143	\grading_scheme_block_set_- operation_tl:Nn	429
\grading_scheme_block_gadd_- element:PP	516, 1121	\grading_scheme_element_cases ...	30
\grading_scheme_block_gclear:N	421	\grading_scheme_element_cases:Nnn ...	829
\grading_scheme_block_get_- description:NN	20, 461, 475	\grading_scheme_element_cases:NnnTF ...	829, 834, 839, 892, 945, 968
\grading_scheme_block_get_- elements:NN	20, 461, 483	\grading_scheme_element_clear:N	787
\grading_scheme_block_get_- height:NN	26, 585, 684	\grading_scheme_element_format:NnnnN ...	545, 859
\grading_scheme_block_get_- height:PN	684, 951	\grading_scheme_element_format:PnnnN ...	859
\grading_scheme_block_get_- natural_width:NN	741	\grading_scheme_element_from_- block_new:NN	36, 1081
\grading_scheme_block_get_- natural_width:PN	741, 974	\grading_scheme_element_from_- block_new:NP	1081
\grading_scheme_block_get_- operation:NN	20, 461, 479	\grading_scheme_element_from_- block_new:PP	1090
\grading_scheme_block_gformat:NnnnN ...	542	\grading_scheme_element_from_- entry_new:NN	35, 1055
\grading_scheme_block_gformat:PnnnN ...	542, 869	\grading_scheme_element_from_- entry_new:NP	1055
\grading_scheme_block_gset_- description:Nn	429, 1039	\grading_scheme_element_from_- entry_new:PP	1055
\grading_scheme_block_gset_- elements:NN	429	\grading_scheme_element_from_- gblock_new:NP	1081
\grading_scheme_block_gset_- elements:NP	429, 496	\grading_scheme_element_gclear:N	789
\grading_scheme_block_gset_eq:NN	421	\grading_scheme_element_get_- content:NN	29, 813, 823
\grading_scheme_block_gset_- operation:Nn	441, 1040, 1049	\grading_scheme_element_get_- height:NN	718, 943
\grading_scheme_block_gset_- operation_tl:Nn	429	\grading_scheme_element_get_- height:PN	943
\grading_scheme_block_if_- description:Nn	696, 722, 722, 727	\grading_scheme_element_get_- natural_width:NN	774, 966, 993
\grading_scheme_block_new:N	421, 1038, 1048	\grading_scheme_element_get_- natural_width:PN	966
\grading_scheme_block_new:Nn	35, 1046	\grading_scheme_element_get_- type:NN	29, 813, 827
\grading_scheme_block_new:Nnn	34, 1036	\grading_scheme_element_gformat:NnnnN ...	552, 859, 1007
\grading_scheme_block_new:Pn	1046, 1133	\grading_scheme_element_gformat:PnnnN ...	859

\grading_scheme_element_gfrom_- block_new:NN	1081	\grading_scheme_entry_put:NnnN ..	17
\grading_scheme_element_gfrom_- block_new:NP	1081	\grading_scheme_entry_set_- description:Nn	311
\grading_scheme_element_gfrom_- block_new:PP	1103 , 1138 , 1183	\grading_scheme_entry_set_eq:NN	304
\grading_scheme_element_gfrom_- entry_new:NN	1055	\grading_scheme_entry_set_- points:Nn	319
\grading_scheme_element_gfrom_- entry_new:NP	1055	grading internal commands:	
\grading_scheme_element_gfrom_- entry_new:PP	1055 , 1118	__grading_scheme_tl_put_- right:Nn	23
\grading_scheme_element_gset_- block:NN	793 , 1097	__grading_scheme_after_group:	1141 , 1147
\grading_scheme_element_gset_- entry:NN	793 , 1071	__grading_scheme_block_assert_- operation:N	610 , 660
\grading_scheme_element_gset_- eq:NN	790	__grading_scheme_block_begin:n	1125 , 1160 , 1178
\grading_scheme_element_new:N	786 , 1057 , 1070 , 1083 , 1096	__grading_scheme_block_begin:nn	1125 , 1157 , 1175
\grading_scheme_element_set_- block:NN	29 , 793 , 1084	\l__grading_scheme_block_- description_tl	426 , 475 , 586 , 597 , 725 , 730 , 735
\grading_scheme_element_set_- entry:NN	793 , 1058	\l__grading_scheme_block_- elements_clist_ptr	426 , 483 , 504 , 506 , 509 , 511 , 513 , 519 , 524 , 579 , 644 , 694 , 752
\grading_scheme_element_set_- eq:NN	788	__grading_scheme_block_end:	1125 , 1164
\grading_scheme_element_typeset:N	991	__grading_scheme_block_ensure_- elements:N	485 , 518 , 642
\grading_scheme_element_typeset:P	991 , 1186	__grading_scheme_block_ensure_- elements_aux:nN	490 , 497 , 501
\grading_scheme_entry_clear:N ..	304	__grading_scheme_block_format:NnnnN	547 , 554 , 568
\grading_scheme_entry_format:NnnN	22 , 343	__grading_scheme_block_format:PnnnN	568
\grading_scheme_entry_format:PnnN	343 , 861	__grading_scheme_block_format_- aux:PnnnN	862 , 869 , 911
\grading_scheme_entry_gclear:N .	304	__grading_scheme_block_format_- element:N	632 , 645
\grading_scheme_entry_get_- description:NN	327 , 341	__grading_scheme_block_gensure_- elements:N	492 , 523
\grading_scheme_entry_get_- points:NN	331 , 337	__grading_scheme_block_height_- accumulator:N	695 , 716
\grading_scheme_entry_gformat:NnnN	343	__grading_scheme_block_height_- int	567
\grading_scheme_entry_gformat:PnnN	343 , 868	\l__grading_scheme_block_height_- int	567 , 585 , 615
\grading_scheme_entry_gset_- description:Nn	311 , 1029	__grading_scheme_block_indent:nN	22 , 25 , 588 , 609 , 671
\grading_scheme_entry_gset_eq:NN	304	\l__grading_scheme_block_indent_- bool	25 , 566 , 582 , 631 , 638 , 640 , 673 , 680
\grading_scheme_entry_gset_- points:Nn	319 , 1030	__grading_scheme_block_load_- description:N	473 , 584 , 724 , 729 , 734
\grading_scheme_entry_new:N	304 , 1028		
\grading_scheme_entry_new:Nnn	34 , 1026		
\grading_scheme_entry_new:Pnn	1026 , 1114		

__grading_scheme_block_load_-
 elements:N . 481, 503, 578, 691, 749
 __grading_scheme_block_load_-
 elements:NN 473
 __grading_scheme_block_load_-
 operation:N 473, 662
 \l__grading_scheme_block_-
 operation_tl ... 426, 479, 625, 663
 __grading_scheme_block_set_-
 elements_optg:NP 488, 495, 500, 507
 __grading_scheme_block_width_-
 accumulator:N 753, 768
 __grading_scheme_cline:n 48
 __grading_scheme_cline:nn
 48, 600, 926
 \g__grading_scheme_curr_block_-
 ptr 36
 \l__grading_scheme_curr_block_-
 ptr 1107, 1122, 1127,
 1128, 1132, 1133, 1140, 1144, 1185
 __grading_scheme_element_-
 cases:Nnnw . 832, 837, 842, 847, 849
 __grading_scheme_element_-
 clear:N 786
 \l__grading_scheme_element_-
 content_void_ptr
 791, 823, 897, 904, 912, 952, 975
 __grading_scheme_element_-
 format:NnnnN 864, 871, 881
 __grading_scheme_element_-
 format:PnnnN 881
 __grading_scheme_element_-
 format_optg:NnnnN .. 545, 552, 634
 __grading_scheme_element_-
 gclear:N 786
 __grading_scheme_element_gset_-
 eq:NN 786
 __grading_scheme_element_load_-
 content:N 821, 891, 950, 973
 __grading_scheme_element_load_-
 type:N 825, 851
 __grading_scheme_element_load_-
 type:NN 821
 __grading_scheme_element_new:N 786
 \l__grading_scheme_element_ptr ..
 1107, 1113, 1119, 1123,
 1137, 1139, 1145, 1182, 1184, 1186
 __grading_scheme_element_set_-
 eq:NN 786
 \l__grading_scheme_element_type_-
 str 791, 827, 852
 __grading_scheme_entry:nn
 38, 1110, 1151, 1195
 __grading_scheme_entry_assert_-
 description:N 370, 399
 __grading_scheme_entry_assert_-
 points:N 371, 399
 \l__grading_scheme_entry_-
 description_tl . 309, 341, 381, 402
 __grading_scheme_entry_format:NnnN
 346, 351, 361
 __grading_scheme_entry_format:PnnN
 361
 __grading_scheme_entry_format_-
 aux:PnnN 861, 868, 896, 903
 __grading_scheme_entry_forformat:PnnN
 395
 __grading_scheme_entry_load_-
 description:N 339, 401
 __grading_scheme_entry_load_-
 points:N 335, 410
 __grading_scheme_entry_online:w
 1170, 1210
 \l__grading_scheme_entry_points_-
 int 309, 337, 386, 411
 \l__grading_scheme_entry_ptr ...
 1107, 1112, 1115, 1120
 \l__grading_scheme_height_acc_-
 int 682, 718, 720
 \l__grading_scheme_height_sum_-
 int 682, 692, 698, 702, 719
 __grading_scheme_hline: ... 53, 1005
 __grading_scheme_log:n
 31, 364, 366, 392, 571, 653,
 687, 705, 744, 760, 771, 783, 884, 936
 __grading_scheme_log_decr:
 27, 391, 652, 704, 759, 782, 935
 \g__grading_scheme_log_depth_int
 22, 25, 29, 35, 40
 __grading_scheme_log_incr:
 23, 368, 580, 688, 746, 772, 888
 \l__grading_scheme_max_width_int
 739, 739, 750, 754, 757, 775, 779
 __grading_scheme_multicolumn:nnn
 46, 375, 591
 __grading_scheme_multirow:nnn ..
 47, 613
 __grading_scheme_operation:n 62, 623
 __grading_scheme_parse_entry:w .
 39, 1193, 1213
 \g__grading_scheme_pipe_syntax_-
 bool 3, 6, 1168, 1188
 __grading_scheme_points:n .. 58, 384
 __grading_scheme_replace_-
 newline:w 39, 1197, 1212
 __grading_scheme_replace_-
 newline_aux:w 1197

__grading_scheme_rotatebox:nnn .	\msg_new:nnnn 161
. 54, 619	\multicolumn 6, 46
__grading_scheme_str_case:nnw . .	\multirow 6, 47
. 30, 831, 836, 841, 846, 852	
\g__grading_scheme_table_tl	N
. 990, 994, 995, 999, 1012, 1013, 1019	\NewDocumentCommand 1149
__grading_scheme_tl_put_-	\NewDocumentEnvironment 1153, 1166
right:Nn 17, 345, 350, 372,	
544, 551, 589, 611, 675, 863, 870, 924	P
__grading_scheme_tl_set:Nn	prg commands:
. 546, 553, 648	\prg_replicate:nn 35, 40, 374, 677, 1002
\l__grading_scheme_width_acc_int	\ProcessKeysOptions 18
. 739, 774, 778	prop commands:
\l__grading_scheme_width_int	\prop_clear:N 305, 422, 787
. 989, 993, 1002, 1010	\prop_gclear:N 307, 424, 789
gradingscheme (environment) 1	\prop_get:NnN
gradingscheme 1166 7, 329, 333, 463, 467, 471, 815, 819
group commands:	\prop_gput:Nnn 317,
\group_begin: 583, 690, 748, 1017, 1199	325, 435, 443, 451, 800, 801, 810, 811
\group_end: . 647, 701, 756, 1020, 1206	\prop_gset_eq:NN 308, 423, 425, 790
\group_insert_after:N 1147	\prop_log:N 367, 577, 889
	\prop_new:N 7, 304, 421, 786
H	\prop_put:Nn 7
\hline 6, 53	\prop_put:Nnn 313,
	321, 431, 439, 447, 795, 796, 805, 806
I	\prop_set_eq:NN 306, 788
\IfValueTF 1155, 1173	\prop_show:N 285
int commands:	\prop_show:P 243
\int_add:Nn 719	ptr commands:
\int_eval:n 321,	\ptr_clear:N 168
325, 377, 593, 602, 605, 615, 928, 931	\ptr_clear:Nn 173,
\int_gdecr:N 29	506, 1112, 1113, 1127, 1132, 1137, 1182
\int_gincr:N 25, 133	\ptr_if_null:NTF 231, 231, 235, 290, 511
\int_incr:N 698, 754	\ptr_new:N 9, 68, 426, 791
\int_max:nn 777	\ptr_new:Nn 73, 1107, 1108, 1109
\int_new:N 22,	\ptr_new:Nnn 71, 75, 78, 91
66, 309, 567, 682, 683, 739, 740, 989	\ptr_set:NN 177
\int_set:Nn 702, 757, 775, 947, 970	\ptr_set_eq:NN 178
\int_use:N 128, 615, 708, 720	\ptr_show:N 287
\int_zero:N 692, 750	\ptr_use:N 179
iow commands:	ptr internal commands:
\iow_log:n 38	__ptr_check:N 180, 196,
\iow_term:n 33	201, 207, 212, 217, 222, 227, 228, 289
K	__ptr_check:NN 194
keys commands:	__ptr_check:nN 199
\keys_define:nn 4	\l_ptr_content_tl 204
\l_keys_key_str 10	\g_ptr_counter_int 66, 128, 133
	__ptr_get_var_prefix:NN
M 70, 96, 136, 170
msg commands:	__ptr_init:Nnn
\msg_error:nnn 10, 153, 184 89, 94, 118, 135, 171, 175
\msg_error:nnnn	__ptr_init_aux:Nnn 116, 120
. 404, 413, 665, 919, 956, 979	__ptr_poly_tl_set:Nn
\msg_new:nnn 13, 190, 417 102, 106, 110, 122

<code>\l_ptr_var_prefix_str</code>		T	
. 67 , 70 , 71 , 96 , 98 , 114 , 170 , 171		<code>\textbf</code>	60
	Q	tl commands:	
quark commands:		<code>\tl_const:Nn</code>	110
<code>\q_no_value</code>	20	<code>\tl_gclear:N</code>	994
<code>\quark_if_no_value:NTF</code>		<code>\tl_gput_right:Nn</code>	
402 , 411 , 504 , 586 , 663 , 725 , 730 , 735		350 , 551 , 995 , 999 , 1013
	R	<code>\tl_gset:Nn</code>	106 , 553
<code>\RequirePackage</code>	17 , 19 , 20	<code>\tl_if_head_eq_charcode:nNTF</code>	
<code>\rotatebox</code>	56	138 , 143 , 148
	S	<code>\tl_if_single:NTF</code>	182
<code>\sc</code>	64	<code>\tl_new:N</code>	87 , 204 , 310 , 427 , 428 , 990
scan commands:		<code>\tl_put_gright:Nn</code>	870
<code>\s_endline</code>	39 , 1193 , 1207	<code>\tl_put_right:Nn</code>	345 , 544 , 863
<code>\scan_new:N</code>	1192	<code>\tl_set:Nn</code>	102 , 177 , 546 , 1018
scan internal commands:		<code>\tl_set_eq:NN</code>	14 , 178 , 245 , 249 , 253
<code>\s_grading_scheme_endline</code>	39 , 1192	<code>\tl_set_eq:NP</code>	243
str commands:		<code>\tl_set_eq:PN</code>	243
<code>\str_case:nn</code>	831	<code>\tl_set_eq:PP</code>	243
<code>\str_case:nnTF</code>	30 , 80 , 97 , 836 , 841 , 846	<code>\tl_show:n</code>	292 , 298
<code>\str_new:N</code>	67 , 792	<code>\tl_use:N</code>	179 , 257 , 386 , 1019
<code>\str_set:Nn</code>	140 , 145 , 150	<code>\tl_use:P</code>	243
<code>\str_show:N</code>	114	token commands:	
		<code>\token_to_str:N</code>	
		155 , 186 , 294 , 300 , 365 , 392 , 405 , 414 , 573 , 653 , 667 , 687 , 707 , 745 , 761 , 771 , 783 , 886 , 921 , 936 , 958 , 981