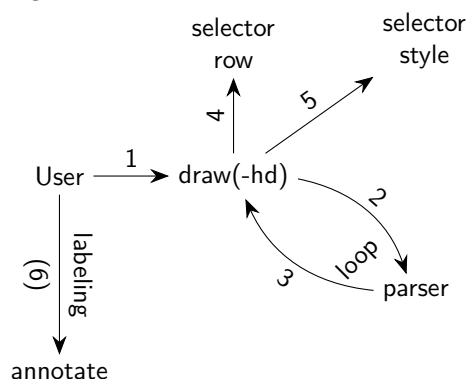


hexdumptikz-code – Printing and annotating hexdumps with TikZ

Lukas Heindl*

v1.0.0 from 2026-06-16

High-level overview of how the different modules / sub-packages work together:



File I Implementation

Identify the package and give the over all version information.

```
1 \ProvidesExplPackage {hexdumptikz} {2026-06-16} {1.0.0}
2 {Printing and annotating hexdumps with TikZ}

Load external dependencies
3 \RequirePackage { tikz }
4 \ExplSyntaxOff
5 \usetikzlibrary{chains}
6 \ExplSyntaxOn

Load internal (sub-)packages
7 \RequirePackage { hexdumptikz-common }
8 \RequirePackage { hexdumptikz-draw }
9 \RequirePackage { hexdumptikz-annotate }
10 \RequirePackage { hexdumptikz-addrcalc }
```

*E-Mail: oss.heindl+latex@protonmail.com

Issue tracker: codeberg.org/atticus-sullivan/hexdumptikz/issues

Codeberg (mirror): codeberg.org/atticus-sullivan/hexdumptikz

1.1 Errors

unknown-row (*error*)

```
11 \msg_new:nnn { hexdumptikz } { unknown-row }
12 { The~(#2)~row~with~the~id~#1~is~unknown. }
```

invalid-address (*error*)

```
13 \msg_new:nnn { hexdumptikz } { invalid-address }
14 { #1~is~not~a~valid~address. }
```

address-too-short (*error*)

```
15 \msg_new:nnn { hexdumptikz } { address-too-short }
16 { Address~'~#1'~is~too~short.~Must~be~at~least~#2~long. }
```

invalid-boolean (*error*)

```
17 \msg_new:nnn { hexdumptikz } { invalid-boolean }
18 { '~#1'~cannot~be~parsed~as~boolean. }
```

weird-num-bytes-per-row (*error*)

```
19 \msg_new:nnn { hexdumptikz } { weird-num-bytes-per-row }
20 { '~#1'~is~a~weird~num~bytes~per~row. }
```

1.2 Main/Public functions/Macros

\@@_begin:nn (*fn.*) Begin-code when a new hexdumptikz environment is started

Arguments:

```
#1 in pgfkeys
#2 in filename from which to read the hexdump from

21 \cs_new_protected:Npn \@@_begin:nn #1 #2 {
22   \pgfkeys {
23     /hexdumptikz,
24     #1
25   }
26   \str_set:Nl \l_hexdumptikz_common_input_file_str { #2 }
27   \begin { scope }[/hexdumptikz/next~scope]
28     \pgfkeys { /hexdumptikz/next~scope/.style = { } }
29 }
```

\@@_end: (*fn.*) End-code when a hexdumptikz environment stops – doing come cleanup

Arguments:

```
30 \cs_new_protected:Nn \@@_end: {
31   \prop_gclear:N \g_hexdumptikz_common_cur_offsets_prop
32   \end { scope }
33 }
```

hexdumptikz (*env.*) Define an environment which wraps the hexdumptikz related operations. SHOULD be placed inside a tikz-environment. TODO

```
34 \NewDocumentEnvironment { hexdumptikz } { 0 { } m }
35 {
36   \@@_begin:nn { #1 } { #2 }
```

```

37 } {
38   \@@_end:
39 }

```

\hexdumptikzPrint Define a macro which prints the hexdump which was previously specified (via its filename). SHOULD be placed inside a hexdumptikz-environment TODO

```

40 \NewDocumentCommand{ \hexdumptikzPrint } { 0 { } m }
41 {
42   \hexdumptikz_draw_print:nn { #1 } { #2 }
43 }

```

\hexdumptikzLabelBytes Define a macro which labels a sequence of bytes in the printed hexdump. SHOULD be placed inside a hexdumptikz-environment TODO Should be placed after the hexdump was printed.

```

44 \NewDocumentCommand{ \hexdumptikzLabelBytes } { 0 { } mm }
45 {
46   \__hexdumptikz_annotate_label_bytes:nnn { #1 } { #2 } { #3 }
47 }

```

\hexdumptikzLabelBytesFallback Define a macro which labels a sequence of bytes in the printed hexdump. Also works without the data gathered while printing the dump (outside of the environment). But the output does look a little bit less good compared to the normal annotation.

```

48 \NewDocumentCommand{ \hexdumptikzLabelBytesFallback } { 0 { } mm }
49 {
50   \__hexdumptikz_annotate_label_bytes_fallback:nnn { #1 } { #2 } { #3 }
51 }

```

\hexdumptikzAddrToNodename Define a helper macro which converts a full address to its nodename in the hexdump. Can be used by the user for custom drawings.

```

52 \NewDocumentCommand{ \hexdumptikzAddrToNodename } { m }
53 {
54   \__hexdumptikz_addrcalc_address_to_nodename:e { #1 }
55 }

```

\hexdumptikzAddrToRow Define a helper macro which converts a full address to the row in which the byte is placed. Can be used by the user for custom drawings.

```

56 \NewDocumentCommand{ \hexdumptikzAddrToRow } { m }
57 {
58   \__hexdumptikz_addrcalc_address_to_row:e { #1 }
59 }

```

\hexdumptikzAddrToCol Define a helper macro which converts a full address to the column in which the byte is placed. Can be used by the user for custom drawings.

```

60 \NewDocumentCommand{ \hexdumptikzAddrToCol } { m }
61 {
62   \__hexdumptikz_addrcalc_address_to_col:e { #1 }
63 }

```

hexdumptikz-common

Lukas Heindl

v1.0.0 from 2026-06-16

Abstract

Defines helpers and variables which are use throughout the whole project.

Identify the package and give the over all version information.

```
64 \ProvidesExplPackage {hexdumptikz-common} {2026-06-16} {1.0.0}  
65 {Common functionality for working with hexdumps and (large) addresses}
```

2.1 Conventions of this (sub-)package

None of the function defined in this (sub-)package makes use of the expl3 scratch-variables.

The idea behind this is that this (sub-)package only defines small helper functions. If they would clobber expl3 scratch-variables functions calling the helpers would not be able to use the normal expl3 scratch-variables anymore. Thus, this (sub-)package defines its own scratch-variables if functions need them.

2.2 Implementation

2.2.1 pgfkeys

Set up common pgf-keys / the space where all pgfkeys will live in

```
66 \RequirePackage { pgfkeys }  
67 \pgfkeys {
```

Make it a real “directory”. /tikz should also be searched to allow easily setting tikz-options.

```
68 /hexdumptikz/.is~family,  
69 /hexdumptikz/.search~also = { /tikz },
```

Not all tikz-options work natively. Thus, this option allows setting options on the next scope.

Note: All tikz drawings are wrapped in a scope.

/hexdumptikz/tikz (*pgfkey*)

```
70 /hexdumptikz/next~scope/.style = { },  
71 /hexdumptikz/tikz/.style = { /hexdumptikz/next~scope/.append~style = { #1 } },  
72 }
```

2.3 Regexes

Set up regexes commonly used through the whole package

`\hexdumptikz_common_hex_regex (var.)` Matches an hexadecimal number

```
73 \regex_const:Nn
74   \c_hexdumptikz_common_hex_regex
75   { \A 0x[0-9a-fA-F]+ \Z }
```

`\hexdumptikz_common_hex_x_regex (var.)` Matches a hexadecimal number which is (optionally) accompanied by a specification of the x-index.

```
76 \regex_const:Nn
77   \c_hexdumptikz_common_hex_x_regex
78   { \A 0x[0-9a-fA-F]+ (?: / [0-9]+ )? \Z }
```

`\hexdumptikz_common_idx_regex (var.)` Matches an index specification. Can (optionally) be accompanied by a specification of the x-index.

```
79 \regex_const:Nn
80   \c_hexdumptikz_common_idx_regex
81   { \A [0-9]+ (?: / [0-9]+ )? \Z }
```

`\hexdumptikz_common_leading_hex_base_regex (var.)` Matches a *leading* hexadecimal-base specifier.

```
82 \regex_const:Nn
83   \c_hexdumptikz_common_leading_hex_base_regex
84   { \A 0x }
```

Generate variants of `\expl` functions which are used often (having this in `\hexdumptikz_common` means this only runs once)

```
85 \cs_generate_variant:Nn \int_from_hex:n { f }
86 \cs_generate_variant:Nn \msg_critical:nnn { nnV }
87 \cs_generate_variant:Nn \msg_critical:nnnn { nnVV , nneV }
88 \cs_generate_variant:Nn \msg_critical:nnnnn { nnVeV }
89 \cs_generate_variant:Nn \msg_warning:nnn { nnV }
```

2.4 Variables used by multiple other (sub-)packages

Define the variables only once here. Note that this does not mean the variables are global.

`\hexdumptikz_common_nodename_prefix_tl (var.)` Prefix for nodenames generated

```
90 \tl_new:N \l_hexdumptikz_common_nodename_prefix_tl
```

`\hexdumptikz_common_input_file_str (var.)` Path to the input file from which to read the hexdump from

```
91 \str_new:N \l_hexdumptikz_common_input_file_str
```

`\hexdumptikz_common_bytes_per_row_int (var.)` Number of bytes shown in one row. (optionally) enforced by the parser and used at some other locations too.

```
92 \int_new:N \l_hexdumptikz_common_bytes_per_row_int
```

`\tikz_common_addr_len_int (var.)` padd addresses to this length (only refers to the hex-digits without the leading 0x)

```
93 \int_new:N \l_hexdumptikz_common_addr_len_int
```

`\tikz_common_cur_offsets_prop (var.)` Stores a mapping *addr/offset* to *out-index*. Required to draw from one row to the row above/below since this package avoids calculating on the addresses.

needs to be global since the assignment happens inside a tikz-scope so otherwise the assignment is gone when the prop is needed

```
94 \prop_new:N \g_hexdumptikz_common_cur_offsets_prop
```

2.5 Functions

`\tikz_common_pad_left:Nnn (fn.)` Sideeffects:

```
clobbered   dir   l_tmpb_int
```

Arguments:

```
#1  in/out   tl variable to work on
```

```
#2   in      target length (int expression)
```

```
#3   in      padding token (should have length 1)
```

```
95 \cs_new_protected:Npn \hexdumptikz_common_pad_left:Nnn #1 #2 #3
```

```
96 {
```

calculate the needed padding (target - current length)

```
97   \int_set:Nn \l_tmpb_int { #2 - \tl_count:N #1 }
```

perform the padding

```
98   \int_compare:nNtT { \l_tmpb_int } > { 0 } {
```

```
99     {
```

```
100       \tl_put_left:Ne
```

```
101         #1
```

```
102         { \prg_replicate:nn { \l_tmpb_int } { #3 } }
```

```
103     }
```

```
104 }
```

`\tikz_common_pad_address:N (fn.)` Sideeffects:

```
clobbered   indir   l_tmpa_int
```

```
clobbered   indir   l_tmpb_int
```

Arguments:

```
#1  in/out   address (tl)
```

```
#-   in      l_hexdumptikz_common_addr_len_int
```

```
105 \cs_new_protected:Npn \hexdumptikz_common_pad_address:N #1
```

```
106 {
```

Remove leading 0x (hex base indicator)

```
107   \regex_replace_once:NnNTF
```

```
108     \c_hexdumptikz_common_leading_hex_base_regex
```

```
109     { }
```

```
110     #1
```

```
111   {
```

Pad the address and place back the hex base indicator

```

112   \hexdumptikz_common_pad_left:Nnn
113   #1
114   { \l_hexdumptikz_common_addr_len_int }
115   { 0 }
116   \tl_put_left:Nn #1 { 0x }
117 }
118 {

```

Pad the address (no base-indicator present before -i, none present after)

```

119   \hexdumptikz_common_pad_left:Nnn
120   #1
121   { \l_hexdumptikz_common_addr_len_int }
122   { 0 }
123 }
124 }

```

otikz_common_parse_bool:Nn (*fn.*) Sideeffects:

Arguments:

```

#1 out bool var (bool)
#2 in user input

```

```

125 \cs_new_protected:Npn \hexdumptikz_common_parse_bool:Nn #1 #2
126 {
127   \str_case:enF { \tl_trim_spaces:n { #2 } }
128   {
129     { true } { \bool_set_true:N #1 }
130     { false } { \bool_set_false:N #1 }
131   }
132   {
133     \msg_critical:nne { hexdumptikz } { invalid-boolean } { #2 }
134   }
135 }

```

hexdumptikz-draw

Lukas Heindl

v1.0.0 from 2026-06-16

Abstract

Draws the hexdump. For the most part this defines configuration keys regarding the hexdump drawing and provides a callback which then can be passed to the parser.

Identify the package and give the over all version information.

```
136 \ProvidesExplPackage {hexdumptikz-draw} {2026-06-16} {1.0.0}
137 {Common functionality for working with hexdumps and (large) addresses}
```

Load the requirements.

```
138 \RequirePackage { pgfkeys }
139 \RequirePackage { hexdumptikz-common }
140 \RequirePackage { hexdumptikz-selector }
141 \RequirePackage { hexdumptikz-parser }
```

`@@_print_selector_ctx_tl (var.)` Initialize / Declare a selector-context which is used for selecting which rows should be shown in the output.

```
142 \tl_new:N \l_@@_print_selector_ctx_tl
143 \hexdumptikz_selector_ctx_new:N \l_@@_print_selector_ctx_tl
```

`l_@@_print_styled_ctx_tl (var.)` Initialize / Declare a selector-context which is used for assigning *styles* to the bytes.

```
144 \tl_new:N \l_@@_print_styled_ctx_tl
145 \hexdumptikz_selector_ctx_new:N \l_@@_print_styled_ctx_tl
```

3.1 Variables

`\g_@@_addr_index_int (var.)` Variables for counting the node indices
`g_@@_addr_used_index_int (var.)` 146 `\int_new:N \g_@@_addr_index_int`
`\g_@@_byte_index_int (var.)` 147 `\int_new:N \g_@@_addr_used_index_int`
148 `\int_new:N \g_@@_byte_index_int`

`\g_@@_last_addr_node_tl (var.)` Store the nodenames of the most recently generated nodes to be able to create
`\g_@@_last_byte_node_tl (var.)` -end aliases. Due to the nesting of tikz-scopes, these variables must be global.

```
149 \tl_new:N \g_@@_last_addr_node_tl
150 \tl_new:N \g_@@_last_byte_node_tl
```


`\l_@@_cur_addr_tl` (*var.*) Variables for storing the current (padded) address

`\l_@@_cur_addr_padded_str` (*var.*)

```
151 \tl_new:N \l_@@_cur_addr_tl
152 \str_new:N \l_@@_cur_addr_padded_str
```

`\g_@@_first_addr_node_bool` (*var.*) The first address node gets a special style applied. This allows e.g. for placing the whole hexdump at a specific position.

```
153 \bool_new:N \g_@@_first_addr_node_bool
```

3.2 Constants

`\l_@@_tmp_box` (*var.*) this box has the maximal height and depth a hexadecimal node can have

```
154 \box_new:N \l_@@_tmp_box
155 \hbox_set:Nn \l_@@_tmp_box { \ttfamily 0x0123456789abcdef }
```

`\l_@@_tmp_box` (*var.*) variables for uniform node heights

```
156 \dim_const:Nn
157   \c_@@_hexheight_dim
158   { \box_ht:N \l_@@_tmp_box }
159 \dim_const:Nn
160   \c_@@_hexdepth_dim
161   { \box_dp:N \l_@@_tmp_box }
```

3.3 Settings / Config Variables

decisions about address printing

`\draw_show_addr_padded_bool` (*var.*) Whether the address which gets shown shall be padded

```
162 \bool_new:N \l_hexdumptikz_draw_show_addr_padded_bool
```

`\tiktikz_draw_show_addr_bool` (*var.*) Whether the address shall be shown at all in the output

```
163 \bool_new:N \l_hexdumptikz_draw_show_addr_bool
```

`\draw_show_addr_base_bool` (*var.*) Whether the shown address should be prefixed with 0x as indicator of the hexadecimal base

```
164 \bool_new:N \l_hexdumptikz_draw_show_addr_base_bool
```

`\tiktikz_draw_byte_sep_int` (*var.*) insert separating space in the hexdump. Every `int` columns, insert `dim` additional

`\tiktikz_draw_byte_sep_tl` (*var.*) space.

```
165 \int_new:N \l_hexdumptikz_draw_byte_sep_int
166 \tl_new:N \l_hexdumptikz_draw_byte_sep_tl
```

3.3.1 pgfkeys wiring

define explicit keys

```
167 \pgfkeys {
168   /hexdumptikz,
169   % prefix for the nodenames
170   name~prefix/.code = {
171     \tl_set:Nc \l_hexdumptikz_common_nodename_prefix_tl { #1 }
172     \pgfkeys { /hexdumptikz/next~scope/.append~style = { name~prefix = #1 } }
173   },
174   name~prefix = { },
175   name~prefix/.value~required,
176
177   % insert space after that many bytes
178   bytes~sep~after/.code = {
179     \int_set:Nn \l_hexdumptikz_draw_byte_sep_int { \int_abs:n { #1 } }
180   },
181   bytes~sep~after = 64, % basically deactivated
182   bytes~sep~after/.default = { 8 },
183
184   % size of the gap
185   bytes~sep/.code = { \tl_set:Nn \l_hexdumptikz_draw_byte_sep_tl { #1 } },
186   bytes~sep = { 2em },
187   bytes~sep/.value~required,
188
189   % whether to even show the address nodes
190   % this is a bit more complicated as the address-nodes cannot be simply omitted as they are
191   show~addr/.code = {
192     \hexdumptikz_common_parse_bool:Nn
193       \l_hexdumptikz_draw_show_addr_bool
194       { #1 }
195   },
196   show~addr = { true },
197   show~addr/.default = { true },
198
199   % whether to display 0x when showing the addresses
200   show~addr~base/.code = {
201     \hexdumptikz_common_parse_bool:Nn
202       \l_hexdumptikz_draw_show_addr_base_bool
203       { #1 }
204   },
205   show~addr~base = { true },
206   show~addr~base/.default = { true },
207
208   % whether the shown addresses should be padded (otherwise padding only for ranges, node-na
209   show~addr~padded/.code = {
210     \hexdumptikz_common_parse_bool:Nn
211       \l_hexdumptikz_draw_show_addr_padded_bool
212       { #1 }
213   },
214   show~addr~padded = { false },
215   show~addr~padded/.default = { true },
216
217   styled~bytes/.code = {
```

```

218     \hexdumptikz_selector_parse:Nn
219     \l_@@_print_styled_ctx_tl
220     { #1 }
221 }
222 }

```

Define styles which get applied to the corresponding nodes

```

223 \pgfkeys {
224   /hexdumptikz,
225   address~node~first/.style= { },
226   address~node/.style= {
227     left,
228     on~chain,
229     text~height = \c_@@_hexheight_dim,
230     text~depth  = \c_@@_hexdepth_dim,
231   },
232   byte~node/.style = {
233     on~chain,
234     text~height = \c_@@_hexheight_dim,
235     text~depth  = \c_@@_hexdepth_dim,
236   },
237 }

```

3.4 Actual drawing

in principle there are the following node-names:

- one chain going down where every node gets three names:
 1. based on the index in the input
 2. based on the index in the output
 3. based on the address/offset
- three chains in each row for every node-name going right where every node just gets the intuitive name
- every chain comes with an -end node

3.4.1 Helpers for generating specific nodes

`_print_handle_first_node:` (*fn.*) Small helper for handling the *first node* which also modifies the state accordingly.

```

238 \prg_new_conditional:Npnn
239   \@@_print_handle_first_node:
240   { TF }
241 {
242   \bool_if:NTF \g_@@_first_addr_node_bool
243   {
244     \bool_gset_false:N \g_@@_first_addr_node_bool
245     \prg_return_true:
246   } {
247     \prg_return_false:
248   }
249 }

```

`\@@_generate_gap_node:n (fn.)` Generate an empty node which is intended to be a *gap* node (same placement / category like an address node)

Sideeffects:

Arguments:

```
#1   in   number/index of the gap
#-   in   g_@@_addr_index_int
#-   in   l_@@_cur_addr_padded_str
#-   out   g_@@_last_addr_node_tl
```

```
250 \cs_new_protected:Npn \@@_generate_gap_node:n #1
```

```
251 {
```

Construct the nodename only once so it can be reused later

```
252   \tl_gset:Ne
253     \g_@@_last_addr_node_tl
254     { hexdumptikz-in- \int_use:N \g_@@_addr_index_int -gap #1 }
```

Draw the node depending on whether it's the first one (in this hexdump) or

not

```
255   \@@_print_handle_first_node:TF
256   {
257     \node[
258       /hexdumptikz/address~node,
259       /hexdumptikz/address~node~first,
260     ]
261     ( \tl_use:N \g_@@_last_addr_node_tl )
262     { }
263     ;
264   } {
265     \node[
266       /hexdumptikz/address~node,
267     ]
268     ( \tl_use:N \g_@@_last_addr_node_tl )
269     { }
270     ;
271   }
```

Add some node-aliases. This encodes additional information about the node.

```
272   \pgfnodealias
273     { \tl_use:N \l_hexdumptikz_common_nodename_prefix_tl hexdumptikz- \str_use:N \l_@@_cur_a
274     { \tl_use:N \l_hexdumptikz_common_nodename_prefix_tl \tl_use:N \g_@@_last_addr_node_tl }
275   \pgfnodealias
276     { \tl_use:N \l_hexdumptikz_common_nodename_prefix_tl hexdumptikz-out- \int_use:N \g_@@_a
277     { \tl_use:N \l_hexdumptikz_common_nodename_prefix_tl \tl_use:N \g_@@_last_addr_node_tl }
278 }
```

`\@@_generate_addr_node:n (fn.)` Generate a node showing the address / offset

Sideeffects:

Arguments:

```
#1   in   address which shall be shown
#-   in   g_@@_addr_index_int
#-   out   g_@@_last_addr_node_tl
```

```

279 \cs_new_protected:Npn \@@_generate_addr_node:n #1
280 {

```

Construct the nodename only once so it can be reused later

```

281 \tl_gset:Ne
282 \g_@@_last_addr_node_tl
283 { hexdumptikz-in- \int_use:N \g_@@_addr_index_int }

```

Draw the node depending on whether it's the first one (in this hexdump) or

not

```

284 \@@_print_handle_first_node:TF
285 {
286   \node[
287     /hexdumptikz/address~node,
288     /hexdumptikz/address~node~first,
289   ]
290     ( \tl_use:N \g_@@_last_addr_node_tl )
291     { #1 }
292   ;
293 } {
294   \node[
295     /hexdumptikz/address~node,
296   ]
297     ( \tl_use:N \g_@@_last_addr_node_tl )
298     { #1 }
299   ;
300 }
301 }
302 \cs_generate_variant:Nn \@@_generate_addr_node:n { e }

```

`\@@_generate_byte_node:nn` (*fn.*) Generate a node showing a single byte
Sideeffects:

Arguments:

```

#1   in   value of the byte
#2   in   style which is to be applied
#-   in   g_@@_addr_index_int
#-   in   g_@@_byte_index_int
#-   out   g_@@_last_addr_node_tl

```

```

303 \cs_new_protected:Npn \@@_generate_byte_node:nn #1 #2
304 {

```

Construct the nodename only once so it can be reused later

```

305 \tl_gset:Ne
306 \g_@@_last_byte_node_tl
307 {
308   hexdumptikz-in-
309   \int_use:N \g_@@_addr_index_int
310   -
311   \int_use:N \g_@@_byte_index_int
312 }

```

Draw the node depending on its position:

pos=0 needs special positioning

$x \bmod m = 0$ add gap for separation

- normal node

```

313 \int_compare:nNnTF { \g_@@_byte_index_int } = { 0 }
314 {
315   \node[right=of~hexdumptikz-in- \int_use:N \g_@@_addr_index_int ,/hexdumptikz/byte~node,#
316   ( \tl_use:N \g_@@_last_byte_node_tl )
317   { #1 }
318   ;
319 }
320 {
321   \int_compare:nNnTF
322   {
323     \int_mod:nn
324     { \g_@@_byte_index_int }
325     { \l_hexdumptikz_draw_byte_sep_int }
326   }
327   =
328   { 0 }
329   {
330     \node[ xshift = \tl_use:N \l_hexdumptikz_draw_byte_sep_tl ,/hexdumptikz/byte~node,#2]
331     ( \tl_use:N \g_@@_last_byte_node_tl )
332     { #1 }
333     ;
334   } {
335     \node[/hexdumptikz/byte~node,#2]
336     ( \tl_use:N \g_@@_last_byte_node_tl )
337     { #1 }
338     ;
339   }
340 }
341 }
342 \cs_generate_variant:Nn \@@_generate_byte_node:nn { ev }

```

3.4.2 Main drawing functions

`\hexdumptikz_draw_init:nn` (*fn.*) Initialize the drawing logic (necessary as it is stateful)

Sideeffects:

`clobbered` `dir` all kinds of variables, see the source-code

Arguments:

`#1` in pgfkeys for the whole hexdump

`#2` in row-selector

```

343 \cs_new_protected:Npn \hexdumptikz_draw_init:nn #1 #2
344 {

```

Clear and initialize the selector contexts

```

345 \hexdumptikz_selector_parse:Ne
346   \l_@@_print_selector_ctx_tl
347   { #2 }
348 \hexdumptikz_selector_ctx_clear_iter:N
349   \l_@@_print_selector_ctx_tl

```

```

350 \hexdumptikz_selector_ctx_clear_storage:N
351 \l_@@_print_styled_ctx_tl
    Clear the variables with global state.
352 \bool_gset_true:N \g_@@_first_addr_node_bool
353 \int_gzero:N \g_@@_addr_index_int
354 \int_gzero:N \g_@@_addr_used_index_int
    handle the pgfkeys. Needs to be after clearing the print_styled logic.
355 \pgfkeys {
356   /hexdumptikz,
357   #1
358 }
359 \hexdumptikz_selector_ctx_clear_iter:N
360 \l_@@_print_styled_ctx_tl
Start the tikz-scope which holds the entire hexdump
361 \begin{ scope } [
362   start~chain = hexdumptikz-rows~going~{ below=of~\tikzchainprevious.south~east,anchor=nor
363   node~distance = .3ex~and~.2em,
364   font = \ttfamily\small,
365   /hexdumptikz/next~scope,
366 ]
367 }

```

`\hexdumptikz_draw:NNNN` (*fn.*) Draw-callback which is passed to the parser. Draws one line / row of the hexdump.

Sideeffects:

```

clobbered   dir   l_tmpa_int
clobbered   dir   l_tmpa_tl

```

Arguments:

```

#1   in   index of the line (in the input) (int)
#2   in   offset / address (tl)
#3   in   bytes (seq)
#4   out  backchannel for signaling the parser to stop

```

TODO: `\g_@@_addr_index_int` is #1 (ok there are differences regarding empty lines) → simplify

```

368 \cs_new_protected:Npn \hexdumptikz_draw:NNNN #1 #2 #3 #4
369 {

```

Create a working copy of the address, padd it and make a proper string out of it

```

370 \tl_set:Nx \l_@@_cur_addr_tl { #2 }
371 \hexdumptikz_common_pad_address:N #2
372 \str_set:Nx
373   \l_@@_cur_addr_padded_str
374   { \tl_to_str:N #2 }

```

Query the user-supplied row selector whether this row should be present in the output

```

375 \hexdumptikz_selector_match_rows:NnnN
376 \l_@@_print_selector_ctx_tl
377 { \g_@@_addr_index_int } % y
378 { 0 } % x
379 \l_@@_cur_addr_padded_str

```

Only show the row when the selector said so

```

380 \bool_if:cT
381 {
382   \hexdumtikz_selector_show_bool:N
383   \l_@@_print_selector_ctx_tl
384 }
385 {

```

Place the address in a lookup-table which allows translating $\text{addr} \rightarrow \text{idx}$. This may then later used for annotating the hexdump.

```

386   \prop_gput:Nee
387   \g_hexdumtikz_common_cur_offsets_prop
388   { \l_@@_cur_addr_padded_str }
389   { \int_use:N \g_@@_addr_used_index_int }

```

The row-selector might have encountered gaps before the current range \rightarrow draw them

```

390   \int_set_eq:Nc
391   \l_tmpa_int
392   {
393     \hexdumtikz_selector_gap_int:N
394     \l_@@_print_selector_ctx_tl
395   }
396   \int_step_function:nnnN
397   { 1 }
398   { 1 }
399   { \l_tmpa_int }
400   \@@_generate_gap_node:n
401   \int_gzero:c
402   {
403     \hexdumtikz_selector_gap_int:N
404     \l_@@_print_selector_ctx_tl
405   }

```

Determine how the address should be presented (padded or unpadded) (avoids having to use more branches later)

```

406   \bool_if:NTF \l_hexdumtikz_draw_show_addr_padded_bool
407   { \tl_set:Ne \l_tmpa_tl { \l_@@_cur_addr_padded_str } }
408   { \tl_set:Ne \l_tmpa_tl { \l_@@_cur_addr_tl } }

```

Remove the leading 0x if the user does not want it to be present in the output

```

409   \bool_if:NF \l_hexdumtikz_draw_show_addr_base_bool
410   {
411     \regex_replace_once:NnN
412     \c_hexdumtikz_common_leading_hex_base_regex
413     { }
414     \l_tmpa_tl
415   }
416
417   \@@_generate_addr_node:e { \l_tmpa_tl }

```

Create various node-aliases encoding additional information about the node.

```

418   \pgfnodealias
419   { \tl_use:N \l_hexdumtikz_common_nodename_prefix_tl hexdumtikz- \str_use:N \l_@@_
420     { \tl_use:N \l_hexdumtikz_common_nodename_prefix_tl \tl_use:N \g_@@_last_addr_node_tl
421     \pgfnodealias

```



```

422     { \tl_use:N \l_hexdumptikz_common_nodename_prefix_tl hexdumptikz-out- \int_use:N \g_@@
423     { \tl_use:N \l_hexdumptikz_common_nodename_prefix_tl \tl_use:N \g_@@_last_addr_node_tl

```

Start a new tikz-scope + chain for the row which started with the address
containing the byte values

```

424     \begin{ scope } [start~chain=hexdumptikz-row~going~right]
425         \int_zero:N \g_@@_byte_index_int
426         \seq_map_inline:Nn #3
427         {

```

Determine what style to apply to the node

```

428         \hexdumptikz_selector_match_styles:NnnN
429         \l_@@_print_styled_ctx_tl
430         { \g_@@_addr_used_index_int } % y
431         { \g_@@_byte_index_int } % x
432         \l_@@_cur_addr_padded_str
433
434         \@@_generate_byte_node:ev
435         { ##1 }
436         {
437             \hexdumptikz_selector_style_tl:N
438             \l_@@_print_styled_ctx_tl
439         }

```

Create various node-aliases encoding additional information about the node.

```

440         \pgfnodealias
441         { \tl_use:N \l_hexdumptikz_common_nodename_prefix_tl hexdumptikz- \str_use:N \l_@@
442         { \tl_use:N \l_hexdumptikz_common_nodename_prefix_tl \tl_use:N \g_@@_last_byte_node
443         \pgfnodealias
444         { \tl_use:N \l_hexdumptikz_common_nodename_prefix_tl hexdumptikz-out- \int_use:N \
445         { \tl_use:N \l_hexdumptikz_common_nodename_prefix_tl \tl_use:N \g_@@_last_byte_node

```

Count the byte nodes for indexing

```

446         \int_gincr:N \g_@@_byte_index_int
447     }

```

Row finished → create more node-aliases for easier drawing later

```

448     \pgfnodealias
449     { \tl_use:N \l_hexdumptikz_common_nodename_prefix_tl hexdumptikz-in- \int_use:N \g_@@
450     { \tl_use:N \l_hexdumptikz_common_nodename_prefix_tl \tl_use:N \g_@@_last_byte_node
451     \pgfnodealias
452     { \tl_use:N \l_hexdumptikz_common_nodename_prefix_tl hexdumptikz-out- \int_use:N \g_@@
453     { \tl_use:N \l_hexdumptikz_common_nodename_prefix_tl \tl_use:N \g_@@_last_byte_node
454     \pgfnodealias
455     { \tl_use:N \l_hexdumptikz_common_nodename_prefix_tl hexdumptikz- \str_use:N \l_@@_c
456     { \tl_use:N \l_hexdumptikz_common_nodename_prefix_tl \tl_use:N \g_@@_last_byte_node
457     \end{ scope }
458 }

```

Count the selected rows for indexing

```

459     \int_gincr:N \g_@@_addr_used_index_int

```

Forward the row-selector's decision whether the parser should continue parsing

```

460     \bool_set_eq:Nc
461     \l_hexdumptikz_parser_finished_bool
462     {
463         \hexdumptikz_selector_finished_bool:N

```

```

464     \l_@@_print_selector_ctx_tl
465 }
    Count the parsed rows for indexing
466 \int_gincr:N \g_@@_addr_index_int
467 }

```

`\hexdumptikz_draw_fin: (fn.)` Finalize the drawing logic (necessary as it is stateful)

```

468 \cs_new_protected:Npn \hexdumptikz_draw_fin:
469 {
    Hexdump finished → create more node-aliases for easier drawing later
470 \pgfnodealias
471 { \tl_use:N \l_hexdumptikz_common_nodename_prefix_tl hexdumptikz-end }
472 { \tl_use:N \l_hexdumptikz_common_nodename_prefix_tl \g_@@_last_addr_node_tl }
473 \pgfnodealias
474 { \tl_use:N \l_hexdumptikz_common_nodename_prefix_tl hexdumptikz-end-end }
475 { \tl_use:N \l_hexdumptikz_common_nodename_prefix_tl \g_@@_last_byte_node_tl }
476 \end{ scope }
477 \pgfkeys { /hexdumptikz/next~scope/.style = { } }
    Cleanup the created state (additional hexdump might be drawn and the infor-
    mation might leak)
478 \hexdumptikz_selector_ctx_clear_storage:N
479 \l_@@_print_styled_ctx_tl
480 \hexdumptikz_selector_ctx_clear_storage:N
481 \l_@@_print_selector_ctx_tl
482 }

```

3.5 Usual entrance point

`\hexdumptikz_draw_print:nn (fn.)` Usual entry-point handling the complete drawing state logic and the incovation of the selected parser.

Arguments:

#1 in pgfkeys for the whole hexdump
 #2 in row-selector (tl)

```

483 \cs_new_protected:Npn \hexdumptikz_draw_print:nn #1 #2
484 {

```

Note, this handling the pgfkeys can still select a different parser.

```

485 \hexdumptikz_draw_init:nn { #1 } { #2 }
486
487 \hexdumptikz_parser_parse:nN
488 { \l_hexdumptikz_common_input_file_str }
489 \hexdumptikz_draw:NNNN
490
491 \hexdumptikz_draw_fin:
492 }

```

hexdumptikz-addrcalc

Lukas Heindl

v1.0.0 from 2026-06-16

Abstract

Defines helpers and variables which are use throughout the whole project.

Identify the package and give the over all version information.

```
493 \ProvidesExplPackage {hexdumptikz-addrcalc} {2026-06-16} {1.0.0}
494 {Functionality to work / calculate (limited) with larger addresses}
495
496 \RequirePackage { hexdumptikz-common }
```

4.1 Helpers

`\@@_byte_index:n (fn.)` Leaves the index (column) of a byte on the output
Sideeffects:

Arguments:

```
#1 in lower digits of an address
#- in l_hexdumptikz_common_bytes_per_row_int
497 \cs_new:Npn \@@_byte_index:n #1
498 {
499   \int_mod:nn
500   { \int_from_hex:f { #1 } }
501   { \l_hexdumptikz_common_bytes_per_row_int }
502 }
```

`\@@_row_lower:n (fn.)` Leaves the lower part of the address (“y-coord”) on the output. Truncates the input address.
Sideeffects:

Arguments:

```
#1 in lower digits of an address
#- in l_hexdumptikz_common_bytes_per_row_int
503 \cs_new:Npn \@@_row_lower:n #1
504 {
505   \int_to_hex:n
506   {
507     \int_div_truncate:nn
508     { \int_from_hex:f { #1 } }
509     { \l_hexdumptikz_common_bytes_per_row_int }
510     * \l_hexdumptikz_common_bytes_per_row_int
511   }
512 }
```

4.2 pgfkeys

Define pgfkeys with is concerned with address calculations

```
l_@@_hex_digits_calc_int (var.) size of the address-suffix which is required for calculations
513 \int_new:N \l_@@_hex_digits_calc_int

514 \pgfkeys {
515   /hexdumtikz,
516   % amount of bytes in a row -> needs to be enforced during parsing
517   bytes~per~row/.code = {
518     \int_set:Nn \l_hexdumtikz_common_bytes_per_row_int { \int_abs:n { #1 } }
519     \int_set:Nn \l_@@_hex_digits_calc_int
520       % 16 because addresses must be hexadecimal
521       { ( \l_hexdumtikz_common_bytes_per_row_int + 16 - 1 ) / 16 }
522     \int_compare:nNnT
523     { \l_@@_hex_digits_calc_int }
524     >
525     { 30 }
526     {
527       \msg_warn:nnV
528       { hexdumtikz }
529       { too-many-digits }
530       \l_@@_hex_digits_calc_int
531     }
532     \int_case:nnF { \l_hexdumtikz_common_bytes_per_row_int }
533     {
534       { 1 } { }
535       { 2 } { }
536       { 4 } { }
537       { 8 } { }
538       { 16 } { }
539     }
540     {
541       \int_compare:nNnF
542       { \int_mod:nn { \l_hexdumtikz_common_bytes_per_row_int } { 16 } }
543       =
544       { 0 }
545       {
546         \msg_warning:nnV
547         { hexdumtikz }
548         { weird-num-bytes-per-row }
549         \l_hexdumtikz_common_bytes_per_row_int
550       }
551     }
552   },
553   bytes~per~row = { 8 },
554   bytes~per~row/.value~required,
555
556   % amount of digits the addresses are padded to (without the leading 0x)
557   addr~len/.code = {
558     \int_set:Nn \l_hexdumtikz_common_addr_len_int { \int_abs:n { #1 } }
559   },
560   addr~len = { 12 },
561   addr~len/.value~required,
```

```
562 }
```

4.3 Variables

Define variables which are used in the calculation itself. Note these variables are not used for configuration!

```

\l_@@_addr_tl (var.)
\l_@@_addr_prefix_tl (var.) 563 \tl_new:N \l_@@_addr_tl
\l_@@_addr_suffix_tl (var.) 564 \tl_new:N \l_@@_addr_prefix_tl
\l_@@_row_tl (var.) 565 \tl_new:N \l_@@_addr_suffix_tl
566 \tl_new:N \l_@@_row_tl

\l_@@_addr_len_int (var.)
\l_@@_addr_prefix_len_int (var.) 567 \int_new:N \l_@@_addr_len_int % used in calculation != configuration option
\l_@@_byte_index_int (var.) 568 \int_new:N \l_@@_addr_prefix_len_int
569 \int_new:N \l_@@_byte_index_int
```

4.4 Functions

`to_nodename_components:nNN (fn.)` Function which splits an address into its two components (row/offset and column) as it corresponds to the hexdump.

Sideeffects:

```

clobbered   dir  l_@@_addr_tl
clobbered   dir  l_@@_addr_len_int
clobbered   dir  l_@@_addr_prefix_len_int
clobbered   dir  l_@@_addr_prefix_tl
clobbered   dir  l_@@_byte_index_int
clobbered   dir  l_@@_addr_suffix_tl
clobbered   dir  l_@@_row_tl
```

Arguments:

```

#1   in   tl address to work on
#2   out  output macro for the y-component of the coordinate (tl)
#3   out  output macro for the x-component of the coordinate (tl)
#-   in   l_@@_hex.digits.calc_int
```

```

570 \cs_new_protected:Npn \hexdumptikz_address_to_nodename_components:nNN #1 #2 #3
571 {
```

Make a working copy of the address

```

572 \tl_set:Nx \l_@@_addr_tl { #1 }
    validate address

573 \regex_if_match:NVF
574 \c_hexdumptikz_common_hex_regex
575 \l_@@_addr_tl
576 {
577     \msg_critical:nnV
578     { hexdumptikz }
579     { invalid-address }
580     \l_@@_addr_tl
581 }
```

```

    Padd the address to avoid issues regarding the indices
582  \hexdumptikz_common_pad_address:N \l_@@_addr_tl
    strip the leading 0x
583  \regex_replace_once:NnN
584  \c_hexdumptikz_common_leading_hex_base_regex
585  { }
586  \l_@@_addr_tl
    obtain the total length (might be longer than to what was padded)
587  \int_set:Nn \l_@@_addr_len_int
588  { \tl_count:N \l_@@_addr_tl }
    check length for validity to avoid issues later (should never happen due to the
padding)
589  \int_compare:nNnT
590  { \l_@@_addr_len_int }
591  <
592  { \l_@@_hex_digits_calc_int }
593  {
594      \msg_critical:nnVV
595      { hexdumptikz }
596      { address-too-short }
597      \l_@@_addr_tl
598      \l_@@_hex_digits_calc_int
599  }
    calculate the length the prefix should have
600  \int_set:Nn \l_@@_addr_prefix_len_int
601  {
602      \l_@@_addr_len_int
603      -
604      \l_@@_hex_digits_calc_int
605  }
    extract the prefix
606  \tl_set:Ne \l_@@_addr_prefix_tl
607  {
608      \tl_range:Nnn \l_@@_addr_tl
609      { 1 }
610      { \l_@@_addr_prefix_len_int }
611  }
    extract the suffix
612  \tl_set:Ne \l_@@_addr_suffix_tl
613  {
614      \tl_range:Nnn \l_@@_addr_tl
615      { \l_@@_addr_prefix_len_int + 1 }
616      { \l_@@_addr_len_int }
617  }
    obtain the x-coordinate and (part of) the y-coordinate based on the suffix
618  \int_set:Nn \l_@@_byte_index_int
619  {
620      \@@_byte_index:n
621      { \l_@@_addr_suffix_tl }

```

```

622     }
623     \tl_set:Nc \l_@@_row_tl
624     {
625         \@@_row_lower:n
626         { \l_@@_addr_suffix_tl }
627     }

    padd the y-coordinate suffix to avoid issues when concatenating with the prefix
628     \hexdumptikz_common_pad_left:Nnn
629     \l_@@_row_tl
630     { \l_@@_hex_digits_calc_int }
631     { 0 }

    Store the two coord components in the output macros
632     \tl_set:Nc #2
633     {
634         \l_@@_addr_prefix_tl
635         \l_@@_row_tl
636     }
637     \tl_set:Nc #3
638     {
639         \int_to_arabic:n { \l_@@_byte_index_int }
640     }
641 }
642 \cs_generate_variant:Nn \hexdumptikz_address_to_nodename_components:nNN { eNN }

```

`\@@_address_to_nodename:n (fn.)` Convert a full address to its nodename as used in the printed hexdump (leaves the nodename on the output)

Sideeffects:

```

clobbered   dir  l_tmpa_tl
clobbered   dir  l_tmpb_tl

```

Arguments:

#1 in tl address to work on

```

643 \cs_new:Npn \@@_address_to_nodename:n #1
644 {
645     \hexdumptikz_address_to_nodename_components:nNN
646     { #1 }
647     \l_tmpa_tl
648     \l_tmpb_tl
649
650     \l_tmpa_tl
651     \l_tmpb_tl
652 }
653 \cs_generate_variant:Nn \@@_address_to_nodename:n { e }

```

`\@@_address_to_row:n (fn.)` Convert a full address to its row/address-component (leaves the row/address-component on the output)

Sideeffects:

```

clobbered   dir  l_tmpa_tl
clobbered   dir  l_tmpb_tl

```

Arguments:

#1 in tl address to work on

```

654 \cs_new:Npn \@@_address_to_row:n #1
655 {
656   \hexdumptikz_address_to_nodename_components:nNN
657   { #1 }
658   \l_tmpa_tl
659   \l_tmpb_tl
660
661   \l_tmpa_tl
662 }
663 \cs_generate_variant:Nn \@@_address_to_row:n { e }

```

`\@@_address_to_col:n` (*fn.*) Convert a full address to its column-component (leaves the column-component on the output)

Sideeffects:

```

clobbered   dir  l_tmpa_tl
clobbered   dir  l_tmpb_tl

```

Arguments:

#1 in tl address to work on

```

664 \cs_new:Npn \@@_address_to_col:n #1
665 {
666   \hexdumptikz_address_to_nodename_components:nNN
667   { #1 }
668   \l_tmpa_tl
669   \l_tmpb_tl
670
671   \l_tmpb_tl
672 }
673 \cs_generate_variant:Nn \@@_address_to_col:n { e }

```


hexdumptikz-annotate

Lukas Heindl

v1.0.0 from 2026-06-16

Abstract

Functions for annotating a hexdump after it has been drawn.

Identify the package and give the over all version information.

```
674 \ProvidesExplPackage {hexdumptikz-annotate} {2026-06-16} {1.0.0}
675   {Annotate hexdumps}
```

Load the internal libraries which are used

```
676 \RequirePackage { hexdumptikz-addrcalc }
```

`\l_@@_start_row_tl` (*var.*) variables used when labeling a hexdump

```
\l_@@_start_col_tl (var.) 677 \tl_new:N \l_@@_start_row_tl
```

```
\l_@@_end_row_tl (var.) 678 \tl_new:N \l_@@_start_col_tl
```

```
\l_@@_end_col_tl (var.) 679 \tl_new:N \l_@@_end_row_tl
```

```
680 \tl_new:N \l_@@_end_col_tl
```

`\l_@@_start_row_int` (*var.*) variables used for drawing multi-line spanning annotations

```
\l_@@_end_row_int (var.) 681 \int_new:N \l_@@_start_row_int
```

```
\l_@@_middle_n_row_tl (var.) 682 \int_new:N \l_@@_end_row_int
```

```
\l_@@_middle_s_row_tl (var.) 683 \tl_new:N \l_@@_middle_n_row_tl
```

```
684 \tl_new:N \l_@@_middle_s_row_tl
```

`\l_@@_tmpa_tl` (*var.*) Variable to avoid clobbering the usual scratch variables.

```
685 \tl_new:N \l_@@_tmpa_tl
```

`\l_@@_offset_lookup:nnN` (*fn.*) When drawing the hexdump, it stored a mapping from offset to (y-)index. This function allows a lookup in this mapping allowing to “calculate” with the offset as it becomes an index.

Sideeffects:

clobbered `dir` `l_tmpa_int`

Arguments:

#1 `in` key (base-address / y-component)

#2 `in` some context which is printed with the error

#3 `out` int variable to store the (y-)index in

#- `in` `g_hexdumptikz_common_cur_offsets_prop`

```
686 \cs_new_protected:Npn \l_@@_offset_lookup:nnN #1 #2 #3
```

```
687 {
```

```
688   \prop_get:NnNTF \g_hexdumptikz_common_cur_offsets_prop
```

```

689 { #1 }
690 \l_@@_tmpa_tl
691 {
692   \int_set:Nn #3 { \l_@@_tmpa_tl }
693 }
694 {
695   \msg_critical:nnnn
696   { hexdumptikz }
697   { unknown-row }
698   { #1 }
699   { #2 }
700 }
701 }
702 \cs_generate_variant:Nn \@@_offset_lookup:nnN { enN }

```

`\l_@@_label_bytes_init:nnn` (*fn.*) Some common functionalities which are required by multiple annotation functions. Sideeffects:

Arguments:

```

#1   in   pgfkeys for the line
#2   in   start-address
#3   in   end-address
#-   out  \l_@@_start_row_tl
#-   out  \l_@@_start_col_tl
#-   out  \l_@@_end_row_tl
#-   out  \l_@@_end_col_tl

703 \cs_new_protected:Npn \@@_label_bytes_init:nnn #1 #2 #3
704 {
705   \pgfkeys {
706     /hexdumptikz,
707     #1
708   }
709   \hexdumptikz_address_to_nodename_components:nnN
710   { #2 }
711   \l_@@_start_row_tl
712   \l_@@_start_col_tl
713   \hexdumptikz_address_to_nodename_components:nnN
714   { #3 }
715   \l_@@_end_row_tl
716   \l_@@_end_col_tl
717 }

```

`\l_@@_label_bytes:nnn` (*fn.*) Enclose a sequence of bytes with a line.

This does not list sideeffects as there are many variables affected by this. On the other hand, the code is enclosed by a fresh group, so changes are local anyhow.

Arguments:

```

#1   in   pgfkeys for the line
#2   in   start-address
#3   in   end-address

718 \cs_new_protected:Npn \@@_label_bytes:nnn #1 #2 #3
719 {

```

grouping needed to avoid pgfkeys leaking options

```
720 \group_begin:
```

common code across multiple annotation functionalities

```
721 \l_@@_label_bytes_init:nmn { #1 } { #2 } { #3 }
```

check if the annotation spans multiple rows/lines → maybe the drawing can be simplified

```
722 \str_if_eq:VVTf
```

```
723 \l_@@_start_row_tl
```

```
724 \l_@@_end_row_tl
```

```
725 {
```

```
726 \begin{ scope }[/hexdumptikz/next~scope]
```

```
727 \draw
```

```
728 (hexdumptikz- 0x \tl_use:N \l_@@_start_row_tl - \tl_use:N \l_@@_start_col_tl .north~
```

```
729 (hexdumptikz- 0x \tl_use:N \l_@@_start_row_tl - \tl_use:N \l_@@_end_col_tl .north~
```

```
730 (hexdumptikz- 0x \tl_use:N \l_@@_start_row_tl - \tl_use:N \l_@@_end_col_tl .south~
```

```
731 (hexdumptikz- 0x \tl_use:N \l_@@_start_row_tl - \tl_use:N \l_@@_start_col_tl .south~
```

```
732 cycle
```

```
733 ;
```

```
734 \end{ scope }
```

```
735 }
```

```
736 {
```

the annotation spans multiple rows/lines → obtain the y-indices so calculations are possible

```
737 \l_@@_offset_lookup:enN
```

```
738 { 0x \l_@@_start_row_tl }
```

```
739 { start }
```

```
740 \l_@@_start_row_int
```

```
741 \l_@@_offset_lookup:enN
```

```
742 { 0x \l_@@_end_row_tl }
```

```
743 { end }
```

```
744 \l_@@_end_row_int
```

check if there is a “true” middle row. Else use the start-/end-row as replacement

Note two middle rows are important:

1. the topmost middle row `middle_n_row` (*north*) – the row to which a vertical line from the *end-row* extends to
2. the bottommost middle row `middle_s_row` (*south*) – the row to which a vertical line from the *begin-row* extends to

```
745 \int_compare:nNnTF
```

```
746 { \l_@@_start_row_int + 1 }
```

```
747 =
```

```
748 { \l_@@_end_row_int }
```

```
749 {
```

```
750 \tl_set:Nn
```

```
751 \l_@@_middle_n_row_tl
```

```
752 { \int_to_arabic:n { \l_@@_end_row_int } }
```

```
753 \tl_set:Nn
```

```
754 \l_@@_middle_s_row_tl
```

```
755 { \int_to_arabic:n { \l_@@_start_row_int } }
```

```

756     }
757     {
758         \tl_set:Nn
759             \l_@@_middle_n_row_tl
760             { \int_to_arabic:n { \l_@@_start_row_int + 1 } }
761         \tl_set:Nn
762             \l_@@_middle_s_row_tl
763             { \int_to_arabic:n { \l_@@_end_row_int - 1 } }
764     }

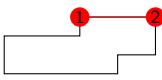
```

Do the actual drawing (starting at north west, going clockwise)

```

765     \begin{ scope }[/hexdumptikz/next~scope]
766         \draw

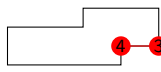
```

top left corner to top right corner of the first line 

```

767         (hexdumptikz- 0x \tl_use:N \l_@@_start_row_tl - \tl_use:N \l_@@_start_col_tl
768         (hexdumptikz- 0x \tl_use:N \l_@@_start_row_tl -end

```

bottom right corner of the lowest fully captured line 

```

769         (hexdumptikz-out- \tl_use:N \l_@@_middle_s_row_tl -end
770         (hexdumptikz-out- \tl_use:N \l_@@_middle_s_row_tl - \tl_use:N \l_@@_end_col_tl

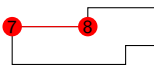
```

bottom right to bottom left corner of the last line 

```

771         (hexdumptikz- 0x \tl_use:N \l_@@_end_row_tl - \tl_use:N \l_@@_end_col_tl
772         (hexdumptikz- 0x \tl_use:N \l_@@_end_row_tl - 0

```

top left corner of the highest fully captured line 

```

773         (hexdumptikz-out- \tl_use:N \l_@@_middle_n_row_tl - 0
774         (hexdumptikz-out- \tl_use:N \l_@@_middle_n_row_tl - \tl_use:N \l_@@_start_col_tl
775         cycle
776         ;
777     \end{ scope }
778 }
779 \group_end:
780 }

```

`\label_bytes_fallback:nnn` (*fn.*) Enclose a sequence of bytes with a line – fallback which also works without the lookup. Thus, this function can also be used once the mapping which was created by printing the hexdump is lost. Though, the output is not as nice as with that mapping available.

This does not list sideeffects as there are many variables affected by this. On the other hand, the code is enclosed by a fresh group, so changes are local anyhow. Arguments:

```

#1 in pgfkeys for the line
#2 in start-address
#3 in end-address

```

```

781 \cs_new_protected:Npn \@@_label_bytes_fallback:nnn #1 #2 #3
782 {

```

grouping needed to avoid pgfkeys leaking options

```
783 \group_begin:
784 \l_@@_label_bytes_init:nmn { #1 } { #2 } { #3 }
```

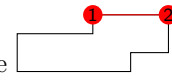
check if the annotation spans multiple rows/lines → maybe the drawing can be simplified

```
785 \str_if_eq:VTF
786 \l_@@_start_row_tl
787 \l_@@_end_row_tl
788 {
789   \begin{ scope }[/hexdumptikz/next~scope]
790     \draw
791       (hexdumptikz- 0x \tl_use:N \l_@@_start_row_tl - \tl_use:N \l_@@_start_col_tl .north~
792       (hexdumptikz- 0x \tl_use:N \l_@@_start_row_tl - \tl_use:N \l_@@_end_col_tl .north~
793       (hexdumptikz- 0x \tl_use:N \l_@@_start_row_tl - \tl_use:N \l_@@_end_col_tl .south~
794       (hexdumptikz- 0x \tl_use:N \l_@@_start_row_tl - \tl_use:N \l_@@_start_col_tl .south~
795       cycle
796       ;
797   \end{ scope }
798 }
799 {
```

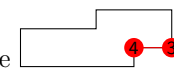
this procedure does not rely on being able to translate addresses to row-indices
Thus, this can serve as fallback if the prop required for the lookup is not available anymore
Though, the spacing of the drawn border is not as nice as with the correct handling of the middle line

draws the line beginning at the top left clockwise

```
800 \begin{ scope }[/hexdumptikz/next~scope]
801 \draw
```

top left corner to top right corner of the first line 

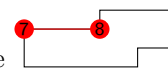
```
802 (hexdumptikz- 0x \tl_use:N \l_@@_start_row_tl - \tl_use:N \l_@@_start_col_tl .north~
803 (hexdumptikz- 0x \tl_use:N \l_@@_start_row_tl -end .north~
```

bottom right corner of the lowest fully captured line 

```
804 (hexdumptikz- 0x \tl_use:N \l_@@_end_row_tl -end.north~east) -- % noqa: S103
805 (hexdumptikz- 0x \tl_use:N \l_@@_end_row_tl - \tl_use:N \l_@@_end_col_tl .north~
```

bottom right to bottom left corner of the last line 

```
806 (hexdumptikz- 0x \tl_use:N \l_@@_end_row_tl - \tl_use:N \l_@@_end_col_tl .south~
807 (hexdumptikz- 0x \tl_use:N \l_@@_end_row_tl - 0 .south~
```

top left corner of the highest fully captured line 

```
808 (hexdumptikz- 0x \tl_use:N \l_@@_start_row_tl - 0 .south~
809 (hexdumptikz- 0x \tl_use:N \l_@@_start_row_tl - \tl_use:N \l_@@_start_col_tl .south~
810 cycle
811 ;
812 \end{ scope }
813 }
814 \group_end:
815 }
```

hexdumptikz-parser

Lukas Heindl

v1.0.0 from 2026-06-16

Abstract

General variables and helpers for parsing hexdump data from file. Also includes some documentation about the general parsing architecture.

Generally, the idea is as follows. The user (indirectly) calls the respective parser and passes a callback along. This callback is then responsible for drawing a line of the hexdump.

6.1 Interfaces

Callback:

Arguments:

- #1 in line-index
- #2 in parsed offset/address
- #3 in parsed bytes (seq)
- #4 out variable to signal whether printing / the selection has finished

Parser:

Arguments:

- #1 in filename which shall be parsed
- #2 in callback

6.2 Implementation

Identify the package and give the over all version information.

```
816 \ProvidesExplPackage {hexdumptikz-parser} {2026-06-16} {1.0.0}
817   {Printing and annotating hexdumps with TikZ}
818
819 \RequirePackage { hexdumptikz-parser-hd }
820
```

`\l_hexdumptikz_parser_ior` (*var.*) IO variable for opening the file

```
821 \ior_new:N \l_hexdumptikz_parser_ior
```

`\dumptikz_parser_line_int` (*var.*) Counter for the lines in the input

```
822 \int_new:N \l_hexdumptikz_parser_line_int
```

`\dumptikz_parser_offset_tl` (*var.*) Parsed offset / the address

```
823 \tl_new:N \l_hexdumptikz_parser_offset_tl
```

`hexdumptikz_parser_line_tl (var.)` Complete parsed line
824 `\tl_new:N \l_hexdumptikz_parser_line_tl`

`hexdumptikz_parser_bytes_seq (var.)` Sequence of the parsed bytes
825 `\seq_new:N \l_hexdumptikz_parser_bytes_seq`

`hexdumptikz_parser_last_line_seen_bool (var.)` Whether the parser assumes the last line has already been processed (might be useful for some stricter input validation tests)
826 `\bool_new:N \l_hexdumptikz_parser_last_line_seen_bool`

`hexdumptikz_parser_finished_bool (var.)` Variable which is passed to the callback as notification channel in the other direction
827 `\bool_new:N \l_hexdumptikz_parser_finished_bool`

Variables configuring the parsers.

`hexdumptikz_parser_strict_byte_num_bool (var.)` Whether the number of bytes should be checked and an error being raised if there is a mismatch with the expected number.
828 `\bool_new:N \l_hexdumptikz_parser_strict_byte_num_bool`

`hexdumptikz_parser_strict_hex_bool (var.)` Whether to check for valid hex characters in the byte input.
829 `\bool_new:N \l_hexdumptikz_parser_strict_hex_bool`

`hexdumptikz_parser_leading_base_bool (var.)` Whether the value specification includes the base (0x)
830 `\bool_new:N \l_hexdumptikz_parser_leading_base_bool`

`hexdumptikz_parser_parse:nN (fn.)` An alias to the active parser.
831 `\cs_new_eq:NN \hexdumptikz_parser_parse:nN \hexdumptikz_parser_hd:nN`
832 `\cs_generate_variant:Nn \str_range_ignore_spaces:nnn { Vnn }`

`hexdumptikz_parser_addr_regex (var.)` Matches an address specifier with an optional leading 0x and an optional trailing `..`. Extracts the raw address in its first capturing group.
833 `\regex_const:Nn`
834 `\c_hexdumptikz_parser_addr_regex`
835 `{ \A \s* (?:0x)? ([0-9A-Fa-f]+) \s* :? \s* }`

`hexdumptikz_parser_leading_hex_byte_regex (var.)` Matches a byte in its hexadecimal representation (2 hex-digits).
836 `\regex_const:Nn`
837 `\c_hexdumptikz_parser_leading_hex_byte_regex`
838 `{ \A [0-9A-Fa-f] { 2 } }`

6.2.1 Pgfkeys

parser (*pgfkey*)

```

839 \pgfkeys {
840   /hexdumptikz,
841   parser/.is~choice,
842   parser/hd/.code = {
843     \cs_set_eq:NN \hexdumptikz_parser_parse:nN \hexdumptikz_parser_hd:nN
844   }
845 }
```

Define interface to simplify setting parser options.

```

846 \pgfkeys {
847   /hexdumptikz/parser~opts~hd/.code={
848     \pgfkeys{
849       /hexdumptikz/parser~opts/hd,
850       #1
851     }
852   },
853 }
```

Define parser options

```

854 \pgfkeys {
855   /hexdumptikz/parser~opts/hd/.is~family,
856   /hexdumptikz/parser~opts/hd,
```

ct byte number per row (*pgfkey*)

```

857   strict~byte~number~per~row/.code={
858     \hexdumptikz_common_parse_bool:Nn
859     \l_hexdumptikz_parser_strict_byte_num_bool
860     { #1 }
861   },
862   strict~byte~number~per~row/.default={true},
```

strict hex digits (*pgfkey*)

```

863   strict~hex~digits/.code={
864     \hexdumptikz_common_parse_bool:Nn
865     \l_hexdumptikz_parser_strict_hex_bool
866     { #1 }
867   },
868   strict~hex~digits/.default={true},
```

leading value base (*pgfkey*)

```

869   leading~value~base/.code={
870     \hexdumptikz_common_parse_bool:Nn
871     \l_hexdumptikz_parser_leading_base_bool
872     { #1 }
873   },
874   leading~value~base/.default={true},
875 }
```


6.2.2 Errors

no-valid-offset (*error*)

```
876 \msg_new:nnn { hexdumptikz-parser } { no-valid-offset }
877 { No~valid~offset/address~found~in~line~'#1' }
```

too-many-bytes (*error*)

```
878 \msg_new:nnn { hexdumptikz-parser } { too-many-bytes }
879 { Line~with~offset~#1~has~more~than~#2~bytes.~Left~over~is:~#3 }
```

invalid-hex-digits (*error*)

```
880 \msg_new:nnn { hexdumptikz-parser } { invalid-hex-digits }
881 { The~next~bytes~on~the~following~line~are~no~valid~hex~bytes:~#1 }
```

weird-byte-count (*error*)

```
882 \msg_new:nnn { hexdumptikz-parser } { weird-byte-count }
883 {
884   A~line~with~less~than~'bytes~per~row'~
885   (#1)~
886   bytes~is~only~allowed~as~last~line.~Line~at~offset:~#2
887 }
```

leading-base-missing (*error*)

```
888 \msg_new:nnn { hexdumptikz-parser } { leading-base-missing }
889 { Expected~leading~'0x'~missing~in~line~#1 }
```

hexdumptikz-parser-hd

Lukas Heindl

v1.0.0 from 2026-06-16

Abstract

Parse files with classical hexdump format such as the one produced by `od`, `hd` or *Wireshark*.

Identify the package and give the over all version information.

```
890 \ProvidesExplPackage {hexdumptikz-parser-hd} {2026-06-16} {1.0.0}
891   {Printing and annotating hexdumps with TikZ}
```

Load dependencies

```
892 \RequirePackage { hexdumptikz-common }
```

7.1 Public

`\hexdumptikz_parser_hd:nN` (*fn.*) Public entrance function for parsing a hexdump

Sideeffects:

```
clobbered   dir  l_hexdumptikz_parser_ior
clobbered   dir  l_hexdumptikz_parser_line_tl
clobbered   dir  l_hexdumptikz_parser_line_int
clobbered   dir  l_hexdumptikz_parser_offset_tl
clobbered   dir  l_hexdumptikz_parser_bytes_seq
clobbered   dir  l_hexdumptikz_parser_finished_bool
```

Arguments:

```
#1  in  filename/-path to parse
#2  in  callback function
```

```
893 \cs_new_protected:Npn \hexdumptikz_parser_hd:nN #1 #2
894 {
```

Initialization

```
895   \bool_set_true:N \l_hexdumptikz_parser_finished_bool
896   \int_zero:N      \l_hexdumptikz_parser_line_int
```

Open the file for reading

```
897   \ior_open:Nn \l_hexdumptikz_parser_ior { #1 }
```

Iterate over the lines of the input file

```
898   \ior_map_variable:NNn
899   \l_hexdumptikz_parser_ior
900   \l_hexdumptikz_parser_line_tl
901   {
902     \tl_set:Ne \l_hexdumptikz_parser_line_tl
903     { \tl_trim_spaces:e { \l_hexdumptikz_parser_line_tl } }
```

```

Silently ignore empty lines
904     \tl_if_blank:nF { \l_hexdumptikz_parser_line_tl }
905     {
        Do the actual parsing
906         \@@_normalize_line:NNN
907         \l_hexdumptikz_parser_line_tl
908         \l_hexdumptikz_parser_offset_tl
909         \l_hexdumptikz_parser_bytes_seq
        Pass the parsed data to the callback
910         #2
911         \l_hexdumptikz_parser_line_int
912         \l_hexdumptikz_parser_offset_tl
913         \l_hexdumptikz_parser_bytes_seq
914         \l_hexdumptikz_parser_finished_bool
        Enable the callback to stop the parsing.
915         \bool_if:NT \l_hexdumptikz_parser_finished_bool
916         { \ior_map_break: }
917     }
        Count all lines including the empty ones.
918     \int_incr:N \l_hexdumptikz_parser_line_int
919 }
        Close the input file again
920 \ior_close:N \l_hexdumptikz_parser_ior
921 }

```

7.2 Helpers

`\@@_normalize_line:NNN` (*fn.*) Normalize a single parsed line (tl) from the slightly different input formats supported by this parser to a unified sequence.

Sideeffects:

`clobbered` `dir` `l_tmpa_seq`

Arguments:

#1 `in` parsed line (tl)

#2 `out` offset (tl)

#3 `out` bytes (seq)

```

922 \cs_new_protected:Npn \@@_normalize_line:NNN #1 #2 #3
923 {

```

Extract the offset / address from the start of the line

```

924   \regex_extract_once:NVNF
925   \c_hexdumptikz_parser_addr_regex
926   #1
927   \l_tmpa_seq
928   {
929     \msg_critical:nnV
930     { hexdumptikz-parser }
931     { no-valid-offset }
932     #1
933   }

```

The regex matched without the 0x (in order to make the prefix optional) → add it here for normalization purposes

```
934 \tl_set:Ne #2
935 { 0x \seq_item:Nn \l_tmpa_seq { 2 } }
```

Remove the leading offset / address so the hex-digits forming the “values” are at the start of the string

```
936 \regex_replace_once:NnN
937 \c_hexdumtikz_parser_addr_regex
938 { }
939 #1
```

Convert the series of hex-digits to a sequence of bytes

```
940 \@@_hexcompact_to_seq:NNN
941 #1
942 #3
943 #2
944 }
```

`\@@_hexcompact_to_seq:NNN (fn.)` Convert/Sanitize a series of hex digits to a sequence of bytes

Sideeffects:

clobbered dir l_tmpa_str

Arguments:

#1 in series of hex-digits (tl)
 #2 out sequence (tl)
 #3 in offset (only used to generate nicer error messages which indicate the location of the error)
 #- in l_hexdumtikz_parser_strict_byte_num_bool
 #- in l_hexdumtikz_common_bytes_per_row_int
 #- in l_hexdumtikz_parser_leading_base_bool
 #- in/out l_hexdumtikz_parser_last_line_seen_bool
 #- in l_hexdumtikz_parser_strict_hex_bool

```
945 \cs_new_protected:Npn \@@_hexcompact_to_seq:NNN #1 #2 #3
946 {
```

First some optional checks and transformations:

Check the number of parsed bytes

```
947 \bool_if:NT \l_hexdumtikz_parser_strict_byte_num_bool
948 {
949   \bool_if:NT \l_hexdumtikz_parser_last_line_seen_bool {
950     \msg_critical:nneV
951     { hexdumtikz-parser }
952     { weird-byte-count }
953     { \int_use:N \l_hexdumtikz_common_bytes_per_row_int }
954     #3
955   }
956 }
```

remove a leading 0x indicating the hexadecimal base once

```
957 \bool_if:NT \l_hexdumtikz_parser_leading_base_bool
958 {
959   \regex_replace_once:NnNF
960   \c_hexdumtikz_common_leading_hex_base_regex
961   { }
```

```

962     #1
963     {
964         \msg_critical:nnV
965         { hexdumptikz-parser }
966         { leading-base-missing }
967         #1
968     }
969 }

```

First clean up and so some initialization

```

970 \seq_clear:N #2
971 \str_set:Ne \l_tmpa_str { #1 }

```

Loop over the string and remove the parsed bytes step by step in the process.

```

972 \bool_while_do:nn
973 { ! \str_if_empty_p:N \l_tmpa_str }
974 {

```

Check the number of bytes parsed from the current line.

Either throw a critical error on violation or just silently stop the parsing of this line. Silently stopping the parsing is needed for rudimentary support of the *canonical* format which also shows the ASCII representation at the end of each line. The support is not full though, as this fails if the last row/line is not fully populated (since this parser is very liberal in terms of where and how many spaces can be present).

```

975 \int_compare:nNnT { \l_hexdumptikz_common_bytes_per_row_int } > { 0 }
976 {
977     \int_compare:nNnF
978     { \seq_count:N #2 }
979     <
980     { \l_hexdumptikz_common_bytes_per_row_int }
981     {
982         \bool_if:NT \l_hexdumptikz_parser_strict_byte_num_bool
983         {
984             \msg_critical:nnVeV
985             { hexdumptikz-parser }
986             { too-many-bytes }
987             #3
988             { \int_use:N \l_hexdumptikz_common_bytes_per_row_int }
989             \l_tmpa_str
990         }

```

basically stop gracefully parsing the current line

```

991     \str_set:Nn \l_tmpa_str { }
992 }
993 }
994 \str_if_empty:NF \l_tmpa_str
995 {

```

optionally check for valid hex characters. In principle there is no issue if non-hex characters are present as the content is not interpreted.

```

996 \bool_if:NT \l_hexdumptikz_parser_strict_hex_bool
997 {
998     \regex_if_match:NVF
999     \c_hexdumptikz_parser_leading_hex_byte_regex

```

```

1000     \l_tmpa_str
1001     {
1002         \msg_critical:nnV
1003         { hexdumptikz-parser }
1004         { invalid-hex-digits }
1005         \l_tmpa_str
1006     }
1007 }

    actual core parsing logic
1008     \seq_put_right:Ne #2
1009     {
1010         \str_range_ignore_spaces:Vnn \l_tmpa_str { 1 } { 2 }
1011     }
1012     \str_set:Ne
1013     \l_tmpa_str
1014     { \str_range_ignore_spaces:Vnn \l_tmpa_str { 3 } { -1 } }
1015 }
1016 }

keep track if we expect this is the last line in the input (the last line normally is
the only line which is not fully populated)
1017 \int_compare:nNnT
1018 { \seq_count:N #2 }
1019 <
1020 { \l_hexdumptikz_common_bytes_per_row_int }
1021 {
1022     \bool_set_true:N \l_hexdumptikz_parser_last_line_seen_bool
1023 }
1024 }

```

`\hexdumptikz_parser_dbg:NNNN` (*fn.*) Dummy callback which just prints its arguments and can be used for debugging. Sideeffects:

Arguments:

```

#1  in   line-index
#2  in   parsed offset/address
#3  in   parsed bytes (seq)
#4  out  variable to signal whether printing / the selection has finished

```

```

1025 \cs_new_protected:Npn \hexdumptikz_parser_dbg:NNNN #1 #2 #3 #4
1026 {
1027     \iow_term:x { line idx:~\int_use:N #1 }
1028     \iow_term:x { offset:~~~\tl_use:N #2 }
1029     \iow_term:x { bytes:~~~~\seq_use:Nn #3 {~|~} }
1030     \iow_term:x { }
1031 }

```

`\hexdumptikzParserDbg` Short macro which can aid in debugging.

```

1032 \NewDocumentCommand { \hexdumptikzParserDbg } { m } {
1033     \hexdumptikz_parser_hd:nN
1034     { #1 }
1035     \hexdumptikz_parser_dbg:NNNN
1036 }

```

hexdumptikz-selector

Lukas Heindl

v1.0.0 from 2026-06-16

Abstract

General utilities for implementing the selection logic. Each instantiation of a selector is associated with a *context* which stores the state of the selector. Among other functionalities, this (sub-)package provides tools to work with these contexts.

Identify the package and give the over all version information.

```
1037 \ProvidesExplPackage {hexdumptikz-selector} {2026-06-16} {1.0.0}
1038 {Parse and work with address/hexdump selectors}
```

Requirements

```
1039 \RequirePackage { hexdumptikz-common }
```

8.1 Errors

Define the error messages associated with the selector mechanism.

invalid-segment-input (*error*)

```
1040 \msg_new:nnn { hexdumptikz-selector } { invalid-segment-input }
1041 { '#1'~is~no~valid~input~for~a~segment. }
```

invalid-addr-input (*error*)

```
1042 \msg_new:nnn { hexdumptikz-selector } { invalid-addr-input }
1043 { '#1'~is~no~valid~address.~An~address~is~for~example~0x014af1. }
```

invalid-address (*error*)

```
1044 \msg_new:nnn { hexdumptikz-selector } { invalid-address }
1045 { '#1'~is~no~valid~address. }
```

invalid-coordinate (*error*)

```
1046 \msg_new:nnn { hexdumptikz-selector } { invalid-coordinate }
1047 { '#1'~is~no~valid~coordinate. }
```

invalid-idx-input (*error*)

```
1048 \msg_new:nnn { hexdumptikz-selector } { invalid-idx-input }
1049 { '#1'~is~no~valid~index.~An~index~must~only~consist~of~digits~0-9. }
```

unknown-predicate (*error*)

```
1050 \msg_new:nnn { hexdumptikz-selector } { unknown-predicate }
1051 { '#1'~is~not~a~valid~predicate. }
```

```

invalid-predicate-axis (error)
1052 \msg_new:nnn { hexdumptikz-selector } { invalid-predicate-axis }
1053 { '#1'~is~not~a~valid~axis~for~a~predicate. }

invalid-predicate-input (error)
1054 \msg_new:nnn { hexdumptikz-selector } { invalid-predicate-input }
1055 { '#1'~is~not~a~valid~predicate~specification. }

invalid-rule (error)
1056 \msg_new:nnn { hexdumptikz-selector } { invalid-rule }
1057 { '#1'~is~not~a~valid~rule. }

weird-mod (error)
1058 \msg_new:nnn { hexdumptikz-selector } { weird-mod }
1059 { with~mod~'#1'~a~remainder~of~'#2'~does~not~make~much~sense. }

invalid-mod (error)
1060 \msg_new:nnn { hexdumptikz-selector } { invalid-mod }
1061 {
1062   With~'#1'~bytes~per~row~(BPR),~mod~'#2'~is~not~valid.~
1063   Mod~must~be~able~to~work~locally~on~the~row.~
1064   BPR~\%~mod~==~0~must~hold.
1065 }

```

8.2 Context Helpers

A context stores its data in *global* variables with the context-id encoded in the variable name.

`\g_@@_ctx_id_int (var.)` Track the index of the next context which is to be allocated

```

1066 \int_new:N \g_@@_ctx_id_int

```

`\@@_ctx_var:nn (fn.)` Construct the variable name specific to a context. Leaves the variable name in the output.
Sideeffects:

Arguments:

#1 in context-id
#2 in name of the variable in the context

```

1067 \cs_new:Npn \@@_ctx_var:nn #1 #2
1068 {
1069   g_@@_ctx_#1_#2
1070 }

```

`\hexdumptikz_selector_ctx_new:N (fn.)` Allocate a new context.
Sideeffects:

Arguments:

#1 out context-id which should be used in the future

```

1071 \cs_new_protected:Npn \hexdumptikz_selector_ctx_new:N #1
1072 {

```


obtain a new context-id and increment the counter

```
1073 \int_gincr:N \g_@@_ctx_id_int
1074 \tl_set:Ne #1 { \int_use:N \g_@@_ctx_id_int }
```

variables to store the parsed rules

```
1075 \seq_new:c { \@@_ctx_var:nn { #1 } { kind_seq } }
1076 \seq_new:c { \@@_ctx_var:nn { #1 } { addr_low_seq } }
1077 \seq_new:c { \@@_ctx_var:nn { #1 } { addr_high_seq } }
1078 \seq_new:c { \@@_ctx_var:nn { #1 } { idx_low_x_seq } }
1079 \seq_new:c { \@@_ctx_var:nn { #1 } { idx_low_y_seq } }
1080 \seq_new:c { \@@_ctx_var:nn { #1 } { idx_high_x_seq } }
1081 \seq_new:c { \@@_ctx_var:nn { #1 } { idx_high_y_seq } }
1082 \seq_new:c { \@@_ctx_var:nn { #1 } { style_seq } }
1083 \seq_new:c { \@@_ctx_var:nn { #1 } { pred_idx_seq } }
```

variables to store the predicates (pred_idx_seq may point to these sequences)

```
1084 \seq_new:c { \@@_ctx_var:nn { #1 } { pred_kind_seq } }
1085 \seq_new:c { \@@_ctx_var:nn { #1 } { pred_axis_seq } }
1086 \seq_new:c { \@@_ctx_var:nn { #1 } { pred_mod_seq } }
1087 \seq_new:c { \@@_ctx_var:nn { #1 } { pred_res_seq } }
```

variables to for iterating over the rules

```
1088 \seq_new:c { \@@_ctx_var:nn { #1 } { active_rules_seq } }
1089 \int_new:c { \@@_ctx_var:nn { #1 } { next_rule_int } }
```

variables which store the iterator result – think of it as yield

```
1090 \int_new:c { \@@_ctx_var:nn { #1 } { gap_cnt_int } }
1091 \tl_new:c { \@@_ctx_var:nn { #1 } { style_tl } }
1092 \bool_new:c { \@@_ctx_var:nn { #1 } { show_bool } }
1093 \bool_new:c { \@@_ctx_var:nn { #1 } { finished_bool } }
1094 }
```

`selector_ctx_clear_storage:N (fn.)` Reset/Clear the storage of a context.

Sideeffects:

Arguments:

#1 in context-id on which to work on

```
1095 \cs_new_protected:Npn \hexdumtikz_selector_ctx_clear_storage:N #1
1096 {
```

Clear the rule storage

```
1097 \seq_gclear:c { \@@_ctx_var:nn { #1 } { kind_seq } }
1098 \seq_gclear:c { \@@_ctx_var:nn { #1 } { addr_low_seq } }
1099 \seq_gclear:c { \@@_ctx_var:nn { #1 } { addr_high_seq } }
1100 \seq_gclear:c { \@@_ctx_var:nn { #1 } { idx_low_x_seq } }
1101 \seq_gclear:c { \@@_ctx_var:nn { #1 } { idx_low_y_seq } }
1102 \seq_gclear:c { \@@_ctx_var:nn { #1 } { idx_high_x_seq } }
1103 \seq_gclear:c { \@@_ctx_var:nn { #1 } { idx_high_y_seq } }
1104 \seq_gclear:c { \@@_ctx_var:nn { #1 } { style_seq } }
1105 \seq_gclear:c { \@@_ctx_var:nn { #1 } { pred_idx_seq } }
```

Clear the predicate storage

```
1106 \seq_gclear:c { \@@_ctx_var:nn { #1 } { pred_kind_seq } }
1107 \seq_gclear:c { \@@_ctx_var:nn { #1 } { pred_axis_seq } }
```

```

1108 \seq_gclear:c { \@@_ctx_var:nn { #1 } { pred_mod_seq } }
1109 \seq_gclear:c { \@@_ctx_var:nn { #1 } { pred_res_seq } }
1110 }

```

`\tikz_selector_ctx_clear_iter:N` (*fn.*) Reset/Clear the iterator state.
Sideeffects:

Arguments:

#1 in context-id on which to work on

```

1111 \cs_new_protected:Npn \hexdumtikz_selector_ctx_clear_iter:N #1
1112 {
1113 \seq_gclear:c { \@@_ctx_var:nn { #1 } { active_rules_seq } } % noqa: S103
1114 \int_gset:cn { \@@_ctx_var:nn { #1 } { next_rule_int } } { 1 } % noqa: S103
1115 }

```

`\tikz_selector_ctx_clear_iter_out:N` (*fn.*) Reset/Clear the iterator output.
Sideeffects:

Arguments:

#1 in context-id on which to work on

```

1116 \cs_new_protected:Npn \hexdumtikz_selector_ctx_clear_iter_out:N #1
1117 {
1118 \tl_gclear:c { \@@_ctx_var:nn { #1 } { style_tl } }
1119 \bool_gset_false:c { \@@_ctx_var:nn { #1 } { show_bool } }
1120 \bool_gset_false:c { \@@_ctx_var:nn { #1 } { finished_bool } } % noqa: S103
1121 }

```

`\tikz_selector_show_bool:N` (*fn.*) Obtain the show_bool variable.
Sideeffects:

Arguments:

#1 in context-id on which to work on

```

1122 \cs_new_protected:Npn \hexdumtikz_selector_show_bool:N #1
1123 {
1124 \@@_ctx_var:nn { #1 } { show_bool }
1125 }

```

`\tikz_selector_finished_bool:N` (*fn.*) Obtain the finished_bool variable.
Sideeffects:

Arguments:

#1 in context-id on which to work on

```

1126 \cs_new_protected:Npn \hexdumtikz_selector_finished_bool:N #1
1127 {
1128 \@@_ctx_var:nn { #1 } { finished_bool }
1129 }

```

`\hexdumptikz_selector_gap_int:N (fn.)` Obtain the `gap_cnt_int` variable.

Sideeffects:

Arguments:

`#1 in` context-id on which to work on

```
1130 \cs_new_protected:Npn \hexdumptikz_selector_gap_int:N #1
1131 {
1132   \@@_ctx_var:nn { #1 } { gap_cnt_int }
1133 }
```

`\hexdumptikz_selector_style_tl:N (fn.)` Obtain the `style_tl` variable.

Sideeffects:

Arguments:

`#1 in` context-id on which to work on

```
1134 \cs_new_protected:Npn \hexdumptikz_selector_style_tl:N #1
1135 {
1136   \@@_ctx_var:nn { #1 } { style_tl }
1137 }
```

8.3 Debugging Helpers

Expl3 functions and latex macros for inspecting a context.

TODO: only available in special debug package(s)

`\@@_ctx_dump_storage:N (fn.)` Dump the storage part of the context

Sideeffects:

Arguments:

`#1 in` context-id on which to inspect

```
1138 \cs_new_protected:Npn \@@_ctx_dump_storage:N #1
1139 {
1140   \iow_term:x { =====~Context~#1~Storage~===== }
```

Dump the part that stores the rules

```
1141 \@@_ctx_dump_seq:Nn #1 { kind_seq }
1142 \@@_ctx_dump_seq:Nn #1 { addr_low_seq }
1143 \@@_ctx_dump_seq:Nn #1 { addr_high_seq }
1144 \@@_ctx_dump_seq:Nn #1 { idx_low_x_seq }
1145 \@@_ctx_dump_seq:Nn #1 { idx_low_y_seq }
1146 \@@_ctx_dump_seq:Nn #1 { idx_high_x_seq }
1147 \@@_ctx_dump_seq:Nn #1 { idx_high_y_seq }
1148 \@@_ctx_dump_seq:Nn #1 { style_seq }
1149 \@@_ctx_dump_seq:Nn #1 { pred_idx_seq }
```

Newline for separation

```
1150 \iow_term:n { }
```

Dump the part that stores the predicates

```
1151 \@@_ctx_dump_seq:Nn #1 { pred_kind_seq }
1152 \@@_ctx_dump_seq:Nn #1 { pred_axis_seq }
1153 \@@_ctx_dump_seq:Nn #1 { pred_mod_seq }
```

```

1154 \@@_ctx_dump_seq:Nn #1 { pred_res_seq }
1155 \iow_term:n { ===== }
1156 }

```

\@@_ctx_dump_iter:N (*fn.*) Dump the iterator state of the context
Sideeffects:

Arguments:

#1 in context-id on which to inspect

```

1157 \cs_new_protected:Npn \@@_ctx_dump_iter:N #1
1158 {
1159 \iow_term:x { =====~Context~#1~Iter~===== }
1160 \@@_ctx_dump_seq:Nn #1 { active_rules_seq }
1161 \@@_ctx_dump_int:Nn #1 { next_rule_int }
1162 \@@_ctx_dump_int:Nn #1 { gap_cnt_int }
1163 \@@_ctx_dump_tl:Nn #1 { style_tl }
1164 \@@_ctx_dump_bool:Nn #1 { show_bool }
1165 \@@_ctx_dump_bool:Nn #1 { finished_bool }
1166 \iow_term:n { ===== }
1167 }

```

\@@_ctx_dump_seq:Nn (*fn.*) Helper for dumping a sequence
Sideeffects:

Arguments:

#1 in context-id on which to inspect

#2 in (seq) variable to dump

```

1168 \cs_new_protected:Npn \@@_ctx_dump_seq:Nn #1 #2
1169 {
1170 \iow_term:x
1171 {
1172 #2 ~ = ~
1173 \seq_use:cn
1174 { \@@_ctx_var:nn { #1 } { #2 } }
1175 { ~|~ }
1176 }
1177 }

```

\@@_ctx_dump_int:Nn (*fn.*) Helper for dumping an integer
Sideeffects:

Arguments:

#1 in context-id on which to inspect

#2 in (int) variable to dump

```

1178 \cs_new_protected:Npn \@@_ctx_dump_int:Nn #1 #2
1179 {
1180 \iow_term:x
1181 {
1182 #2 ~ = ~
1183 \int_use:c

```

```

1184 { \@@_ctx_var:nn { #1 } { #2 } }
1185 }
1186 }

```

`\@@_ctx_dump_tl:Nn` (*fn.*) Helper for dumping a token list
Sideeffects:

Arguments:

#1 in context-id on which to inspect
#2 in (tl) variable to dump

```

1187 \cs_new_protected:Npn \@@_ctx_dump_tl:Nn #1 #2
1188 {
1189   \iow_term:x
1190   {
1191     #2 ~ = ~
1192     \tl_use:c
1193     { \@@_ctx_var:nn { #1 } { #2 } }
1194   }
1195 }

```

`\@@_ctx_dump_bool:Nn` (*fn.*) Helper for dumping a boolean
Sideeffects:

Arguments:

#1 in context-id on which to inspect
#2 in (bool) variable to dump

```

1196 \cs_new_protected:Npn \@@_ctx_dump_bool:Nn #1 #2
1197 {
1198   \iow_term:x
1199   {
1200     #2 ~ = ~
1201     \bool_to_str:c
1202     { \@@_ctx_var:nn { #1 } { #2 } }
1203   }
1204 }

```

`\l_@@_dbg_ctx_tl` (*var.*) Context which is used by the debugging latex macros

```

1205 \tl_new:N \l_@@_dbg_ctx_tl
1206 \hexdumptikz_selector_ctx_new:N \l_@@_dbg_ctx_tl

```

`\hexdumptikzParseRow` Parse the specified input as selector into the debugging context.

```

1207 \NewDocumentCommand { \hexdumptikzParseRow } { m } {
1208   \hexdumptikz_selector_parse:Nn \l_@@_dbg_ctx_tl { #1 }
1209 }

```

`\hexdumptikzParseDump` Dump the debugging context to the console.

```

1210 \NewDocumentCommand { \hexdumptikzParseDump } { } {
1211   \@@_ctx_dump_storage:N \l_@@_dbg_ctx_tl
1212 }

```

`\l_@@_dbg_tmpa_str (var.)` Match the specified $\{\langle addr \rangle\}$, $\{\langle x-idx \rangle\}$, $\{\langle y-idx \rangle\}$ against the rules stored in the debugging context. Matches as row-selection matcher (simplified matcher).

```
\l_@@_dbg_tmpb_int (var.) 1213 \str_new:N \l_@@_dbg_tmpa_str
\hexdumptikzMatchRow 1214 \int_new:N \l_@@_dbg_tmpa_int
1215 \int_new:N \l_@@_dbg_tmpb_int
1216 \NewDocumentCommand { \hexdumptikzMatchRow } { mmm } {
1217   \str_set:Ne \l_@@_dbg_tmpa_str { #1 }
1218   \int_set:Nn \l_@@_dbg_tmpa_int { #2 }
1219   \int_set:Nn \l_@@_dbg_tmpb_int { #3 }
1220
1221   \hexdumptikz_selector_match_rows:NnnN
1222   \l_@@_dbg_ctx_tl
1223   { \l_@@_dbg_tmpa_int }
1224   { \l_@@_dbg_tmpb_int }
1225   \l_@@_dbg_tmpa_str
1226 }
```

`\hexdumptikzReset` Clear/Reset the iterstate of the debugging context

```
1227 \NewDocumentCommand { \hexdumptikzReset } { } {
1228   \hexdumptikz_selector_ctx_clear_iter:N \l_@@_dbg_ctx_tl
1229 }
```

`\hexdumptikzIterDump` Dump the iterstate of the debugging context.

```
1230 \NewDocumentCommand { \hexdumptikzIterDump } { } {
1231   @@_ctx_dump_iter:N \l_@@_dbg_ctx_tl
1232 }
```

`\hexdumptikzMatchStyle` Match the specified $\{\langle addr \rangle\}$, $\{\langle x-idx \rangle\}$, $\{\langle y-idx \rangle\}$ against the rules stored in the debugging context. Matches as styling matcher.

```
1233 \NewDocumentCommand { \hexdumptikzMatchStyle } { mmm } {
1234   \str_set:Ne \l_@@_dbg_tmpa_str { #1 }
1235   \int_set:Nn \l_@@_dbg_tmpa_int { #2 }
1236   \int_set:Nn \l_@@_dbg_tmpb_int { #3 }
1237
1238   \hexdumptikz_selector_match_styles:NnnN
1239   \l_@@_dbg_ctx_tl
1240   { \l_@@_dbg_tmpa_int }
1241   { \l_@@_dbg_tmpb_int }
1242   \l_@@_dbg_tmpa_str
1243 }
```

TODO: not debugging related anymore Load the specific packages in the end as their initialization depends on the utils from this (sub-)package being present.

```
1244 \RequirePackage { hexdumptikz-selector-parser }
1245 \RequirePackage { hexdumptikz-selector-matcher }
```

hexdumptikz-selector-parser

Lukas Heindl

v1.0.0 from 2026-06-16

Abstract

Parser for the selection logic. Parses an (user supplied) “string” to a *context* which can be queried later.

Identify the package and give the over all version information.

```
1246 \ProvidesExplPackage {hexdumptikz-selector-parser} {2026-06-16} {1.0.0}
1247 {Parse address/hexdump selectors}
```

Load internal dependencies

```
1248 \RequirePackage { hexdumptikz-addrcalc }
```

Create a shortcut for obtaining the variable name of a variable in a context.
Only for being able to use @@ in the *.dtx* file.

```
1249 \cs_new_eq:NN
1250 \@@_ctx_var:nn
1251 \_hexdumptikz_selector_ctx_var:nn
```

9.1 Variables

Generally, the parsers parse the input to (local) variables. Then later, the information from these variables gets collectively pushed to the context.

Predicate related variables

`\l_@@_pred_kind_tl` (*var.*) Type of the predicate
1252 \tl_new:N \l_@@_pred_kind_tl

`\l_@@_pred_axis_tl` (*var.*) On which axis the predicate operates on (x or y)?
1253 \tl_new:N \l_@@_pred_axis_tl

`\l_@@_pred_mod_int` (*var.*) For modulo predicates: what is the modulo ($x/y \bmod \mathbf{m} = r$) of this rule
1254 \int_new:N \l_@@_pred_mod_int

`\l_@@_pred_res_int` (*var.*) For modulo predicates: what is the residue ($x/y \bmod m = \mathbf{r}$) of this rule
1255 \int_new:N \l_@@_pred_res_int

`\l_@@_kind_tl` (*var.*) Type/Kind of the current rule
1256 \tl_new:N \l_@@_kind_tl

`\l_@@_low_tl (var.)` Tokenlist-low/high part of the current rule (typically the address)
`\l_@@_high_tl (var.)` 1257 `\tl_new:N \l_@@_low_tl`
1258 `\tl_new:N \l_@@_high_tl`

`\l_@@_low_x_int (var.)` Integer low/high x/y part of the current rule (typically the index)
`\l_@@_low_y_int (var.)` 1259 `\int_new:N \l_@@_low_x_int`
`\l_@@_high_x_int (var.)` 1260 `\int_new:N \l_@@_low_y_int`
`\l_@@_high_y_int (var.)` 1261 `\int_new:N \l_@@_high_x_int`
1262 `\int_new:N \l_@@_high_y_int`

`\l_@@_gap_count_int (var.)` Amount of gap-nodes to insert before the current rule takes effect
1263 `\int_new:N \l_@@_gap_count_int`

`\l_@@_style_tl (var.)` Style which to apply when this rule matches
1264 `\tl_new:N \l_@@_style_tl`

`\l_@@_pred_idx_int (var.)` Which predicate to check when this rule is in range (index in the context's predicates sequence)
1265 `\int_new:N \l_@@_pred_idx_int`

`\l_@@_tmpa_tl (var.)` Scratch-like variable:
1266 `\tl_new:N \l_@@_tmpa_tl`

9.2 Regexes

`\c_@@_mod_regex (var.)` Rexex for parsing modulo predicates. Also extracts the arguments with capture groups.
1267 `\regex_const:Nn`
1268 `\c_@@_mod_regex`
1269 `{ \A mod \(\s* ([0-9]+) \s* , \s* ([0-9]+) \s* \) \Z }`

9.3 Parsers for individual components

`\@@_parse_pred:n (fn.)` Parse a predicate. Yes it is not nice to force parsing into (local) variables, but that many argument would also be not so nice
Sideeffects:
`clobbered dir l_tmpa_tl`
`clobbered dir l_tmpa_seq`
Arguments:
`#1 in tl variable to parse`
`#- out l_@@_pred_axis_tl`
`#- out l_@@_pred_kind_tl`
`#- out l_@@_pred_mod_tl`
`#- out l_@@_pred_res_tl`
`#- in l_hexdumptikz_common_bytes_per_row_int`
1270 `\cs_new_protected:Npn \@@_parse_pred:n #1`
1271 `{`


```

1272 \tl_set:Ne \l_tmpa_tl { \tl_trim_spaces:n {#1} }
1273 \tl_if_empty:NF \l_tmpa_tl
1274 {
1275   \seq_set_split:Nne \l_tmpa_seq { / } { \l_tmpa_tl }
1276   \int_case:nnF { \seq_count:N \l_tmpa_seq }
1277   {
1278     { 1 }
1279     {

```

No axis prefix → default axis = x

```

1280       \tl_set:Nn \l_@@_pred_axis_tl { x }
1281       \tl_set:Ne \l_tmpa_tl { \seq_item:Nn \l_tmpa_seq { 1 } }
1282     }
1283     { 2 }
1284     {
1285       \tl_set:Ne
1286         \l_@@_pred_axis_tl
1287         { \seq_item:Nn \l_tmpa_seq { 1 } }
1288       \tl_set:Ne
1289         \l_tmpa_tl
1290         { \seq_item:Nn \l_tmpa_seq { 2 } }
1291       \str_case:VnF \l_@@_pred_axis_tl
1292       {
1293         { x } { }
1294         { y } { }
1295       }
1296       {
1297         \msg_critical:nnn
1298         { hexdumptikz-selector }
1299         { invalid-predicate-axis }
1300         { #1 }
1301       }
1302     }
1303   }
1304   {
1305     \msg_critical:nnn
1306     { hexdumptikz-selector }
1307     { invalid-predicate-input }
1308     { #1 }
1309   }

```

Parse the right-hand-side

```

1310 \str_case:VnF \l_tmpa_tl
1311 {
1312   { odd }
1313   {
1314     \tl_set:Nn \l_@@_pred_kind_tl { mod }
1315     \int_set:Nn \l_@@_pred_mod_int { 2 }
1316     \int_set:Nn \l_@@_pred_res_int { 1 }
1317   }
1318   { even }
1319   {
1320     \tl_set:Nn \l_@@_pred_kind_tl { mod }
1321     \int_set:Nn \l_@@_pred_mod_int { 2 }
1322     \int_set:Nn \l_@@_pred_res_int { 0 }

```

```

1323     }
1324   }
1325   {
      Else assume it is plain  $mod(s,i)$ 
1326     \regex_extract_once:NVNTF
1327     \c_@@_mod_regex
1328     \l_tmpa_tl
1329     \l_tmpa_seq
1330     {
1331       \tl_set:Nn \l_@@_pred_kind_tl { mod }
1332       \int_set:Nn \l_@@_pred_mod_int { \seq_item:Nn \l_tmpa_seq { 2 } } % noqa: S103
1333       \int_set:Nn \l_@@_pred_res_int { \seq_item:Nn \l_tmpa_seq { 3 } } % noqa: S103
      Enforce additional constraints when the predicate is about the x-axis
1334       \tl_if_eq:NnT \l_@@_pred_axis_tl { x }
1335       {
1336         \int_compare:nNnF
1337         {
1338           \int_mod:nn
1339           { \l_hexdumptikz_common_bytes_per_row_int }
1340           { \l_@@_pred_mod_int }
1341         }
1342         =
1343         { 0 }
1344         {
1345           \msg_critical:nnVV
1346           { hexdumptikz-selector }
1347           { invalid-mod }
1348           \l_hexdumptikz_common_bytes_per_row_int
1349           \l_@@_pred_mod_int
1350         }
1351       }
      Sanity check on the modulus arguments
1352       \int_compare:nNnF
1353       { \l_@@_pred_res_int }
1354       <
1355       { \l_@@_pred_mod_int }
1356       {
1357         \msg_warning:nnVV
1358         { hexdumptikz-selector }
1359         { weird-mod }
1360         \l_@@_pred_mod_int
1361         \l_@@_pred_res_int
1362       }
1363     }
1364   {
1365     \msg_critical:nnn
1366     { hexdumptikz-selector }
1367     { unknown-predicate }
1368     { #1 }
1369   }
1370 }
1371 }

```

```

1372 }
1373 \cs_generate_variant:Nn \@@_parse_pred:n { V }

```

`\@@_parse_coord:NNN` (*fn.*) Parse a coordinate (*y* and optionally also *x*).

Sideeffects:

`clobbered` `dir` `l_tmpa_seq`

Arguments:

`#1` `in` `tl` variable to parse
`#2` `in` parsed *x*-coordinate (zero if unset)
`#3` `in` parsed *y*-coordinate

```

1374 \cs_new_protected:Npn \@@_parse_coord:NNN #1 #2 #3
1375 {
1376   \seq_set_split:NnV \l_tmpa_seq { / } #1
1377   \int_case:nnF { \seq_count:N \l_tmpa_seq }
1378   {
1379     { 1 }
1380     {
1381       \int_zero:N #2
1382       \int_set:Nn #3 { \seq_item:Nn \l_tmpa_seq { 1 } }
1383     }
1384     { 2 }
1385     {
1386       \int_set:Nn #2 { \seq_item:Nn \l_tmpa_seq { 2 } }
1387       \int_set:Nn #3 { \seq_item:Nn \l_tmpa_seq { 1 } }
1388     }
1389   }
1390   {
1391     \msg_critical:nnN
1392     { hexdumptikz-selector }
1393     { invalid-coordinate }
1394     #1
1395   }
1396 }

```

`\@@_parse_addr:NNN` (*fn.*) Parse an address.

Sideeffects:

`clobbered` `dir` `l_tmpa_seq`
`clobbered` `dir` `l_@@_tmpa_int`

Arguments:

`#1` `in` `tl` variable to parse
`#2` `in` parsed *x*-coordinate (int)
`#3` `in` parsed address (*tl*)

```

1397 \cs_new_protected:Npn \@@_parse_addr:NNN #1 #2 #3
1398 {
1399   \hexdumptikz_address_to_nodename_components:eNN
1400   { #1 }
1401   #3
1402   \l_@@_tmpa_tl
1403   \tl_put_left:Nn #3 { 0x }
1404   \int_set:Nn #2 { \l_@@_tmpa_tl }
1405 }

```

\@@_parse_core:n (fn.) Parses the whole range (without style or predicate)

Sideeffects:

```
clobbered   dir  l_tmpa_seq
clobbered   dir  l_tmpa_tl
clobbered   dir  l_tmpb_tl
```

Arguments:

```
#1   in   tl variable to parse
#-   out  l_@@_low_tl
#-   out  l_@@_high_tl
#-   out  l_@@_low_x_int
#-   out  l_@@_low_y_int
#-   out  l_@@_high_x_int
#-   out  l_@@_high_y_int
#-   out  l_@@_kind_tl
#-   out  l_@@_gap_count_int
```

1406 \cs_new_protected:Npn \@@_parse_core:n #1

1407 {

Clear all variables which potentially are not reset to avoid leaking data from the previous parse

```
1408 \tl_clear:N \l_@@_low_tl
1409 \tl_clear:N \l_@@_high_tl
1410 \int_zero:N \l_@@_low_x_int
1411 \int_zero:N \l_@@_low_y_int
1412 \int_zero:N \l_@@_high_x_int
1413 \int_zero:N \l_@@_high_y_int
```

Some fast-path checks

```
1414 \tl_set:Nx \l_tmpa_tl { \tl_trim_spaces:n {#1} }
1415 \tl_if_eq:NnTF \l_tmpa_tl { x }
1416 {
1417   \tl_set:Nn \l_@@_kind_tl { gap }
1418   \int_set:Nn \l_@@_gap_count_int { 1 }
1419 }
```

Start actual parsing/splitting

```
1420 {
1421   \seq_set_split:NnV \l_tmpa_seq { - } \l_tmpa_tl
1422   \int_compare:nNnTF { \seq_count:N \l_tmpa_seq } = { 1 }
1423   {
```

Atom

```
1424     \tl_set:Nx \l_tmpa_tl { \seq_item:Nn \l_tmpa_seq { 1 } }
1425     \regex_if_match:NVTf \c_hexdumtikz_common_hex_x_regex \l_tmpa_tl
1426     {
1427       \tl_set:Nn \l_@@_kind_tl { addr }
1428       \@@_parse_addr:NNN
1429       \l_tmpa_tl
1430       \l_@@_low_x_int
1431       \l_@@_low_tl
1432       \tl_set_eq:NN
1433       \l_@@_high_tl
1434       \l_@@_low_tl
1435       \int_set_eq:NN
```

```

1436         \l_@@_high_x_int
1437         \l_@@_low_x_int
1438     }
1439     {
1440         \regex_if_match:NVTF \c_hexdumptikz_common_idx_regex \l_tmpa_tl
1441     {
1442         \tl_set:Nn \l_@@_kind_tl { idx }
1443         \@@_parse_coord:NNN
1444             \l_tmpa_tl
1445             \l_@@_low_x_int
1446             \l_@@_low_y_int
1447         \int_set_eq:NN
1448             \l_@@_high_x_int
1449             \l_@@_low_x_int
1450         \int_set_eq:NN
1451             \l_@@_high_y_int
1452             \l_@@_low_y_int
1453     }
1454     {
1455         \msg_critical:nnV
1456         { hexdumptikz-selector }
1457         { invalid-segment-input }
1458         \l_tmpa_tl
1459     }
1460 }
1461 }
1462 {

```

Range

```

1463     \tl_set:Ne \l_tmpa_tl { \seq_item:Nn \l_tmpa_seq { 1 } }
1464     \tl_set:Ne \l_tmpb_tl { \seq_item:Nn \l_tmpa_seq { 2 } }
1465     \regex_if_match:NVTF \c_hexdumptikz_common_hex_x_regex \l_tmpa_tl
1466     {

```

Address range

```

1467         \regex_if_match:NVF \c_hexdumptikz_common_hex_x_regex \l_tmpb_tl
1468     {
1469         \msg_critical:nnV
1470         { hexdumptikz-selector }
1471         { invalid-addr-input }
1472         \l_tmpb_tl
1473     }
1474     \tl_set:Nn \l_@@_kind_tl { addr }
1475     \@@_parse_addr:NNN
1476         \l_tmpa_tl
1477         \l_@@_low_x_int
1478         \l_@@_low_tl
1479     \@@_parse_addr:NNN
1480         \l_tmpb_tl
1481         \l_@@_high_x_int
1482         \l_@@_high_tl
1483     }
1484     {

```

Check if its really a valid index range

```

1485     \regex_if_match:NVF \c_hexdumptikz_common_idx_regex \l_tmpa_tl
1486     {
1487         \msg_critical:nnV
1488         { hexdumptikz-selector }
1489         { invalid-segment-input }
1490         \l_tmpa_tl
1491     }
1492     \regex_if_match:NVF \c_hexdumptikz_common_idx_regex \l_tmpb_tl
1493     {
1494         \msg_critical:nnV
1495         { hexdumptikz-selector }
1496         { invalid-idx-input }
1497         \l_tmpb_tl
1498     }
Index range
1499     \tl_set:Nn \l_@@_kind_tl { idx }
1500     \@@_parse_coord:NNN
1501     \l_tmpa_tl
1502     \l_@@_low_x_int
1503     \l_@@_low_y_int
1504     \@@_parse_coord:NNN
1505     \l_tmpb_tl
1506     \l_@@_high_x_int
1507     \l_@@_high_y_int
1508 }
1509 }
1510 }
1511 }
1512 \cs_generate_variant:Nn \@@_parse_core:n { V }

```

9.4 Pushing to the context

`\@@_push_rule:N (fn.)` Push the parsed range / rule to the context
Sideeffects:

Arguments:

```

#1   in  context to push to
#-   in  l_@@_kind_tl
#-   in  l_@@_low_tl
#-   in  l_@@_high_tl
#-   in  l_@@_low_x_int
#-   in  l_@@_low_y_int
#-   in  l_@@_high_x_int
#-   in  l_@@_high_y_int
#-   in  l_@@_style_tl
#-   in  l_@@_pred_idx_tl

```

```

1513 \cs_new_protected:Npn \@@_push_rule:N #1
1514 {

```

rule storage

```

1515     \seq_gput_right:cV
1516     { \@@_ctx_var:nn { #1 } { kind_seq } }

```

```

1517   \l_@@_kind_tl
1518   \seq_gput_right:cV
1519     { \@@_ctx_var:nn { #1 } { addr_low_seq } }
1520     \l_@@_low_tl
1521   \seq_gput_right:cV
1522     { \@@_ctx_var:nn { #1 } { addr_high_seq } }
1523     \l_@@_high_tl
1524   \seq_gput_right:cV
1525     { \@@_ctx_var:nn { #1 } { idx_low_x_seq } }
1526     \l_@@_low_x_int
1527   \seq_gput_right:cV
1528     { \@@_ctx_var:nn { #1 } { idx_low_y_seq } }
1529     \l_@@_low_y_int
1530   \seq_gput_right:cV
1531     { \@@_ctx_var:nn { #1 } { idx_high_x_seq } }
1532     \l_@@_high_x_int
1533   \seq_gput_right:cV
1534     { \@@_ctx_var:nn { #1 } { idx_high_y_seq } }
1535     \l_@@_high_y_int
1536   \seq_gput_right:cV
1537     { \@@_ctx_var:nn { #1 } { style_seq } }
1538     \l_@@_style_tl
1539   \seq_gput_right:cV
1540     { \@@_ctx_var:nn { #1 } { pred_idx_seq } }
1541     \l_@@_pred_idx_int
1542 }

```

\@@_push_pred:N (*fn.*) Push the parsed predicate to the context
Sideeffects:

Arguments:

```

#1   in  context to push to
#-   in  l_@@_pred_kind_tl
#-   in  l_@@_pred_axis_tl
#-   in  l_@@_pred_mod_int
#-   in  l_@@_pred_res_int

```

```

1543 \cs_new_protected:Npn \@@_push_pred:N #1
1544 {

```

pred storage

```

1545   \seq_gput_right:cV
1546     { \@@_ctx_var:nn { #1 } { pred_kind_seq } }
1547     \l_@@_pred_kind_tl
1548   \seq_gput_right:cV
1549     { \@@_ctx_var:nn { #1 } { pred_axis_seq } }
1550     \l_@@_pred_axis_tl
1551   \seq_gput_right:cV
1552     { \@@_ctx_var:nn { #1 } { pred_mod_seq } }
1553     \l_@@_pred_mod_int
1554   \seq_gput_right:cV
1555     { \@@_ctx_var:nn { #1 } { pred_res_seq } }
1556     \l_@@_pred_res_int
1557 }

```

9.5 Public

`hexdumptikz_selector_parse:Nn (fn.)` Parses a complete (c)list of rules.

Sideeffects:

```
clobbered   dir  l_tmpa_seq
clobbered   dir  l_tmpb_seq
clobbered   dir  l_tmpa_tl
clobbered   dir  l_tmpb_tl
```

Arguments:

```
#1  in  context to push to
#2  in  tl to parse as comma separated list of rules
```

```
1558 \cs_new_protected:Npn \hexdumptikz_selector_parse:Nn #1 #2
1559 {
```

First empty the context in terms of storage

```
1560 \hexdumptikz_selector_ctx_clear_storage:N #1
```

Iterate over the whole (c)list of rules

```
1561 \clist_map_inline:nn { #2 }
1562 {
1563   \tl_set:N \l_tmpa_tl { \tl_trim_spaces:n {##1} }
1564   \tl_clear:N \l_@@_style_tl
```

Split off the style part if present

```
1565   \seq_set_split:NnV \l_tmpa_seq { -> } \l_tmpa_tl
1566   \int_compare:nNnT { \seq_count:N \l_tmpa_seq } > { 1 }
1567   {
1568     \tl_set:N \l_@@_style_tl
1569     \l_@@_style_tl
1570     { \seq_item:Nn \l_tmpa_seq { 2 } }
1571   }
```

Handle the rest

```
1572   \tl_set:N \l_tmpa_tl { \seq_item:Nn \l_tmpa_seq { 1 } }
1573   \tl_if_empty:NF \l_tmpa_tl
1574   {
1575     \tl_replace_all:Nee \l_tmpa_tl
1576     { \char_set_catcode_active:N | }
1577     { \char_set_catcode_other:N | }
1578   }
1579   \seq_set_split:Nne \l_tmpb_seq { | } { \l_tmpa_tl }
1580   \int_compare:nNnTF { \seq_count:N \l_tmpb_seq } = { 1 }
1581   {
```

no predicate present

```
1582     \tl_set:N \l_tmpa_tl { \seq_item:Nn \l_tmpb_seq { 1 } }
1583     \int_zero:N \l_@@_pred_idx_int
1584     \@@_parse_core:V \l_tmpa_tl
1585     \@@_push_rule:N #1
1586   }
1587   {
```

one predicate present

parse and push the predicate

```
1588   \tl_set:N \l_tmpa_tl { \seq_item:Nn \l_tmpb_seq { 2 } }
1589   \@@_parse_pred:V \l_tmpa_tl
```



```

1590      \@@_push_pred:N #1
      obtain the index of the just pushed predicate which needs to be referenced in
      the current rule.
1591      \int_set:Nn
1592      \l_@@_pred_idx_int
1593      {
1594      \seq_count:c
1595      {
1596      \@@_ctx_var:nn
1597      { #1 }
1598      { pred_kind_seq }
1599      }
1600      }
      parse and push the rule
1601      \tl_set:Nc \l_tmpa_tl { \seq_item:Nn \l_tmpb_seq { 1 } }
1602      \@@_parse_core:V \l_tmpa_tl
1603      \@@_push_rule:N #1
1604    }
1605  }
1606 }
1607 \cs_generate_variant:Nn \hexdumtikz_selector_parse:Nn { Ne }

```

hexdumptikz-selector-matcher

Lukas Heindl

v1.0.0 from 2026-06-16

Abstract

Matches parsed selector/style rules against coordinates and/or addresses.

Identify the package and give the over all version information.

```
1608 \ProvidesExplPackage {hexdumptikz-selector-matcher} {2026-06-16} {1.0.0}
1609 {Match coordinates and addresses with speficed display rules}
```

Create a shortcut for obtaining the variable name of a variable in a context.
Only for being able to use @@ in the .dtx file.

```
1610 \cs_new_eq:NN
1611   \@@_ctx_var:nn
1612   \__hexdumptikz_selector_ctx_var:nn
```

10.1 Variables

`\l_@@_range_state_int` (*var.*) Functions checking whether an element is inside a given range need to return more than just true/false. Thus, such functions use this (local) variable to return the decision. It basically is an enum which works as follows:

- 1 range has already passed
- +0 range is currently active
- +1 range is still ahead / did not start yet

```
1613 \int_new:N \l_@@_range_state_int
```

`\l_@@_rule_id_int` (*var.*) Scratch variable for storing the rule-id which is currently processed. (avoids clobbering other scratch variables and makes the code more readable)

```
1614 \int_new:N \l_@@_rule_id_int
```

`\l_@@_search_all_bool` (*var.*) Decision variable whether all active rules should be searched. Allows to specify the matcher should stop after finding the first active rule.

Is used as a scratch variable in the public matcher functions (which then gets passed to the util-functions).

```
1615 \bool_new:N \l_@@_search_all_bool
```

`\l_@@_match_loop_bool` (*var.*) Whether to continue searching for a match.

```
1616 \bool_new:N \l_@@_match_loop_bool
```

`\l_@@_active_tmp_seq` (*var.*) Scratch variable for temporarily storing the set of active rules.

```
1617 \seq_new:N \l_@@_active_tmp_seq
```

10.2 Helpers

`\@@_match_mod:nnn` (*fn.*) Check a modulus-rule. Such a rule checks $x \bmod m \stackrel{!}{=} r$
Sideeffects:

Arguments:

#1 in item / **x**
#2 in modulus / **m**
#3 in residue / **r**

```
1618 \cs_generate_variant:Nn \int_show:n { e }
1619 \cs_generate_variant:Nn \int_mod:nn { en }
1620 \prg_new_protected_conditional:Npnn
1621   \@@_match_mod:nnn
1622   #1 #2 #3
1623 { TF }
1624 {
1625   \int_compare:nNnTF { \int_mod:nn { #1 } { #2 } } = { #3 }
1626   { \prg_return_true: }
1627   { \prg_return_false: }
1628 }
```

`\@@_match_apply_style:Nn` (*fn.*) Applies/Appends the style of a specific rule to the current iterator state.
Sideeffects:

Arguments:

#1 in context-id
#2 in rule-id

```
1629 \cs_new_protected:Npn \@@_match_apply_style:Nn #1 #2
1630 {
1631   \tl_if_empty:cTF
1632   { \@@_ctx_var:nn { #1 } { style_tl } }
1633   {
1634     \tl_gset:ce
1635     { \@@_ctx_var:nn { #1 } { style_tl } }
1636     {
1637       \seq_item:cn
1638       { \@@_ctx_var:nn { #1 } { style_seq } }
1639       { #2 }
1640     }
1641   }
1642   {
1643     \tl_gput_right:ce
1644     { \@@_ctx_var:nn { #1 } { style_tl } }
1645     {
1646       ,
1647       \seq_item:cn
1648       { \@@_ctx_var:nn { #1 } { style_seq } }
1649       { #2 }
1650     }
1651   }
1652 }
```

10.3 Predicate Checking

`match_rule_predicate:NnnNn` (*fn.*) Check if the rule matches with respect to the predicate. Does not check the range.

Sideeffects:

`clobbered dir l_tmpa_int`

`clobbered dir l_tmpa_t1`

Arguments:

`#1 in context-id`

`#2 in current y`

`#3 in current x`

`#4 in current address (currently unused)`

`#5 in rule-id`

1653 `\prg_new_protected_conditional:Npnn`

1654 `\@@_match_rule_predicate:NnnNn`

1655 `#1 #2 #3 #4 #5`

1656 `{ T, TF }`

1657 `{`

1658 `\int_set:Nn`

1659 `\l_tmpa_int`

1660 `{`

1661 `\seq_item:cn`

1662 `{ \@@_ctx_var:nn { #1 } { pred_idx_seq } }`

1663 `{ #5 }`

1664 `}`

1665

1666 `\int_compare:nNnTF { \l_tmpa_int } = { 0 }`

1667 `{`

no predicate present \rightarrow matches

1668 `\prg_return_true:`

1669 `}`

1670 `{`

1671 `\tl_set:Ne`

1672 `\l_tmpa_t1`

1673 `{`

1674 `\seq_item:cV`

1675 `{ \@@_ctx_var:nn { #1 } { pred_kind_seq } }`

1676 `\l_tmpa_int`

1677 `}`

1678

1679 `\str_case:VnF \l_tmpa_t1`

1680 `{`

1681 `{ mod }`

1682 `{`

1683 `\tl_set:Ne`

1684 `\l_tmpa_t1`

1685 `{`

1686 `\seq_item:cn`

1687 `{ \@@_ctx_var:nn { #1 } { pred_axis_seq } } % noqa: S103`

1688 `{ \l_tmpa_int }`

1689 `}`

1690 `\int_set:Nn`

1691 `\l_tmpb_int`

1692 `{`

```

1693         \seq_item:cn
1694         { \@@_ctx_var:nn { #1 } { pred_mod_seq } } % noqa: S103
1695         { \l_tmpa_int }
1696     }
1697     \int_set:Nn
1698     \l_tmpa_int
1699     {
1700         \seq_item:cn
1701         { \@@_ctx_var:nn { #1 } { pred_res_seq } } % noqa: S103
1702         { \l_tmpa_int }
1703     }
1704
1705     \str_case:VnF \l_tmpa_tl
1706     {
1707         { x }
1708         {
1709             \@@_match_mod:nnnTF
1710             { #3 }
1711             { \l_tmpb_int }
1712             { \l_tmpa_int }
1713             { \prg_return_true: }
1714             { \prg_return_false: }
1715         }
1716         { y } {
1717             \@@_match_mod:nnnTF
1718             { #2 }
1719             { \l_tmpb_int }
1720             { \l_tmpa_int }
1721             { \prg_return_true: }
1722             { \prg_return_false: }
1723         }
1724     }
1725     {
1726         \prg_return_false:
1727     }
1728 }
1729 }
1730 {
1731     \msg_critical:nnV
1732     { hexdumptikz-selector }
1733     { unknown-predicate }
1734     \l_tmpa_tl
1735 }
1736 }
1737 }

```

10.4 Coordinate/Address Checking

`\@@_coord_lt:nnnn` (*fn.*) checks whether $a < b$ holds
Sideeffects:

Arguments:

```

#1 in ya
#2 in xa
#3 in yb
#4 in xb

1738 \prg_new_protected_conditional:Npnn
1739 \@@_coord_lt:nnnn
1740 #1 #2 #3 #4
1741 { TF }
1742 {
1743   \int_compare:nNnTF { #1 } < { #3 }
1744   { \prg_return_true: }
1745   {
1746     \int_compare:nNnTF { #1 } > { #3 }
1747     { \prg_return_false: }
1748     {
1749       \int_compare:nNnTF { #2 } < { #4 }
1750       { \prg_return_true: }
1751       { \prg_return_false: }
1752     }
1753   }
1754 }

```

`\@@_coord_gt:nnnn` (*fn.*) checks whether $a > b$ holds
Sideeffects:

Arguments:

```

#1 in ya
#2 in xa
#3 in yb
#4 in xb

1755 \prg_new_protected_conditional:Npnn
1756 \@@_coord_gt:nnnn
1757 #1 #2 #3 #4
1758 { TF }
1759 {
1760   \int_compare:nNnTF { #1 } > { #3 }
1761   { \prg_return_true: }
1762   {
1763     \int_compare:nNnTF { #1 } < { #3 }
1764     { \prg_return_false: }
1765     {
1766       \int_compare:nNnTF { #2 } > { #4 }
1767       { \prg_return_true: }
1768       { \prg_return_false: }
1769     }
1770   }
1771 }

```

`\@@_addr_x_lt:nnnn` (*fn.*) checks whether $a < b$ holds
Sideeffects:

Arguments:

```
#1 in  addra
#2 in  xa
#3 in  addrb
#4 in  xb
```

```
1772 \prg_new_protected_conditional:Npnn
1773   \@@_addr_x_lt:nnnn
1774   #1 #2 #3 #4
1775 { TF }
1776 {
1777   \str_compare:eNeTF { #1 } < { #3 }
1778   { \prg_return_true: }
1779   {
1780     \str_compare:eNeTF { #1 } > { #3 }
1781     { \prg_return_false: }
1782     {
1783       \int_compare:nNnTF { #2 } < { #4 }
1784       { \prg_return_true: }
1785       { \prg_return_false: }
1786     }
1787   }
1788 }
```

\@@_addr_x_gt:nnnn (*fn.*) checks whether $a > b$ holds

Sideeffects:

Arguments:

```
#1 in  addra
#2 in  xa
#3 in  addrb
#4 in  xb
```

```
1789 \prg_new_protected_conditional:Npnn
1790   \@@_addr_x_gt:nnnn
1791   #1 #2 #3 #4
1792 { TF }
1793 {
1794   \str_compare:eNeTF { #1 } > { #3 }
1795   { \prg_return_true: }
1796   {
1797     \str_compare:eNeTF { #1 } < { #3 }
1798     { \prg_return_false: }
1799     {
1800       \int_compare:nNnTF { #2 } > { #4 }
1801       { \prg_return_true: }
1802       { \prg_return_false: }
1803     }
1804   }
1805 }
```

\@@_match_rule_idx:nnnn (*fn.*) Match an index rule against the current item.

```

Sideeffects:
  clobbered   dir  l_tmpa_int
  clobbered   dir  l_tmpb_int
Arguments:
  #1   in   context-id
  #2   in   y
  #3   in   x
  #4   in   rule-id
  #-   out  l_@@_range_state_int

1806 \cs_new_protected:Npn
1807   \@@_match_rule_idx:nnnn
1808   #1 #2 #3 #4
1809 {
1810   \int_set:Nn
1811     \l_tmpa_int
1812     {
1813       \seq_item:cn
1814         { \@@_ctx_var:nn { #1 } { idx_low_y_seq } }
1815         { #4 }
1816     }
1817   \int_set:Nn
1818     \l_tmpb_int
1819     {
1820       \seq_item:cn
1821         { \@@_ctx_var:nn { #1 } { idx_low_x_seq } }
1822         { #4 }
1823     }
1824   \@@_coord_lt:nnnnTF
1825   { #2 }
1826   { #3 }
1827   { \l_tmpa_int }
1828   { \l_tmpb_int }
1829   {
range is still ahead
1830     \int_set:Nn \l_@@_range_state_int { +1 }
1831   }
1832   {
1833     \int_set:Nn
1834       \l_tmpa_int
1835       {
1836         \seq_item:cn
1837           { \@@_ctx_var:nn { #1 } { idx_high_y_seq } }
1838           { #4 }
1839       }
1840     \int_set:Nn
1841       \l_tmpb_int
1842       {
1843         \seq_item:cn
1844           { \@@_ctx_var:nn { #1 } { idx_high_x_seq } }
1845           { #4 }
1846       }
1847     \@@_coord_gt:nnnnTF
1848     { #2 }

```



```

1849     { #3 }
1850     { \l_tmpa_int }
1851     { \l_tmpb_int }
1852     {
range has already passed
1853         \int_set:Nn \l_@@_range_state_int { -1 }
1854     }
1855     {
1856         \int_set:Nn \l_@@_range_state_int { 0 }
1857     }
1858 }
1859 }

```

\@@_match_rule_addr:nnnn (*fn.*) Match an address rule against the current item.

Sideeffects:

```

clobbered   dir   l_tmpa_tl
clobbered   dir   l_tmpa_int

```

Arguments:

```

#1   in   context-id
#2   in   address
#3   in   x
#4   in   rule-id
#-   out  l_@@_range_state_int

```

```

1860 \cs_new_protected:Npn
1861   \@@_match_rule_addr:nnnn
1862   #1 #2 #3 #4
1863 {
1864   \tl_set:Nc
1865     \l_tmpa_tl
1866     {
1867       \seq_item:cn
1868       { \@@_ctx_var:nn { #1 } { addr_low_seq } }
1869       { #4 }
1870     }
1871   \int_set:Nn
1872     \l_tmpa_int
1873     {
1874       \seq_item:cn
1875       { \@@_ctx_var:nn { #1 } { idx_low_x_seq } }
1876       { #4 }
1877     }
1878   \@@_addr_x_lt:nnnnTF
1879     { #2 }
1880     { #3 }
1881     { \l_tmpa_tl }
1882     { \l_tmpa_int }
1883     {

```

range is still ahead

```

1884     \int_set:Nn \l_@@_range_state_int { +1 }
1885   }
1886   {

```

```

1887     \tl_set:Ne
1888     \l_tmpa_tl
1889     {
1890       \seq_item:cn
1891       { \@@_ctx_var:nn { #1 } { addr_high_seq } }
1892       { #4 }
1893     }
1894     \int_set:Nn
1895     \l_tmpa_int
1896     {
1897       \seq_item:cn
1898       { \@@_ctx_var:nn { #1 } { idx_high_x_seq } }
1899       { #4 }
1900     }
1901     \@@_addr_x_gt:nnnnTF
1902     { #2 }
1903     { #3 }
1904     { \l_tmpa_tl }
1905     { \l_tmpa_int }
1906     {
range has already passed
1907     \int_set:Nn \l_@@_range_state_int { -1 }
1908     }
1909     {
1910     \int_set:Nn \l_@@_range_state_int { 0 }
1911     }
1912   }
1913 }

```

10.5 Iteration State Updates

`\@@_match_rule_state:NnnNn` (*fn.*) Matches an entire rule against the current position/address and checks if the rule is active, still ahead or already past.

Sideeffects:

`clobbered` `dir` `l_tmpa_tl`

Arguments:

#1 in context-id
#2 in *y*
#3 in *x*
#4 in address
#5 in rule-id
#- out `l_@@_range_state_int`

```

1914 \cs_new_protected:Npn
1915   \@@_match_rule_state:NnnNn
1916   #1 #2 #3 #4 #5
1917 {
1918   \tl_set:Ne
1919   \l_tmpa_tl
1920   {
1921     \seq_item:cn
1922     { \@@_ctx_var:nn { #1 } { kind_seq } }
1923     { #5 }

```

```

1924     }
1925     \str_case:VnF \l_tmpa_tl
1926     {
1927         { gap }
1928     {

```

range already “passed” (in a sense this range/rule’s processing is already done)

```

1929         \int_set:Nn \l_@@_range_state_int { -1 }
1930     }
1931     { addr }
1932     {
1933         \@@_match_rule_addr:nnnn
1934         { #1 }
1935         { #4 }
1936         { #3 }
1937         { #5 }
1938     }
1939     { idx }
1940     {
1941         \@@_match_rule_idx:nnnn
1942         { #1 }
1943         { #2 }
1944         { #3 }
1945         { #5 }
1946     }
1947 }
1948 {
1949     \msg_critical:nnV
1950     { hexdumptikz-selector }
1951     { invalid-rule }
1952     \l_tmpa_tl
1953 }
1954 }

```

`ch_add_active_rules:NnnNN` (*fn.*) Search linearly for rules which should become active and add them to the context’s `active_rules_seq`. Note the search starts at `next_rule_int` so the rules must be in ascending order.

Sideeffects:

```

    clobbered    dir    l_tmpa_tl
    clobbered    dir    l_@@_match_loop_bool

```

Arguments:

```

#1  in  context-id
#2  in  y
#3  in  x
#4  in  address
#5  in  bool whether all rules should be added or only a single one

```

```

1955 \cs_new_protected:Npn
1956   \@@_match_add_active_rules:NnnNN
1957   #1 #2 #3 #4 #5
1958 {

```

add next rules that already started

```

1959   \bool_set_true:N \l_@@_match_loop_bool

```

```

1960 \bool_while_do:Nn \l_@@_match_loop_bool
1961 {
1962   \int_compare:nNnTF
1963     {
1964       \int_use:c
1965       { \@@_ctx_var:nn { #1 } { next_rule_int } }
1966     }
1967     >
1968     {
1969       \seq_count:c
1970       { \@@_ctx_var:nn { #1 } { kind_seq } }
1971     }
1972     {
1973       \bool_set_false:N
1974       \l_@@_match_loop_bool
1975     }
1976     {
1977       \tl_set:Ne
1978       \l_tmpa_tl
1979       {
1980         \seq_item:cn
1981         { \@@_ctx_var:nn { #1 } { kind_seq } }
1982         {
1983           \int_use:c
1984           {
1985             \@@_ctx_var:nn
1986             { #1 }
1987             { next_rule_int }
1988           }
1989         }
1990       }
1991       \str_case:VnF \l_tmpa_tl
1992       {
1993         { gap }
1994         {

```

gaps are not active rules; just skip them here

```

1995         \int_gincr:c
1996         {
1997           \@@_ctx_var:nn
1998           { #1 }
1999           { next_rule_int }
2000         }
2001         \int_gincr:c
2002         {
2003           \@@_ctx_var:nn
2004           { #1 }
2005           { gap_cnt_int }
2006         }
2007       }
2008     }
2009     {
2010       \int_set:Nn
2011       \l_@@_rule_id_int
2012       {

```

```

2013         \int_use:c
2014         {
2015             \@@_ctx_var:nn
2016             { #1 }
2017             { next_rule_int }
2018         }
2019     }
2020
2021     \@@_match_rule_state:NnnNn
2022     #1
2023     { #2 }
2024     { #3 }
2025     #4
2026     { \l_@@_rule_id_int }
2027     \int_case:nn { \l_@@_range_state_int }
2028     {
2029         { -1 }
2030         {
range already passed → continue searching
2031             \int_gincr:c
2032             {
2033                 \@@_ctx_var:nn
2034                 { #1 }
2035                 { next_rule_int }
2036             }
2037         }
2038         { 0 }
2039         {
range is active → add and continue searching
2040             \seq_gput_right:ce
2041             {
2042                 \@@_ctx_var:nn
2043                 { #1 }
2044                 { active_rules_seq }
2045             }
2046             { \int_use:N \l_@@_rule_id_int }
2047             \bool_if:NTF #5
2048             {
continue searching
2049                 \int_gincr:c
2050                 {
2051                     \@@_ctx_var:nn
2052                     { #1 }
2053                     { next_rule_int }
2054                 }
2055             }
2056         {
stop after first found
2057             \int_gincr:c
2058             {
2059                 \@@_ctx_var:nn
2060                 { #1 }

```

```

2061             { next_rule_int }
2062         }
2063         \bool_set_false:N
2064         \l_@@_match_loop_bool
2065     }
2066 }
2067 { +1 }
2068 {
range is still ahead → stop
2069         \bool_set_false:N
2070         \l_@@_match_loop_bool
2071     }
2072 }
2073 }
2074 }
2075 }
2076 }

```

`_update_active_rules:NnnN (fn.)` Update the set of currently active rules in the context. This means rules which are not valid anymore get removed and newly activated rules get searched for and get added.

Sideeffects:

clobbered dir l_@@_active_tmp_seq

Arguments:

#1 in context-id
#2 in y
#3 in x
#4 in address

```

2077 \cs_new_protected:Npn
2078   \@@_match_update_active_rules:NnnN
2079   #1 #2 #3 #4
2080 {
2081   \seq_clear:N \l_@@_active_tmp_seq

```

remove rules that are no longer active

TOOD: can this be done in a more performant way? (than by copying the sequence over)

```

2082   \seq_map_inline:cn
2083     { \@@_ctx_var:nn { #1 } { active_rules_seq } }
2084     {
2085       \@@_match_rule_state:NnnNn
2086       #1
2087       { #2 }
2088       { #3 }
2089       #4
2090       { ##1 }
2091       \int_compare:nNnT
2092       { \l_@@_range_state_int }
2093       =
2094       { 0 }
2095       {
2096         \seq_put_right:Nn

```

```

2097         \l_@@_active_tmp_seq
2098         { ##1 }
2099     }
2100 }
2101 \seq_gset_eq:cN
2102 { \@@_ctx_var:nn { #1 } { active_rules_seq } }
2103 \l_@@_active_tmp_seq
search for rules which shall be added and add them
2104 \bool_set_true:N \l_@@_search_all_bool
2105 \@@_match_add_active_rules:NnnNN
2106     #1
2107     { #2 }
2108     { #3 }
2109     #4
2110     \l_@@_search_all_bool
2111 }

```

active_rules_fastpath:NnnN (*fn.*) fast-path when only one element in the active set is allowed

Sideeffects:

clobbered dir l_@@_search_all_bool

Arguments:

#1 in context-id
#2 in *y*
#3 in *x*
#4 in address

```

2112 \cs_new_protected:Npn
2113 \@@_match_update_active_rules_fastpath:NnnN
2114 #1 #2 #3 #4
2115 {

```

check whether the current rule shall be removed

```

2116 \int_compare:nNnT
2117 {
2118     \seq_count:c
2119     { \@@_ctx_var:nn { #1 } { active_rules_seq } }
2120 }
2121 >
2122 { 0 }
2123 {
2124     \int_set:Nn
2125     \l_@@_rule_id_int
2126     {
2127         \seq_item:cn
2128         {
2129             \@@_ctx_var:nn
2130             { #1 }
2131             { active_rules_seq }
2132         }
2133         { 1 }
2134     }
2135     \@@_match_rule_state:NnnNn
2136     #1

```

```

2137     { #2 }
2138     { #3 }
2139     #4
2140     { \l_@@_rule_id_int }
2141     \int_compare:nNnF
2142     { \l_@@_range_state_int }
2143     =
2144     { 0 }
2145     {
2146         \seq_gclear:c
2147         { \@@_ctx_var:nn { #1 } { active_rules_seq } }
2148     }
2149 }

```

check if has to search for the next rule whose range fits, if so do it and add it

```

2150 \bool_set_false:N \l_@@_search_all_bool
2151 \int_compare:nNnF
2152 {
2153     \seq_count:c
2154     { \@@_ctx_var:nn { #1 } { active_rules_seq } }
2155 }
2156 >
2157 { 0 }
2158 {
2159     \@@_match_add_active_rules:NnnNN
2160     #1
2161     { #2 }
2162     { #3 }
2163     #4
2164     \l_@@_search_all_bool
2165 }
2166 }

```

10.6 Public

`\selector_match_styles:NnnN` (*fn.*) Match the current position/address against the list of rules (adjusts the context's iterator output state). Intended when the selector specifies a style mapping. This supports overlapping ranges. Still the rules needs to be in ascending order, sorted by the start of the respective range.

Sideeffects:

Arguments:

```

#1 in context-id
#2 in y
#3 in x
#4 in address

```

```

2167 \cs_new_protected:Npn \hexdumptikz_selector_match_styles:NnnN #1 #2 #3 #4
2168 {

```

First clean the iterator output

```

2169 \hexdumptikz_selector_ctx_clear_iter_out:N #1

```

Update the set of active rules (remove and add if needed)

```

2170 \@@_match_update_active_rules:NnnN

```



```

2171 #1
2172 { #2 }
2173 { #3 }
2174 #4

```

collect the style which shall be applied from the set of active rules after filtering them with the corresponding predicate.

```

2175 \seq_map_inline:cn
2176 { \@@_ctx_var:nn { #1 } { active_rules_seq } }
2177 {
2178   \@@_match_rule_predicate:NnnNnT
2179   #1
2180   { #2 }
2181   { #3 }
2182   #4
2183   { ##1 }
2184   {
2185     \@@_match_apply_style:Nn #1 { ##1 }
2186     \bool_gset_true:c
2187     {
2188       \@@_ctx_var:nn
2189       { #1 }
2190       { show_bool }
2191     }
2192   }
2193 }
2194 }

```

`_selector_match_rows:NnnN` (*fn.*) Match the current position/address against the list of rules (adjusts the context's iterator output state). Intended when the selector is a raw specification of the selection. This does NOT support overlapping ranges. The rules needs to be in ascending order, sorted by the start of the respective range.

Sideeffects:

Arguments:

```

#1 in context-id
#2 in y
#3 in x
#4 in address

```

```

2195 \cs_new_protected:Npn \hexdumptikz_selector_match_rows:NnnN #1 #2 #3 #4
2196 {

```

First clean the iterator output

```

2197 \hexdumptikz_selector_ctx_clear_iter_out:N #1

```

Update the set of active rules (remove and add if needed) Use the fast-path since this needs to only support a single active rule/range.

```

2198 \@@_match_update_active_rules_fastpath:NnnN
2199 #1
2200 { #2 }
2201 { #3 }
2202 #4

```

check if all rules are already passed \rightarrow caller might want to stop at this point

```

2203 \seq_if_empty:cTF
2204 { \@@_ctx_var:nn { #1 } { active_rules_seq } }
2205 {
2206   \int_compare:nNnT
2207     {
2208       \int_use:c
2209       {
2210         \@@_ctx_var:nn
2211         { #1 }
2212         { next_rule_int }
2213       }
2214     }
2215   >
2216   {
2217     \seq_count:c
2218     { \@@_ctx_var:nn { #1 } { kind_seq } }
2219   }
2220   {
2221     \bool_gset_true:c
2222     {
2223       \@@_ctx_var:nn
2224       { #1 }
2225       { finished_bool }
2226     }
2227   }
2228 }
2229 {

```

no overlapping rules are supported, so we can use a fast-path here and just work with the single active rule

```

2230 \int_set:Nn
2231 \l_@@_rule_id_int
2232 {
2233   \seq_item:cn
2234   {
2235     \@@_ctx_var:nn
2236     { #1 }
2237     { active_rules_seq }
2238   }
2239   { 1 }
2240 }
2241 \@@_match_rule_predicate:NnnNnTF
2242 #1
2243 { #2 }
2244 { #3 }
2245 #4
2246 { \l_@@_rule_id_int }
2247 {
2248   \bool_gset_true:c
2249   {
2250     \@@_ctx_var:nn
2251     { #1 }
2252     { show_bool }

```

```
2253     }
2254   }
2255   {
2256     \bool_gset_false:c
2257     {
2258       \@@_ctx_var:nn
2259       { #1 }
2260       { show_bool }
2261     }
2262   }
2263 }
2264 }
```