tagpdf – A package to experiment with pdf tagging[*]

Ulrike Fischer[†]

Released 2023-03-07

Contents

---

[*]This file describes v0.98e, last revised 2023-03-07.

[†]E-mail: fischer@troubleshooting-tex.de

II   The **tagpdf-user** module
Code related to LATEX2e user commands and document commands

`\ref_value:nnn`	`\ref_value:nnn{`⟨*label*⟩`}{`⟨*attribute*⟩`}{`⟨*fallback default*⟩`}`

This is a temporary definition which will have to move to l3ref. It allows to locally set a default value if the label or the attribute doesn't exist. See issue #4 in Accessible-xref.

`\tag_stop_group_begin:`
`\tag_stop_group_end:`
`\tag_stop:`
`\tag_start:`

We need commands to stop tagging in some places. There simply switches the two local booleans. The grouping commands can be used to group the effect.

`\tag_stop:n`	`\tag_stop:n{`⟨*label*⟩`}`
`\tag_start:n`	`\tag_start:n{`⟨*label*⟩`}`

This commands are intended as a pair. The start command will only restart tagging if the previous stop command with the same label actually stopped tagging.

`activate-space`␣`(setup-key)`

`activate-space` activates the additional parsing needed for interword spaces. It is not documented, the parsing is currently implicitly activated by the known key `interwordspace`, as the code will perhaps move to some other place, now that it is better separated.

`activate-mc`␣`(setup-key)`
`activate-tree`␣`(setup-key)`
`activate-struct`␣`(setup-key)`
`activate-all`␣`(setup-key)`

Keys to activate the various tagging steps

`no-struct-dest`␣`(setup-key)`

The key allows to suppress the creation of structure destinations

`log`␣`(setup-key)`

The `log` takes currently the values `none`, `v`, `vv`, `vvv`, `all`. More details are in `tagpdf-checks`.

`tagunmarked`␣`(setup-key)`

This key allows to set if (in luamode) unmarked text should be marked up as artifact. The initial value is true.

`tabsorder`␣`(setup-key)`

This sets the tabsorder on a page. The values are `row`, `column`, `structure` (default) or `none`. Currently this is set more or less globally. More finer control can be added if needed.

`tagstruct`
`tagstructobj`
`tagabspage`
`tagmcabs`
`tagmcid`

These are attributes used by the label/ref system.

1 Initialization and test if pdfmanagement is active.

```
1  ⟨@@=tag⟩
2  ⟨*package⟩
3  \ProvidesExplPackage {tagpdf} {2023-03-07} {0.98e}
4    { A package to experiment with pdf tagging }
5
6  \bool_if:nF
7    {
8      \bool_lazy_and_p:nn
9        {\cs_if_exist_p:N \pdfmanagement_if_active_p:}
10       { \pdfmanagement_if_active_p: }
11   }
12   {  %error for now, perhaps warning later.
13     \PackageError{tagpdf}
14       {
15         PDF~resource~management~is~no~active!\MessageBreak
16         tagpdf~will~no~work.
17       }
18       {
19         Activate~it~with \MessageBreak
20         \string\RequirePackage{pdfmanagement-testphase}\MessageBreak
21         \string\DocumentMetadata{<options>}\MessageBreak
22         before~\string\documentclass
23       }
24   }
25 ⟨/package⟩
```

```
<*debug>
26 \ProvidesExplPackage {tagpdf-debug} {2023-03-07} {0.98e}
27   { debug code for tagpdf }
28 \@ifpackageloaded{tagpdf}{}{\PackageWarning{tagpdf-debug}{tagpdf~not~loaded,~quitting}\endinpu
```

</debug> We map the internal module name "tag" to "tagpdf" in messages.

```
29 ⟨*package⟩
30 \prop_gput:Nnn \g_msg_module_name_prop { tag }{ tagpdf }
31 ⟨/package⟩
```

Debug mode has its special mapping:

```
32 ⟨*debug⟩
33 \prop_gput:Nnn \g_msg_module_type_prop { tag / debug} {}
34 \prop_gput:Nnn \g_msg_module_name_prop { tag / debug }{tagpdf~DEBUG}
35 ⟨/debug⟩
```

2 base package

To avoid to have to test everywhere if tagpdf has been loaded and is active, we define a base package with dummy functions

```
36 ⟨*base⟩
37 \ProvidesExplPackage {tagpdf-base} {2023-03-07} {0.98e}
38   {part of tagpdf - provide base, no-op versions of the user commands }
39 ⟨/base⟩
```

3 Package options

There are only two options to switch for luatex between generic and luamode, TODO
try to get rid of them.

```
40 ⟨*package⟩
41 \bool_new:N\g__tag_mode_lua_bool
42 \DeclareOption {luamode}    { \sys_if_engine_luatex:T { \bool_gset_true:N \g__tag_mode_lua_boo
43 \DeclareOption {genericmode}{ \bool_gset_false:N\g__tag_mode_lua_bool }
44 \ExecuteOptions{luamode}
45 \ProcessOptions
```

4 Packages

We need the temporary version of l3ref until this is in the kernel.

```
46 \RequirePackage{l3ref-tmp}
```

To be on the safe side for now, load also the base definitions

```
47 \RequirePackage{tagpdf-base}
48 ⟨/package⟩
```

The no-op version should behave a near enough to the real code as possible, so we define
a command which a special in the relevant backends:

```
49 ⟨*base⟩
50 \AddToHook{begindocument}
51  {
52   \str_case:VnF \c_sys_backend_str
53    {
54     { luatex  } { \cs_new_protected:Npn \__tag_whatsits: {} }
55     { dvisvgm } { \cs_new_protected:Npn \__tag_whatsits: {} }
56    }
57    {
58     \cs_new_protected:Npn \__tag_whatsits: {\tex_special:D {} }
59    }
60  }
61 ⟨/base⟩
```

4.1 a LastPage label

See also issue #2 in Accessible-xref

`\__tag_lastpagelabel:`

```
62 ⟨*package⟩
63  \cs_new_protected:Npn \__tag_lastpagelabel:
64   {
65    \legacy_if:nT { @filesw }
66     {
67      \exp_args:NNnx \exp_args:NNx\iow_now:Nn \@auxout
68       {
69        \token_to_str:N \newlabeldata
70         {__tag_LastPage}
71         {
72          {abspage} { \int_use:N \g_shipout_readonly_int}
73          {tagmcabs}{ \int_use:N \c@g__tag_MCID_abs_int }
```

```
74                        {tagstruct}{\int_use:N \c@g__tag_struct_abs_int }
75                    }
76                }
77            }
78        }
79
80    \AddToHook{enddocument/afterlastpage}
81      {\__tag_lastpagelabel:}
```

(*End definition for* `\__tag_lastpagelabel:`.)

\ref_value:nnn   This allows to locally set a default value if the label or the attribute doesn't exist.

```
82    \cs_if_exist:NF \ref_value:nnn
83      {
84        \cs_new:Npn \ref_value:nnn #1#2#3
85          {
86            \exp_args:Nee
87              \__ref_value:nnn
88                { \tl_to_str:n {#1} } { \tl_to_str:n {#2} } {#3}
89          }
90        \cs_new:Npn \__ref_value:nnn #1#2#3
91          {
92            \tl_if_exist:cTF { g__ref_label_ #1 _ #2 _tl }
93              { \tl_use:c { g__ref_label_ #1 _ #2 _tl } }
94              {
95                #3
96              }
97          }
98      }
```

(*End definition for* `\ref_value:nnn`. *This function is documented on page 6.*)

5 Variables

\l__tag_tmpa_tl   A few temporary variables
\l__tag_tmpb_tl
\l__tag_get_tmpc_tl
\l__tag_get_parent_tmpb_tl␣␣␣␣\l__tag_tmpa_str
\l__tag_tmpa_prop
\l__tag_tmpa_seq
\l__tag_tmpb_seq
\l__tag_tmpa_clist
\l__tag_tmpa_int
\l__tag_tmpa_box
\l__tag_tmpb_box

```
99    \tl_new:N    \l__tag_tmpa_tl
100   \tl_new:N    \l__tag_tmpb_tl
101   \tl_new:N    \l__tag_get_tmpc_tl
102   \tl_new:N    \l__tag_get_parent_tmpa_tl
103   \tl_new:N    \l__tag_get_parent_tmpb_tl
104   \str_new:N   \l__tag_tmpa_str
105   \prop_new:N  \l__tag_tmpa_prop
106   \seq_new:N   \l__tag_tmpa_seq
107   \seq_new:N   \l__tag_tmpb_seq
108   \clist_new:N \l__tag_tmpa_clist
109   \int_new:N   \l__tag_tmpa_int
110   \box_new:N   \l__tag_tmpa_box
111   \box_new:N   \l__tag_tmpb_box
```

(*End definition for* `\l__tag_tmpa_tl` *and others.*)

Attribute lists for the label command. We have a list for mc-related labels, and one for structures.

`\c__tag_refmc_clist`
`\c__tag_refstruct_clist`

```
112 \clist_const:Nn \c__tag_refmc_clist     {tagabspage,tagmcabs,tagmcid}
113 \clist_const:Nn \c__tag_refstruct_clist {tagstruct,tagstructobj}
```

(*End definition for* `\c__tag_refmc_clist` *and* `\c__tag_refstruct_clist`.)

`\l__tag_loglevel_int` This integer hold the log-level and so allows to control the messages. TODO: a list which log-level shows what is needed. The current behaviour is quite ad-hoc.

```
114 \int_new:N  \l__tag_loglevel_int
```

(*End definition for* `\l__tag_loglevel_int`.)

`\g__tag_active_space_bool`
`\g__tag_active_mc_bool`
`\g__tag_active_tree_bool`
`\g__tag_active_struct_bool`
`\g_tag_active_struct_dest_bool`

These booleans should help to control the global behaviour of tagpdf. Ideally it should more or less do nothing if all are false. The space-boolean controles the interword space code, the mc-boolean activates `\tag_mc_begin:n`, the tree-boolean activates writing the finish code and the pdfmanagement related commands, the struct-boolean activates the storing of the structure data. In a normal document all should be active, the split is only there for debugging purpose. Structure destination will be activated automatically if pdf version 2.0 is detected, but with the boolean struct-dest-boolean one can suppress them. Also we assume currently that they are set only at begin document. But if some control passing over groups are needed they could be perhaps used in a document too. TODO: check if they are used everywhere as needed and as wanted.

```
115 \bool_new:N \g__tag_active_space_bool
116 \bool_new:N \g__tag_active_mc_bool
117 \bool_new:N \g__tag_active_tree_bool
118 \bool_new:N \g__tag_active_struct_bool
119 \bool_new:N \g__tag_active_struct_dest_bool
120 \bool_gset_true:N \g__tag_active_struct_dest_bool
```

(*End definition for* `\g__tag_active_space_bool` *and others.*)

`\l__tag_active_mc_bool`
`\l__tag_active_struct_bool`

These booleans should help to control the *local* behaviour of tagpdf. In some cases it could e.g. be necessary to stop tagging completely. As local booleans they respect groups. TODO: check if they are used everywhere as needed and as wanted.

```
121 \bool_new:N \l__tag_active_mc_bool
122 \bool_set_true:N \l__tag_active_mc_bool
123 \bool_new:N \l__tag_active_struct_bool
124 \bool_set_true:N \l__tag_active_struct_bool
```

(*End definition for* `\l__tag_active_mc_bool` *and* `\l__tag_active_struct_bool`.)

`\g__tag_tagunmarked_bool` This boolean controls if the code should try to automatically tag parts not in mc-chunk. It is currently only used in luamode. It would be possible to used it in generic mode, but this would create quite a lot empty artifact mc-chunks.

```
125 \bool_new:N \g__tag_tagunmarked_bool
```

(*End definition for* `\g__tag_tagunmarked_bool`.)

6   Variants of l3 commands

```
126 \prg_generate_conditional_variant:Nnn \pdf_object_if_exist:n {e}{T,F}
127 \cs_generate_variant:Nn \pdf_object_ref:n {e}
128 \cs_generate_variant:Nn \pdfannot_dict_put:nnn {nnx}
129 \cs_generate_variant:Nn \pdffile_embed_stream:nnn {nxx,oxx}
130 \cs_generate_variant:Nn \prop_gput:Nnn {Nxx,Nen}
131 \cs_generate_variant:Nn \prop_put:Nnn  {Nxx}
132 \cs_generate_variant:Nn \prop_item:Nn {No,Ne}
133 \cs_generate_variant:Nn \ref_label:nn { nv }
134 \cs_generate_variant:Nn \seq_set_split:Nnn{Nne}
135 \cs_generate_variant:Nn \str_set_convert:Nnnn {Nonn, Noon, Nnon }
136 \cs_generate_variant:Nn \clist_map_inline:nn {on}
```

7   Setup label attributes

tagstruct
tagstructobj
tagabspage
tagmcabs
tagmcid

This are attributes used by the label/ref system. With structures we store the structure number `tagstruct` and the object reference `tagstructobj`. The second is needed to be able to reference a structure which hasn't been created yet. The alternative would be to create the object in such cases, but then we would have to check the object existence all the time.

With mc-chunks we store the absolute page number `tagabspage`, the absolute id `tagmcabc`, and the id on the page `tagmcid`.

```
137 \ref_attribute_gset:nnnn { tagstruct } {0} { now }
138   { \int_use:N \c@g__tag_struct_abs_int }
139 \ref_attribute_gset:nnnn { tagstructobj } {} { now }
140   {
141     \pdf_object_if_exist:eT {__tag/struct/\int_use:N \c@g__tag_struct_abs_int}
142       {
143         \pdf_object_ref:e{__tag/struct/\int_use:N \c@g__tag_struct_abs_int}
144       }
145   }
146 \ref_attribute_gset:nnnn { tagabspage } {0} { shipout }
147   { \int_use:N \g_shipout_readonly_int }
148 \ref_attribute_gset:nnnn { tagmcabs } {0} { now }
149   { \int_use:N \c@g__tag_MCID_abs_int }
150 \ref_attribute_gset:nnnn {tagmcid }  {0} { now }
151   { \int_use:N \g__tag_MCID_tmp_bypage_int }
```

(*End definition for* `tagstruct` *and others. These functions are documented on page 6.*)

8   Label commands

__tag_ref_label:nn

A version of `\ref_label:nn` to set a label which takes a keyword `mc` or `struct` to call the relevant lists. TODO: check if `\@bsphack` and `\@esphack` make sense here.

```
152 \cs_new_protected:Npn \__tag_ref_label:nn #1 #2 %#1 label, #2 name of list mc or struct
153   {
154     \@bsphack
155     \ref_label:nv {#1}{c__tag_ref#2_clist}
156     \@esphack
157   }
158 \cs_generate_variant:Nn \__tag_ref_label:nn {en}
```

(End definition for `\__tag_ref_label:nn`.*)*

`\__tag_ref_value:nnn`   A local version to retrieve the value. It is a direct wrapper, but to keep naming consistent
. . . . It uses the variant defined temporarily above.

```
159 \cs_new:Npn \__tag_ref_value:nnn #1 #2 #3 %#1 label, #2 attribute, #3 default
160   {
161     \ref_value:nnn {#1}{#2}{#3}
162   }
163 \cs_generate_variant:Nn \__tag_ref_value:nnn {enn}
```

(End definition for `\__tag_ref_value:nnn`.*)*

`\__tag_ref_value_lastpage:nn`   A command to retrieve the lastpage label, this will be adapted when there is a proper,
kernel lastpage label.

```
164 \cs_new:Npn \__tag_ref_value_lastpage:nn #1 #2
165   {
166     \ref_value:nnn {__tag_LastPage}{#1}{#2}
167   }
```

(End definition for `\__tag_ref_value_lastpage:nn`.*)*

9   Commands to fill seq and prop

With most engines these are simply copies of the expl3 commands, but luatex will over-
write them, to store the data also in lua tables.

```
\__tag_prop_new:N
\__tag_seq_new:N
\__tag_prop_gput:Nnn
\__tag_seq_gput_right:Nn
\__tag_seq_item:cn
\__tag_prop_item:cn
\__tag_seq_show:N
\__tag_prop_show:N
```

```
168 \cs_set_eq:NN \__tag_prop_new:N          \prop_new:N
169 \cs_set_eq:NN \__tag_seq_new:N           \seq_new:N
170 \cs_set_eq:NN \__tag_prop_gput:Nnn       \prop_gput:Nnn
171 \cs_set_eq:NN \__tag_seq_gput_right:Nn \seq_gput_right:Nn
172 \cs_set_eq:NN \__tag_seq_item:cn         \seq_item:cn
173 \cs_set_eq:NN \__tag_prop_item:cn        \prop_item:cn
174 \cs_set_eq:NN \__tag_seq_show:N          \seq_show:N
175 \cs_set_eq:NN \__tag_prop_show:N         \prop_show:N
176
177 \cs_generate_variant:Nn \__tag_prop_gput:Nnn      { Nxn , Nxx, Nnx , cnn, cxn, cnx, cno}
178 \cs_generate_variant:Nn \__tag_seq_gput_right:Nn { Nx  , No, cn, cx }
179 \cs_generate_variant:Nn \__tag_prop_new:N    { c }
180 \cs_generate_variant:Nn \__tag_seq_new:N     { c }
181 \cs_generate_variant:Nn \__tag_seq_show:N    { c }
182 \cs_generate_variant:Nn \__tag_prop_show:N   { c }
```

(End definition for `\__tag_prop_new:N` *and others.)*

10   General tagging commands

```
\tag_stop_group_begin:
\tag_stop_group_end:
\tag_stop:
\tag_start:
\tag_stop:n
\tag_start:n
```

We need commands to stop tagging in some places. This simply switches the two local
booleans. In some cases tagging should only restart, if it actually was stopped before.
For this it is possible to label a stop.

```
183 \cs_new_protected:Npn \tag_stop_group_begin:
184   {
```

```
185     \group_begin:
186     \bool_set_false:N \l__tag_active_struct_bool
187     \bool_set_false:N \l__tag_active_mc_bool
188   }
189 \cs_set_eq:NN \tag_stop_group_end: \group_end:
190 \cs_set_protected:Npn \tag_stop:
191   {
192     \bool_set_false:N \l__tag_active_struct_bool
193     \bool_set_false:N \l__tag_active_mc_bool
194   }
195 \cs_set_protected:Npn \tag_start:
196   {
197     \bool_set_true:N \l__tag_active_struct_bool
198     \bool_set_true:N \l__tag_active_mc_bool
199   }
200 \prop_new:N\g__tag_state_prop
201 \cs_set_protected:Npn \tag_stop:n #1
202   {
203     \tag_if_active:TF
204       {
205         \bool_set_false:N \l__tag_active_struct_bool
206         \bool_set_false:N \l__tag_active_mc_bool
207         \prop_gput:Nnn \g__tag_state_prop { #1 }{ 1 }
208       }
209       {
210         \prop_gremove:Nn \g__tag_state_prop { #1 }
211       }
212   }
213 \cs_set_protected:Npn \tag_start:n #1
214   {
215     \prop_gpop:NnN \g__tag_state_prop {#1}\l__tag_tmpa_tl
216     \quark_if_no_value:NF \l__tag_tmpa_tl
217       {
218         \bool_set_true:N \l__tag_active_struct_bool
219         \bool_set_true:N \l__tag_active_mc_bool
220       }
221   }
222 ⟨/package⟩
223 ⟨*base⟩
224 \cs_new_protected:Npn \tag_stop:{}
225 \cs_new_protected:Npn \tag_start:{}
226 \cs_new_protected:Npn \tag_stop:n  #1 {}
227 \cs_new_protected:Npn \tag_start:n #1 {}
228 ⟨/base⟩
```

(*End definition for* \tag_stop_group_begin: *and others. These functions are documented on page 6.*)

11   Keys for tagpdfsetup

TODO: the log-levels must be sorted

activate-space␣(setup-key)
activate-mc␣(setup-key)
activate-tree␣(setup-key)
activate-struct␣(setup-key)
activate-all␣(setup-key)
no-struct-dest␣(setup-key)

Keys to (globally) activate tagging. `activate-space` activates the additional parsing needed for interword spaces. It is not documented, the parsing is currently implicitly

13

activated by the known key `interwordspace`, as the code will perhaps move to some other place, now that it is better separated. `no-struct-dest` allows to suppress structure destinations.

```
229 ⟨*package⟩
230 \keys_define:nn { __tag / setup }
231   {
232     activate-space  .bool_gset:N = \g__tag_active_space_bool,
233     activate-mc     .bool_gset:N = \g__tag_active_mc_bool,
234     activate-tree   .bool_gset:N = \g__tag_active_tree_bool,
235     activate-struct .bool_gset:N = \g__tag_active_struct_bool,
236     activate-all    .meta:n =
237       {activate-mc={#1},activate-tree={#1},activate-struct={#1}},
238     activate-all  .default:n = true,
239     no-struct-dest .bool_gset_inverse:N = \g__tag_active_struct_dest_bool,
240
```

(*End definition for* `activate-space` *(setup-key) and others. These functions are documented on page* *6.*)

log␣(setup-key)   The `log` takes currently the values `none`, `v`, `vv`, `vvv`, `all`. The description of the log levels is in tagpdf-checks.

```
241     log             .choice:,
242     log / none      .code:n = {\int_set:Nn \l__tag_loglevel_int { 0 }},
243     log / v         .code:n =
244       {
245         \int_set:Nn \l__tag_loglevel_int { 1 }
246         \cs_set_protected:Nn \__tag_check_typeout_v:n { \iow_term:x {##1} }
247       },
248     log / vv        .code:n = {\int_set:Nn \l__tag_loglevel_int { 2 }},
249     log / vvv       .code:n = {\int_set:Nn \l__tag_loglevel_int { 3 }},
250     log / all       .code:n = {\int_set:Nn \l__tag_loglevel_int { 10 }},
```

(*End definition for* `log` *(setup-key). This function is documented on page* *6.*)

tagunmarked␣(setup-key)   This key allows to set if (in luamode) unmarked text should be marked up as artifact. The initial value is true.

```
251     tagunmarked     .bool_gset:N = \g__tag_tagunmarked_bool,
252     tagunmarked     .initial:n  = true,
```

(*End definition for* `tagunmarked` *(setup-key). This function is documented on page* *6.*)

tabsorder␣(setup-key)   This sets the tabsorder on a page. The values are `row`, `column`, `structure` (default) or `none`. Currently this is set more or less globally. More finer control can be added if needed.

```
253     tabsorder       .choice:,
254     tabsorder / row       .code:n =
255       \pdfmanagement_add:nnn { Page } {Tabs}{/R},
256     tabsorder / column    .code:n =
257       \pdfmanagement_add:nnn { Page } {Tabs}{/C},
258     tabsorder / structure .code:n =
259       \pdfmanagement_add:nnn { Page } {Tabs}{/S},
260     tabsorder / none      .code:n =
261       \pdfmanagement_remove:nn {Page} {Tabs},
262     tabsorder       .initial:n = structure,
```

```
263    uncompress       .code:n = { \pdf_uncompress:  },
264  }
```

(End definition for `tabsorder` *(setup-key). This function is documented on page* *.)*

12 loading of engine/more dependent code

```
265 \sys_if_engine_luatex:T
266   {
267     \file_input:n {tagpdf-luatex.def}
268   }
269 ⟨/package⟩
270 ⟨∗mcloading⟩
271 \bool_if:NTF \g__tag_mode_lua_bool
272   {
273    \RequirePackage {tagpdf-mc-code-lua}
274   }
275   {
276    \RequirePackage {tagpdf-mc-code-generic} %
277   }
278 ⟨/mcloading⟩
279 ⟨∗debug⟩
280 \bool_if:NTF \g__tag_mode_lua_bool
281   {
282    \RequirePackage {tagpdf-debug-lua}
283   }
284   {
285    \RequirePackage {tagpdf-debug-generic} %
286   }
287 ⟨/debug⟩
```

Part I
The **tagpdf-checks** module
Messages and check code
Part of the tagpdf package

1 Commands

`\tag_if_active_p:` ⋆	
`\tag_if_active:`*TF* ⋆	

This command tests if tagging is active. It only gives true if all tagging has been activated, *and* if tagging hasn't been stopped locally.

`\tag_get:n` ⋆	

`\tag_get:n{`⟨*keyword*⟩`}`

This is a generic command to retrieve data for the current structure or mc-chunk. Currently the only sensible values for the argument ⟨*keyword*⟩ are `mc_tag`, `struct_tag`, `struct_id` and `struct_num`.

2 Description of log messages

2.1 \ShowTagging command

Argument	type	note
\ShowTaggingmc-data = num	log+term	lua-only
\ShowTaggingmc-current	log+term	
\ShowTaggingstruck-stack= [log|show]	log or term+stop	

2.2 Messages in checks and commands

command	message	action
`\@@_check_structure_has_tag:n`	`struct-missing-tag`	error
`\@@_check_structure_tag:N`	`role-unknown-tag`	warning
`\@@_check_info_closing_struct:n`	`struct-show-closing`	info
`\@@_check_no_open_struct:`	`struct-faulty-nesting`	error
`\@@_check_struct_used:n`	`struct-used-twice`	warning
`\@@_check_add_tag_role:nn`	`role-missing, role-tag, role-unknown`	warning, info (>0), warning
`\@@_check_mc_if_nested:,`	`mc-nested`	warning
`\@@_check_mc_if_open:`	`mc-not-open`	warning
`\@@_check_mc_pushed_popped:nn`	`mc-pushed, mc-popped`	info (2), info+`seq_log` (>2)
`\@@_check_mc_tag:N`	`mc-tag-missing, role-unknown-tag`	error (missing), warning (unknown).
`\@@_check_mc_used:n`	`mc-used-twice`	warning
`\@@_check_show_MCID_by_page:`		
`\tag_mc_use:n`	`mc-label-unknown, mc-used-twice`	warning
`\role_add_tag:nn`	`new-tag`	info (>0)
	`sys-no-interwordspace`	warning
`\@@_struct_write_obj:n`	`struct-no-objnum`	error
`\tag_struct_begin:n`	`struct-faulty-nesting`	error
`\@@_struct_insert_annot:nn`	`struct-faulty-nesting`	error
`tag_struct_use:n`	`struct-label-unknown`	warning
`attribute-class, attribute`	`attr-unknown`	error
`\@@_tree_fill_parenttree:`	`tree-mcid-index-wrong`	warning TODO: should trigger a standard rerun m
in `enddocument/info`-hook	`para-hook-count-wrong`	error (warning?)

2.3 Messages from the ptagging code

A few messages are issued in generic mode from the code which reinserts missing TMB/TME. This is currently done if log-level is larger than zero. TODO: reconsider log-level and messages when this code settles down.

2.4 Warning messages from the lua-code

The messages are triggered if the log-level is at least equal to the number.

message	log-level	remark
WARN TAG-NOT-TAGGED:	1	
WARN TAG-OPEN-MC:	1	
WARN SHIPOUT-MC-OPEN:	1	
WARN SHIPOUT-UPS:	0	shouldn't happen
WARN TEX-MC-INSERT-MISSING:	0	shouldn't happen
WARN TEX-MC-INSERT-NO-KIDS:	2	e.g. from empty hbox

2.5 Info messages from the lua-code

The messages are triggered if the log-level is at least equal to the number. `TAG` messages are from the traversing function, `TEX` from code used in the tagpdf-mc module. `PARENTREE` is the code building the parenttree.

message	log-level	remark
INFO SHIPOUT-INSERT-LAST-EMC	3	finish of shipout code
INFO SPACE-FUNCTION-FONT	3	interwordspace code
INFO TAG-ABSPAGE	3	
INFO TAG-ARGS	4	
INFO TAG-ENDHEAD	4	
INFO TAG-ENDHEAD	4	
INFO TAG-HEAD	3	
INFO TAG-INSERT-ARTIFACT	3	
INFO TAG-INSERT-BDC	3	
INFO TAG-INSERT-EMC	3	
INFO TAG-INSERT-TAG	3	
INFO TAG-KERN-SUBTYPE	4	
INFO TAG-MATH-SUBTYPE	4	
INFO TAG-MC-COMPARE	4	
INFO TAG-MC-INTO-PAGE	3	
INFO TAG-NEW-MC-NODE	4	
INFO TAG-NODE	3	
INFO TAG-NO-HEAD	3	
INFO TAG-NOT-TAGGED	2	replaced by artifact
INFO TAG-QUITTING-BOX	4	
INFO TAG-STORE-MC-KID	4	
INFO TAG-TRAVERSING-BOX 3		
INFO TAG-USE-ACTUALTEXT	3	
INFO TAG-USE-ALT	3	
INFO TAG-USE-RAW	3	
INFO TEX-MC-INSERT-KID	3	

message	log-level	remark
INFO TEX-MC-INSERT-KID-TEST	4	
INFO TEX-MC-INTO-STRUCT	3	
INFO TEX-STORE-MC-DATA	3	
INFO TEX-STORE-MC-KID	3	
INFO PARENTTREE-CHUNKS	3	
INFO PARENTTREE-NO-DATA	3	
INFO PARENTTREE-NUM	3	
INFO PARENTTREE-NUMENTRY	3	
INFO PARENTTREE-STRUCT-OBJREF	4	

2.6   Debug mode messages and code

If the package tagpdf-debug is loaded a number of commands are redefined and enhanced with additional commands which can be used to output debug messages or collect statistics. The commands are present but do nothing if the log-level is zero.

command	name	action	remark
\tag_mc_begin:n	mc-begin-insert	msg	
	mc-begin-ignore	msg	if inactive

2.7   Messages

```
mc-nested
mc-tag-missing
mc-label-unknown
mc-used-twice
mc-not-open
mc-pushed
mc-popped
mc-current
```
Various messages related to mc-chunks. TODO document their meaning.

```
struct-no-objnum
struct-faulty-nesting
struct-missing-tag
struct-used-twice
struct-label-unknown
struct-show-closing
```
Various messages related to structure. TODO document their meaning.

```
attr-unknown
```
Message if an attribute i sunknown.

```
role-missing
role-unknown
role-unknown-tag
role-tag
new-tag
```
Messages related to role mapping.

tree-mcid-index-wrong Used in the tree code, typically indicates the document must be rerun.

sys-no-interwordspace Message if an engine doesn't support inter word spaces

para-hook-count-wrong Message if the number of begin paragraph and end paragraph differ. This normally means faulty structure.

```
1 ⟨@@=tag⟩
2 ⟨∗header⟩
3 \ProvidesExplPackage {tagpdf-checks-code} {2023-03-07} {0.98e}
4   {part of tagpdf - code related to checks, conditionals, debugging and messages}
5 ⟨/header⟩
```

3 Messages

3.1 Messages related to mc-chunks

mc-nested This message is issue is a mc is opened before the previous has been closed. This is not relevant for luamode, as the attributes don't care about this. It is used in the `\@@_check_mc_if_nested:` test.

```
6 ⟨∗package⟩
7 \msg_new:nnn { tag } {mc-nested} { nested~marked~content~found~-~mcid~#1 }
```

(*End definition for* mc-nested. *This function is documented on page 18.*)

mc-tag-missing If the tag is missing

```
8 \msg_new:nnn { tag } {mc-tag-missing} { required~tag~missing~-~mcid~#1 }
```

(*End definition for* mc-tag-missing. *This function is documented on page 18.*)

mc-label-unknown If the label of a mc that is used in another place is not known (yet) or has been undefined as the mc was already used.

```
9  \msg_new:nnn { tag } {mc-label-unknown}
10   { label~#1~unknown~or~has~been~already~used.\\
11     Either~rerun~or~remove~one~of~the~uses. }
```

(*End definition for* mc-label-unknown. *This function is documented on page 18.*)

mc-used-twice An mc-chunk can be inserted only in one structure. This indicates wrong coding and so should at least give a warning.

```
12 \msg_new:nnn { tag } {mc-used-twice} { mc~#1~has~been~already~used }
```

(*End definition for* mc-used-twice. *This function is documented on page 18.*)

mc-not-open This is issued if a `\tag_mc_end:` is issued wrongly, wrong coding.

```
13 \msg_new:nnn { tag } {mc-not-open} { there~is~no~mc~to~end~at~#1 }
```

(*End definition for* mc-not-open. *This function is documented on page 18.*)

mc-pushed
mc-popped Informational messages about mc-pushing.

```
14 \msg_new:nnn { tag } {mc-pushed} { #1~has~been~pushed~to~the~mc~stack}
15 \msg_new:nnn { tag } {mc-popped} { #1~has~been~removed~from~the~mc~stack }
```

(*End definition for* mc-pushed *and* mc-popped*. These functions are documented on page 18.*)

mc-current Informational messages about current mc state.

```
16 \msg_new:nnn { tag } {mc-current}
17   { current~MC:~
18     \bool_if:NTF\g__tag_in_mc_bool
19       {abscnt=\__tag_get_mc_abs_cnt:,~tag=\g__tag_mc_key_tag_tl}
20       {no~MC~open,~current~abscnt=\__tag_get_mc_abs_cnt:"}
21   }
```

(*End definition for* mc-current*. This function is documented on page 18.*)

3.2 Messages related to structures

struct-unknown if for example a parent key value points to structure that doesn't exist (yet)

```
22 \msg_new:nnn { tag } {struct-unknown}
23   { structure~with~number~#1~doesn't~exist\\ #2 }
```

(*End definition for* struct-unknown*. This function is documented on page* **??***.*)

struct-no-objnum Should not happen . . .

```
24 \msg_new:nnn { tag } {struct-no-objnum} { objnum~missing~for~structure~#1 }
```

(*End definition for* struct-no-objnum*. This function is documented on page 18.*)

struct-faulty-nesting This indicates that there is somewhere one \tag_struct_end: too much. This should be normally an error.

```
25 \msg_new:nnn { tag }
26   {struct-faulty-nesting}
27   { there~is~no~open~structure~on~the~stack }
```

(*End definition for* struct-faulty-nesting*. This function is documented on page 18.*)

struct-missing-tag A structure must have a tag.

```
28 \msg_new:nnn { tag } {struct-missing-tag} { a~structure~must~have~a~tag! }
```

(*End definition for* struct-missing-tag*. This function is documented on page 18.*)

struct-used-twice

```
29 \msg_new:nnn { tag } {struct-used-twice}
30   { structure~with~label~#1~has~already~been~used}
```

(*End definition for* struct-used-twice*. This function is documented on page 18.*)

struct-label-unknown label is unknown, typically needs a rerun.

```
31 \msg_new:nnn { tag } {struct-label-unknown}
32   { structure~with~label~#1~is~unknown~rerun}
```

(*End definition for* struct-label-unknown*. This function is documented on page 18.*)

struct-show-closing Informational message shown if log-mode is high enough

```
33 \msg_new:nnn { tag } {struct-show-closing}
34   { closing~structure~#1~tagged~\use:e{\prop_item:cn{g__tag_struct_#1_prop}{S}} }
```

(*End definition for* `struct-show-closing`. *This function is documented on page 18.*)

tree-struct-still-open Message issued at the end if there are beside Root other open structures on the stack.

```
35 \msg_new:nnn { tag } {tree-struct-still-open}
36   {
37     There~are~still~open~structures~on~the~stack!\\
38     The~stack~contains~\seq_use:Nn\g__tag_struct_tag_stack_seq{,}.\\
39     The~structures~are~automatically~closed,\\
40     but~their~nesting~can~be~wrong.
41   }
```

(*End definition for* `tree-struct-still-open`. *This function is documented on page* **??**.)

3.3 Attributes

Not much yet, as attributes aren't used so much.

attr-unknown

```
42 \msg_new:nnn { tag } {attr-unknown}  { attribute~#1~is~unknown}
```

(*End definition for* `attr-unknown`. *This function is documented on page 18.*)

3.4 Roles

role-missing Warning message if either the tag or the role is missing
role-unknown
role-unknown-tag

```
43 \msg_new:nnn { tag } {role-missing}     { tag~#1~has~no~role~assigned  }
44 \msg_new:nnn { tag } {role-unknown}     { role~#1~is~not~known  }
45 \msg_new:nnn { tag } {role-unknown-tag} { tag~#1~is~not~known  }
```

(*End definition for* `role-missing`, `role-unknown`, *and* `role-unknown-tag`. *These functions are documented on page 18.*)

role-parent-child This is info and warning message about the containment rules between child and parent tags.

```
46 \msg_new:nnn { tag } {role-parent-child}
47   { The~rule~between~parent~'#1'~\\and~child~'#2'~is~#3}
```

(*End definition for* `role-parent-child`. *This function is documented on page* **??**.)

role-parent-child This is info and warning message about the containment rules between child and parent tags.

```
48 \msg_new:nnn { tag } {role-remapping}
49   { remapping~tag~to~#1 }
```

(*End definition for* `role-parent-child`. *This function is documented on page* **??**.)

role-tag

new-tag

Info messages.

```
50 \msg_new:nnn { tag } {role-tag}          { mapping~tag~#1~to~role~#2  }
51 \msg_new:nnn { tag } {new-tag}           { adding~new~tag~#1 }
52 \msg_new:nnn { tag } {read-namespace}    { reading~namespace~definitions~tagpdf-ns-
   #1.def }
53 \msg_new:nnn { tag } {namespace-missing}{ namespace~definitions~tagpdf-ns-#1.def~not~found }
54 \msg_new:nnn { tag } {namespace-unknown}{ namespace~#1~is~not~declared }
```

(*End definition for* role-tag *and* new-tag. *These functions are documented on page 18.*)

3.5   Miscellaneous

tree-mcid-index-wrong

Used in the tree code, typically indicates the document must be rerun.

```
55 \msg_new:nnn { tag } {tree-mcid-index-wrong}
56   {something~is~wrong~with~the~mcid--rerun}
```

(*End definition for* tree-mcid-index-wrong. *This function is documented on page 19.*)

sys-no-interwordspace

Currently only pdflatex and lualatex have some support for real spaces.

```
57 \msg_new:nnn { tag } {sys-no-interwordspace}
58   {engine/output~mode~#1~doesn't~support~the~interword~spaces}
```

(*End definition for* sys-no-interwordspace. *This function is documented on page 19.*)

__tag_check_typeout_v:n

A simple logging function. By default is gobbles its argument, but the log-keys sets it to
typeout.

```
59 \cs_set_eq:NN \__tag_check_typeout_v:n \use_none:n
```

(*End definition for* __tag_check_typeout_v:n.)

para-hook-count-wrong

At the end of the document we check if the count of para-begin and para-end is identical.
If not we issue a warning: this is normally a coding error and and breaks the structure.

```
60 \msg_new:nnnn { tag } {para-hook-count-wrong}
61   {The~number~of~automatic~begin~(#1)~and~end~(#2)~#3~para~hooks~differ!}
62   {This~quite~probably~a~coding~error~and~the~structure~will~be~wrong!}
63 ⟨/package⟩
```

(*End definition for* para-hook-count-wrong. *This function is documented on page 19.*)

4   Retrieving data

\tag_get:n

This retrieves some data. This is a generic command to retrieve data. Currently the only
sensible values for the argument are mc_tag, struct_tag and struct_num.

```
64 ⟨base⟩\cs_new:Npn \tag_get:n #1   { \use:c {__tag_get_data_#1: } }
```

(*End definition for* \tag_get:n. *This function is documented on page 16.*)

5 User conditionals

This is a test it tagging is active. This allows packages to add conditional code. The test is true if all booleans, the global and the two local one are true.

```
65 ⟨*base⟩
66 \prg_new_conditional:Npnn \tag_if_active: { p , T , TF, F }
67   { \prg_return_false: }
68 ⟨/base⟩
69 ⟨*package⟩
70 \prg_set_conditional:Npnn \tag_if_active: { p , T , TF, F }
71   {
72     \bool_lazy_all:nTF
73       {
74         {\g__tag_active_struct_bool}
75         {\g__tag_active_mc_bool}
76         {\g__tag_active_tree_bool}
77         {\l__tag_active_struct_bool}
78         {\l__tag_active_mc_bool}
79       }
80       {
81         \prg_return_true:
82       }
83       {
84         \prg_return_false:
85       }
86   }
```

(*End definition for* `\tag_if_active:TF`*. This function is documented on page* *16*.)

6 Internal checks

These are checks used in various places in the code.

6.1 checks for active tagging

This checks if mc are active.

```
87 \prg_new_conditional:Npnn \__tag_check_if_active_mc: {T,F,TF}
88   {
89     \bool_lazy_and:nnTF { \g__tag_active_mc_bool } { \l__tag_active_mc_bool }
90       {
91         \prg_return_true:
92       }
93       {
94         \prg_return_false:
95       }
96   }
97 \prg_new_conditional:Npnn \__tag_check_if_active_struct: {T,F,TF}
98   {
99     \bool_lazy_and:nnTF { \g__tag_active_struct_bool } { \l__tag_active_struct_bool }
100      {
101        \prg_return_true:
102      }
```

23

```
103        {
104            \prg_return_false:
105        }
106    }
```

(*End definition for* `\__tag_check_if_active_mc:TF` *and* `\__tag_check_if_active_struct:TF`.)

6.2   Checks related to structures

__tag_check_structure_has_tag:n   Structures must have a tag, so we check if the S entry is in the property. It is an error if this is missing. The argument is a number. The tests for existence and type is split in structures, as the tags are stored differently to the mc case.

```
107 \cs_new_protected:Npn \__tag_check_structure_has_tag:n #1 %#1 struct num
108    {
109        \prop_if_in:cnF { g__tag_struct_#1_prop }
110        {S}
111        {
112            \msg_error:nn { tag } {struct-missing-tag}
113        }
114    }
```

(*End definition for* `\__tag_check_structure_has_tag:n`.)

__tag_check_structure_tag:N   This checks if the name of the tag is known, either because it is a standard type or has been rolemapped.

```
115 \cs_new_protected:Npn \__tag_check_structure_tag:N #1
116    {
117        \prop_if_in:NoF \g__tag_role_tags_NS_prop {#1}
118        {
119            \msg_warning:nnx { tag } {role-unknown-tag} {#1}
120        }
121    }
```

(*End definition for* `\__tag_check_structure_tag:N`.)

__tag_check_info_closing_struct:n   This info message is issued at a closing structure, the use should be guarded by log-level.

```
122 \cs_new_protected:Npn \__tag_check_info_closing_struct:n #1 %#1 struct num
123    {
124        \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
125        {
126            \msg_info:nnn { tag } {struct-show-closing} {#1}
127        }
128    }
129
130 \cs_generate_variant:Nn \__tag_check_info_closing_struct:n {o,x}
```

(*End definition for* `\__tag_check_info_closing_struct:n`.)

__tag_check_no_open_struct: This checks if there is an open structure. It should be used when trying to close a structure. It errors if false.

```
131 \cs_new_protected:Npn \__tag_check_no_open_struct:
132    {
133        \msg_error:nn { tag } {struct-faulty-nesting}
134    }
```

24

(*End definition for* `\__tag_check_no_open_struct:`.)

`\__tag_check_struct_used:n` This checks if a stashed structure has already been used.

```
135 \cs_new_protected:Npn \__tag_check_struct_used:n #1 %#1 label
136   {
137     \prop_get:cnNT
138       {g__tag_struct_\__tag_ref_value:enn{tagpdfstruct-#1}{tagstruct}{unknown}_prop}
139       {P}
140       \l_tmpa_tl
141       {
142         \msg_warning:nnn { tag } {struct-used-twice} {#1}
143       }
144   }
```

(*End definition for* `\__tag_check_struct_used:n`.)

6.3 Checks related to roles

`\__tag_check_add_tag_role:nn` This check is used when defining a new role mapping.

```
145 \cs_new_protected:Npn \__tag_check_add_tag_role:nn #1 #2 %#1 tag, #2 role
146   {
147     \tl_if_empty:nTF {#2}
148       {
149         \msg_error:nnn { tag } {role-missing} {#1}
150       }
151       {
152         \prop_get:NnNTF \g__tag_role_tags_NS_prop {#2} \l_tmpa_tl
153           {
154             \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
155               {
156                 \msg_info:nnnn { tag } {role-tag} {#1} {#2}
157               }
158           }
159           {
160             \msg_error:nnn { tag } {role-unknown} {#2}
161           }
162       }
163   }
```

Similar with a namespace

```
164 \cs_new_protected:Npn \__tag_check_add_tag_role:nnn #1 #2 #3 %#1 tag/NS, #2 role #3 namespace
165   {
166     \tl_if_empty:nTF {#2}
167       {
168         \msg_error:nnn { tag } {role-missing} {#1}
169       }
170       {
171         \prop_get:cnNTF { g__tag_role_NS_#3_prop } {#2} \l_tmpa_tl
172           {
173             \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
174               {
175                 \msg_info:nnnn { tag } {role-tag} {#1} {#2/#3}
176               }
177           }
```

```
178             {
179                 \msg_error:nnn { tag } {role-unknown} {#2/#3}
180             }
181         }
182     }
```

(*End definition for* `\__tag_check_add_tag_role:nn`.)

6.4 Check related to mc-chunks

`\__tag_check_mc_if_nested:`
`\__tag_check_mc_if_open:`

Two tests if a mc is currently open. One for the true (for begin code), one for the false part (for end code).

```
183 \cs_new_protected:Npn \__tag_check_mc_if_nested:
184     {
185         \__tag_mc_if_in:T
186             {
187                 \msg_warning:nnx { tag } {mc-nested} { \__tag_get_mc_abs_cnt: }
188             }
189     }
190
191 \cs_new_protected:Npn \__tag_check_mc_if_open:
192     {
193         \__tag_mc_if_in:F
194             {
195                 \msg_warning:nnx { tag } {mc-not-open} { \__tag_get_mc_abs_cnt: }
196             }
197     }
```

(*End definition for* `\__tag_check_mc_if_nested:` *and* `\__tag_check_mc_if_open:`.)

`\__tag_check_mc_pushed_popped:nn`

This creates an information message if mc's are pushed or popped. The first argument is a word (pushed or popped), the second the tag name. With larger log-level the stack is shown too.

```
198 \cs_new_protected:Npn \__tag_check_mc_pushed_popped:nn #1 #2
199     {
200         \int_compare:nNnT
201             { \l__tag_loglevel_int } =( 2 }
202             { \msg_info:nnx {tag}{mc-#1}{#2} }
203         \int_compare:nNnT
204             { \l__tag_loglevel_int } > { 2 }
205             {
206                 \msg_info:nnx {tag}{mc-#1}{#2}
207                 \seq_log:N \g__tag_mc_stack_seq
208             }
209     }
```

(*End definition for* `\__tag_check_mc_pushed_popped:nn`.)

`\__tag_check_mc_tag:N`

This checks if the mc has a (known) tag.

```
210 \cs_new_protected:Npn \__tag_check_mc_tag:N #1  %#1 is var with a tag name in it
211     {
212         \tl_if_empty:NT #1
213             {
214                 \msg_error:nnx { tag } {mc-tag-missing} { \__tag_get_mc_abs_cnt: }
```

```
215      }
216    \prop_if_in:NoF \g__tag_role_tags_NS_prop {#1}
217      {
218        \msg_warning:nnx { tag } {role-unknown-tag} {#1}
219      }
220    }
```

(*End definition for* `\__tag_check_mc_tag:N.`)

`\g__tag_check_mc_used_intarray`
`\__tag_check_init_mc_used:`
This variable holds the list of used mc numbers. Everytime we store a mc-number we will add one the relevant array index If everything is right at the end there should be only 1 until the max count of the mcid. 2 indicates that one mcid was used twice, 0 that we lost one. In engines other than luatex the total number of all intarray entries are restricted so we use only a rather small value of 65536, and we initialize the array only at first used, guarded by the log-level. This check is probably only needed for debugging. TODO does this really make sense to check? When can it happen??

```
221 \cs_new_protected:Npn \__tag_check_init_mc_used:
222    {
223      \intarray_new:Nn \g__tag_check_mc_used_intarray { 65536 }
224      \cs_gset_eq:NN \__tag_check_init_mc_used: \prg_do_nothing:
225    }
```

(*End definition for* `\g__tag_check_mc_used_intarray` *and* `\__tag_check_init_mc_used:.`)

`\__tag_check_mc_used:n`
This checks if a mc is used twice.

```
226 \cs_new_protected:Npn \__tag_check_mc_used:n #1 %#1 mcid abscnt
227    {
228      \int_compare:nNnT {\l__tag_loglevel_int} > { 2 }
229        {
230          \__tag_check_init_mc_used:
231          \intarray_gset:Nnn \g__tag_check_mc_used_intarray
232            {#1}
233            { \intarray_item:Nn \g__tag_check_mc_used_intarray {#1} + 1 }
234          \int_compare:nNnT
235            {
236              \intarray_item:Nn \g__tag_check_mc_used_intarray {#1}
237            }
238            >
239            { 1 }
240            {
241              \msg_warning:nnn { tag } {mc-used-twice} {#1}
242            }
243        }
244    }
```

(*End definition for* `\__tag_check_mc_used:n.`)

`\__tag_check_show_MCID_by_page:`
This allows to show the mc on a page. Currently unused.

```
245 \cs_new_protected:Npn \__tag_check_show_MCID_by_page:
246    {
247      \tl_set:Nx \l__tag_tmpa_tl
248        {
249          \__tag_ref_value_lastpage:nn
250            {abspage}
```

27

```
251        {-1}
252      }
253    \int_step_inline:nnnn {1}{1}
254      {
255        \l__tag_tmpa_tl
256      }
257      {
258        \seq_clear:N \l_tmpa_seq
259        \int_step_inline:nnnn
260          {1}
261          {1}
262          {
263            \__tag_ref_value_lastpage:nn
264              {tagmcabs}
265              {-1}
266          }
267          {
268            \int_compare:nT
269              {
270                \__tag_ref_value:enn
271                  {mcid-####1}
272                  {tagabspage}
273                  {-1}
274                =
275                ##1
276              }
277              {
278                \seq_gput_right:Nx \l_tmpa_seq
279                  {
280                    Page##1-####1-
281                    \__tag_ref_value:enn
282                      {mcid-####1}
283                      {tagmcid}
284                      {-1}
285                  }
286              }
287          }
288        \seq_show:N \l_tmpa_seq
289      }
290  }
```

(*End definition for* `\__tag_check_show_MCID_by_page:`.)

6.5 Checks related to the state of MC on a page or in a split stream

The following checks are currently only usable in generic mode as they rely on the marks defined in the mc-generic module. They are used to detect if a mc-chunk has been split by a page break or similar and additional end/begin commands are needed.

__tag_check_mc_in_galley_p:
__tag_check_mc_in_galley:*TF*

At first we need a test to decide if `\tag_mc_begin:n` (tmb) and `\tag_mc_end:` (tme) has been used at all on the current galley. As each command issues two slightly different marks we can do it by comparing firstmarks and botmarks. The test assumes that

the marks have been already mapped into the sequence with `\@@_mc_get_marks:`. As `\seq_if_eq:NNTF` doesn't exist we use the tl-test.

```
291 \prg_new_conditional:Npnn \__tag_check_if_mc_in_galley: { T,F,TF }
292   {
293     \tl_if_eq:NNTF \l__tag_mc_firstmarks_seq \l__tag_mc_botmarks_seq
294       { \prg_return_false: }
295       { \prg_return_true: }
296   }
```

(*End definition for* `\__tag_check_mc_in_galley:TF`.)

`\_tag_check_if_mc_tmb_missing_p:`
`\_tag_check_if_mc_tmb_missing:`*TF*

This checks if a extra top mark ("extra-tmb") is needed. According to the analysis this the case if the firstmarks start with `e-` or `b+`. Like above we assume that the marks content is already in the seq's.

```
297 \prg_new_conditional:Npnn \__tag_check_if_mc_tmb_missing: { T,F,TF }
298   {
299     \bool_if:nTF
300       {
301         \str_if_eq_p:ee {\seq_item:Nn \l__tag_mc_firstmarks_seq {1}}{e-}
302         ||
303         \str_if_eq_p:ee {\seq_item:Nn \l__tag_mc_firstmarks_seq {1}}{b+}
304       }
305       { \prg_return_true: }
306       { \prg_return_false: }
307   }
```

(*End definition for* `\__tag_check_if_mc_tmb_missing:TF`.)

`\_tag_check_if_mc_tme_missing_p:`
`\_tag_check_if_mc_tme_missing:`*TF*

This checks if a extra bottom mark ("extra-tme") is needed. According to the analysis this the case if the botmarks starts with `b+`. Like above we assume that the marks content is already in the seq's.

```
308 \prg_new_conditional:Npnn \__tag_check_if_mc_tme_missing: { T,F,TF }
309   {
310     \str_if_eq:eeTF {\seq_item:Nn \l__tag_mc_botmarks_seq {1}}{b+}
311       { \prg_return_true: }
312       { \prg_return_false: }
313   }
```

(*End definition for* `\__tag_check_if_mc_tme_missing:TF`.)

```
314 ⟨/package⟩
```

```
315 ⟨*debug⟩
```

Code for tagpdf-debug. This will probably change over time. At first something for the mc commands.

```
316 \msg_new:nnn { tag / debug } {mc-begin} { MC~begin~#1~with~options:~\tl_to_str:n{#2}~[\msg_lir
317 \msg_new:nnn { tag / debug } {mc-end}   { MC~end~#1~[\msg_line_context:] }
318
319 \cs_new_protected:Npn \__tag_debug_mc_begin_insert:n #1
320   {
321     \int_compare:nNnT { \l__tag_loglevel_int } > {0}
322       {
323         \msg_note:nnnn { tag / debug } {mc-begin} {inserted} { #1 }
324       }
```

```
325  }
326  \cs_new_protected:Npn \__tag_debug_mc_begin_ignore:n #1
327  {
328    \int_compare:nNnT { \l__tag_loglevel_int } > {0}
329      {
330        \msg_note:nnnn { tag / debug } {mc-begin} {ignored} { #1 }
331      }
332  }
333  \cs_new_protected:Npn \__tag_debug_mc_end_insert:
334  {
335    \int_compare:nNnT { \l__tag_loglevel_int } > {0}
336      {
337        \msg_note:nnn { tag / debug } {mc-end} {inserted}
338      }
339  }
340  \cs_new_protected:Npn \__tag_debug_mc_end_ignore:
341  {
342    \int_compare:nNnT { \l__tag_loglevel_int } > {0}
343      {
344        \msg_note:nnn { tag / debug } {mc-end } {ignored}
345      }
346  }
```
And now something for the structures
```
347  \msg_new:nnn { tag / debug } {struct-begin}
348    {
349      Struct~\tag_get:n{struct_num}~begin~#1~with~options:~\tl_to_str:n{#2}~[\msg_line_context:]
350    }
351  \msg_new:nnn { tag / debug } {struct-end}
352    {
353      Struct~end~#1~[\msg_line_context:]
354    }
355  \msg_new:nnn { tag / debug } {struct-end-wrong}
356    {
357      Struct~end~'#1'~doesn't~fit~start~'#2'~[\msg_line_context:]
358    }
359
360  \cs_new_protected:Npn \__tag_debug_struct_begin_insert:n #1
361  {
362    \int_compare:nNnT { \l__tag_loglevel_int } > {0}
363      {
364        \msg_note:nnnn { tag / debug } {struct-begin} {inserted} { #1 }
365        \seq_log:N \g__tag_struct_tag_stack_seq
366      }
367  }
368  \cs_new_protected:Npn \__tag_debug_struct_begin_ignore:n #1
369  {
370    \int_compare:nNnT { \l__tag_loglevel_int } > {0}
371      {
372        \msg_note:nnnn { tag / debug } {struct-begin } {ignored} { #1 }
373      }
374  }
375  \cs_new_protected:Npn \__tag_debug_struct_end_insert:
376  {
377    \int_compare:nNnT { \l__tag_loglevel_int } > {0}
```

```
378        {
379            \msg_note:nnn { tag / debug } {struct-end} {inserted}
380            \seq_log:N \g__tag_struct_tag_stack_seq
381        }
382    }
383 \cs_new_protected:Npn \__tag_debug_struct_end_ignore:
384    {
385        \int_compare:nNnT { \l__tag_loglevel_int } > {0}
386        {
387            \msg_note:nnn { tag / debug } {struct-end } {ignored}
388        }
389    }
390 \cs_new_protected:Npn \__tag_debug_struct_end_check:n #1
391    {
392        \int_compare:nNnT { \l__tag_loglevel_int } > {0}
393        {
394            \seq_get:NNT \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
395            {
396                \str_if_eq:eeF
397                {#1}
398                {\exp_last_unbraced:NV\use_i:nn \l__tag_tmpa_tl}
399                {
400                    \msg_warning:nnxx { tag/debug }{ struct-end-wrong }
401                    {#1}
402                    {\exp_last_unbraced:NV\use_i:nn \l__tag_tmpa_tl}
403                }
404            }
405        }
406    }
407
408 ⟨/debug⟩
```

Part II

The **tagpdf-user** module
Code related to LaTeX2e user
commands and document commands
Part of the tagpdf package

1 Setup commands

`\tagpdfsetup` `\tagpdfsetup{⟨key val list⟩}`

This is the main setup command to adapt the behaviour of tagpdf. It can be used in the preamble and in the document (but not all keys make sense there).

`activate␣(setup-key)` And additional setup key which combine the other activate keys `activate-mc`, `activate-tree`, `activate-struct` and additionally add a document structure.

`\tag_tool:n` `\tag_tool:n{⟨key val⟩}`
`\tagtool`

The tagging of basic document elements will require a variaty of small commands to configure and adapt the tagging. This command will collect them under a command interface. The argument is *one* key-value like string. This is work in progress and both syntax, known arguments and implementation can change!

2 Commands related to mc-chunks

`\tagmcbegin` `\tagmcbegin {⟨key-val⟩}`
`\tagmcend` `\tagmcend`
`\tagmcuse` `\tagmcuse{⟨label⟩}`

These are wrappers around `\tag_mc_begin:n`, `\tag_mc_end:` and `\tag_mc_use:n`. The commands and their argument are documentated in the **tagpdf-mc** module. In difference to the expl3 commands, `\tagmcbegin` issues also an `\ignorespaces`, and `\tagmcend` will issue in horizontal mode an `\unskip`.

`\tagmcifinTF` `\tagmcifin {⟨true code⟩}{⟨false code⟩}`

This is a wrapper around `\tag_mc_if_in:TF`. and tests if an mc is open or not. It is mostly of importance for pdflatex as lualatex doesn't mind much if a mc tag is not correctly closed. Unlike the expl3 command it is not expandable.

The command is probably not of much use and will perhaps disappear in future versions. It normally makes more sense to push/pop an mc-chunk.

3 Commands related to structures

`\tagstructbegin`	`\tagstructbegin {⟨key-val⟩}`
`\tagstructend`	`\tagstructend`
`\tagstructuse`	`\tagstructuse{⟨label⟩}`

These are direct wrappers around `\tag_struct_begin:n`, `\tag_struct_end:` and `\tag_struct_use:n`. The commands and their argument are documentated in the `tagpdf-struct` module.

4 Debugging

`\ShowTagging` `\ShowTagging {⟨key-val⟩}`

This is a generic function to output various debugging helps. It not necessarily stops the compilation. The keys and their function are described below.

`mc-data␣(show-key)` `mc-data = ⟨number⟩`

This key is (currently?) relevant for lua mode only. It shows the data of all mc-chunks created so far. It is accurate only after shipout (and perhaps a second compilation), so typically should be issued after a newpage. The value is a positive integer and sets the first mc-shown. If no value is given, 1 is used and so all mc-chunks created so far are shown.

`mc-current␣(show-key)` `mc-current`

This key shows the number and the tag of the currently open mc-chunk. If no chunk is open it shows only the state of the abs count. It works in all mode, but the output in luamode looks different.

`mc-marks␣(show-key)` `mc-marks = show|use`

This key helps to debug the page marks. It should only be used at shipout in header or footer.

`struct-stack␣(show-key)` `struct-stack = log|show`

This key shows the current structure stack. With `log` the info is only written to the log-file, `show` stops the compilation and shows on the terminal. If no value is used, then the default is `show`.

5 Extension commands

The following commands and code parts are not core command of tagpdf. They either provide work-arounds for missing functionality elsewhere, or do a first step to apply tagpdf commands to document commands.

The commands and keys should be view as experimental!

This part will be regularly revisited to check if the code should go to a better place or can be improved and so can change easily.

5.1 Fake space

\pdffakespace (lua-only) This provides a lua-version of the \pdffakespace primitive of pdftex.

5.2 Paratagging

This is a first try to make use of the new paragraph hooks in a current LaTeX to automate the tagging of paragraph. It requires sane paragraph nesting, faulty code, e.g. a missing \par at the end of a low-level vbox can highly confuse the tagging. The tags should be carefully checked if this is used.

paratagging␣(setup-key)   paratagging = true|false
paratagging-show␣(setup-key) paratagging-show = true|false

This keys can be used in \tagpdfsetup and enable/disable paratagging. paratagging-show puts small red numbers at the begin and end of a paragraph. This is meant as a debugging help. The number are boxes and have a (tiny) height, so they can affect typesetting.

\tagpdfparaOn These commands allow to enable/disable para tagging too and are a bit faster then
\tagpdfparaOff \tagpdfsetup. But I'm not sure if the names are good.

\tagpdfsuppressmarks This command allows to suppress the creation of the marks. It takes an argument which should normally be one of the mc-commands, puts a group around it and suppress the marks creation in this group. This command should be used if the begin and end command are at different boxing levels. E.g.

```
\@hangfrom
 {
  \tagstructbegin{tag=H1}%
  \tagmcbegin    {tag=H1}%
  #2
 }
 {#3\tagpdfsuppressmarks{\tagmcend}\tagstructend}%
```

5.3 Header and footer

Header and footer are automatically excluded from tagging. This can be disabled with the following key. If some real content is in the header and footer, tagging must be restarted there explicitly. The key accepts the values `true` which surrounds the header with an artifact mc-chunk, `false` which disables the automatic tagging, and `pagination` which additionally adds an artifact structure with an pagination attribute.

exclude-header-footer␣(setup-key) exclude-header-footer = true|false|pagination

5.4 Link tagging

Links need a special structure and cross reference system. This is added through hooks of the l3pdfannot module and will work automatically if tagging is activated.

Links should (probably) have an alternative text in the Contents key. It is unclear which text this should be and how to get it. Currently the code simply adds the fix texts `url` and `ref`. Another text can be added by changing the dictionary value:

```
\pdfannot_dict_put:nnn
{ link/GoTo }
{ Contents }
{ (ref) }
```

6 User commands and extensions of document commands

```
1  ⟨@@=tag⟩
2  ⟨*header⟩
3  \ProvidesExplPackage {tagpdf-user} {2023-03-07} {0.98e}
4    {tagpdf - user commands}
5  ⟨/header⟩
```

7 Setup and preamble commands

\tagpdfsetup

```
6  ⟨base⟩\NewDocumentCommand \tagpdfsetup { m }{}
7  ⟨*package⟩
8  \RenewDocumentCommand \tagpdfsetup { m }
9    {
10     \keys_set:nn { __tag / setup } { #1 }
11    }
12 ⟨/package⟩
```

(*End definition for* \tagpdfsetup*. This function is documented on page 32.*)

\tag_tool:n
\tagtool

This is a first definition of the tool command. Currently it uses key-val, but this should be probably be flattened to speed it up.

```
13 ⟨base⟩\cs_new_protected:Npn\tag_tool:n #1 {}
14 ⟨base⟩\cs_set_eq:NN\tagtool\tag_tool:n
15 ⟨*package⟩
16 \cs_set_protected:Npn\tag_tool:n #1
17   {
18     \tag_if_active:T { \keys_set:nn {tag / tool}{#1} }
19   }
20 \cs_set_eq:NN\tagtool\tag_tool:n
21 ⟨/package⟩
```

(*End definition for* \tag_tool:n *and* \tagtool*. These functions are documented on page 32.*)

8 Commands for the mc-chunks

`\tagmcbegin`
`\tagmcend`
`\tagmcuse`

```
22 ⟨*base⟩
23 \NewDocumentCommand \tagmcbegin { m }
24   {
25     \tag_mc_begin:n {#1}
26   }
27
28
29 \NewDocumentCommand \tagmcend {  }
30   {
31     \tag_mc_end:
32   }
33
34 \NewDocumentCommand \tagmcuse { m }
35   {
36     \tag_mc_use:n {#1}
37   }
38 ⟨/base⟩
```

(*End definition for* `\tagmcbegin`*,* `\tagmcend`*, and* `\tagmcuse`*. These functions are documented on page 32.*)

`\tagmcifinTF` This is a wrapper around `\tag_mc_if_in:` and tests if an mc is open or not. It is mostly of importance for pdflatex as lualatex doesn't mind much if a mc tag is not correctly closed. Unlike the expl3 command it is not expandable.

```
39 ⟨*package⟩
40 \NewDocumentCommand \tagmcifinTF { m m }
41   {
42     \tag_mc_if_in:TF { #1 } { #2 }
43   }
44 ⟨/package⟩
```

(*End definition for* `\tagmcifinTF`*. This function is documented on page 32.*)

9 Commands for the structure

`\tagstructbegin`
`\tagstructend`
`\tagstructuse`

These are structure related user commands. There are direct wrapper around the expl3 variants.

```
45 ⟨*base⟩
46 \NewDocumentCommand \tagstructbegin { m }
47   {
48     \tag_struct_begin:n {#1}
49   }
50
51 \NewDocumentCommand \tagstructend {  }
52   {
53    \tag_struct_end:
54   }
55
56 \NewDocumentCommand \tagstructuse { m }
57   {
```

```
58        \tag_struct_use:n {#1}
59      }
60  ⟨/base⟩
```

(*End definition for* \tagstructbegin *,* \tagstructend *, and* \tagstructuse*. These functions are documented on page 33.*)

10 Debugging

\ShowTagging   This is a generic command for various show commands. It takes a keyval list, the various keys are implemented below.

```
61  ⟨∗package⟩
62  \NewDocumentCommand\ShowTagging { m }
63    {
64      \keys_set:nn { __tag / show }{ #1}
65
66    }
```

(*End definition for* \ShowTagging*. This function is documented on page 33.*)

mc-data␣(show-key)   This key is (currently?) relevant for lua mode only. It shows the data of all mc-chunks created so far. It is accurate only after shipout, so typically should be issued after a newpage. With the optional argument the minimal number can be set.

```
67  \keys_define:nn { __tag / show }
68    {
69      mc-data .code:n =
70        {
71          \sys_if_engine_luatex:T
72            {
73              \lua_now:e{ltx.__tag.trace.show_all_mc_data(#1,\__tag_get_mc_abs_cnt:,0)}
74            }
75        }
76      ,mc-data .default:n = 1
77    }
78
```

(*End definition for* mc-data (show-key)*. This function is documented on page 33.*)

mc-current␣(show-key)   This shows some info about the current mc-chunk. It works in generic and lua-mode.

```
79  \keys_define:nn { __tag / show }
80    { mc-current .code:n =
81        {
82          \bool_if:NTF \g__tag_mode_lua_bool
83            {
84              \sys_if_engine_luatex:T
85                {
86                  \int_compare:nNnTF
87                    { -2147483647 }
88                      =
89                    {
90                      \lua_now:e
91                        {
92                          tex.print
```

```
93                      (tex.getattribute
94                        (luatexbase.attributes.g__tag_mc_cnt_attr))
95                  }
96              }
97              {
98                \lua_now:e
99                  {
100                   ltx.__tag.trace.log
101                     (
102                       "mc-current:~no~MC~open,~current~abscnt
103                        =\__tag_get_mc_abs_cnt:"
104                       ,0
105                     )
106                   texio.write_nl("")
107                 }
108             }
109             {
110               \lua_now:e
111                 {
112                   ltx.__tag.trace.log
113                     (
114                       "mc-current:~abscnt=\__tag_get_mc_abs_cnt:=="
115                        ..
116                        tex.getattribute(luatexbase.attributes.g__tag_mc_cnt_attr)
117                        ..
118                        "~=>tag="
119                        ..
120                        tostring
121                          (ltx.__tag.func.get_tag_from
122                            (tex.getattribute
123                              (luatexbase.attributes.g__tag_mc_type_attr)))
124                        ..
125                        "="
126                        ..
127                        tex.getattribute
128                         (luatexbase.attributes.g__tag_mc_type_attr)
129                       ,0
130                     )
131                   texio.write_nl("")
132                 }
133           }
134         }
135       }
136       {
137        \msg_note:nn{ tag }{ mc-current }
138       }
139     }
140   }
```

(End definition for `mc-current` *(show-key). This function is documented on page 33.)*

mc-marks␣(show-key) It maps the mc-marks into the sequences and then shows them. This allows to inspect the first and last mc-Mark on a page. It should only be used in the shipout (header/footer).

```
141 \keys_define:nn { __tag / show }
```

```
142    {
143      mc-marks .choice: ,
144      mc-marks / show .code:n =
145        {
146          \__tag_mc_get_marks:
147          \__tag_check_if_mc_in_galley:TF
148            {
149             \iow_term:n {Marks~from~this~page:~}
150            }
151            {
152              \iow_term:n {Marks~from~a~previous~page:~}
153            }
154          \seq_show:N \l__tag_mc_firstmarks_seq
155          \seq_show:N \l__tag_mc_botmarks_seq
156          \__tag_check_if_mc_tmb_missing:T
157            {
158              \iow_term:n {BDC~missing~on~this~page!}
159            }
160          \__tag_check_if_mc_tme_missing:T
161            {
162              \iow_term:n {EMC~missing~on~this~page!}
163            }
164        },
165      mc-marks / use .code:n =
166        {
167          \__tag_mc_get_marks:
168          \__tag_check_if_mc_in_galley:TF
169          { Marks~from~this~page:~}
170          { Marks~from~a~previous~page:~}
171          \seq_use:Nn \l__tag_mc_firstmarks_seq {,~}\quad
172          \seq_use:Nn \l__tag_mc_botmarks_seq {,~}\quad
173          \__tag_check_if_mc_tmb_missing:T
174            {
175              BDC~missing~
176            }
177          \__tag_check_if_mc_tme_missing:T
178            {
179              EMC~missing
180            }
181        },
182     mc-marks .default:n = show
183    }
```

(*End definition for* mc-marks *(show-key). This function is documented on page 33.*)

```
184 \keys_define:nn { __tag / show }
185   {
186      struct-stack .choice:
187     ,struct-stack / log .code:n = \seq_log:N \g__tag_struct_tag_stack_seq
188     ,struct-stack / show .code:n = \seq_show:N \g__tag_struct_tag_stack_seq
189     ,struct-stack .default:n = show
190   }
```

(*End definition for* struct-stack *(show-key). This function is documented on page 33.*)

11 Commands to extend document commands

The following commands and code parts are not core command of tagpdf. The either provide work arounds for missing functionality elsewhere, or do a first step to apply tagpdf commands to document commands. This part should be regularly revisited to check if the code should go to a better place or can be improved.

11.1 Document structure

__tag_add_document_structure:n

activate␣(setup-key)

```
191 \cs_new_protected:Npn \__tag_add_document_structure:n #1
192   {
193     \hook_gput_code:nnn{begindocument}{tagpdf}{\tagstructbegin{tag=#1}}
194     \hook_gput_code:nnn{tagpdf/finish/before}{tagpdf}{\tagstructend}
195   }
196 \keys_define:nn { __tag / setup}
197   {
198     activate   .code:n =
199       {
200         \keys_set:nn { __tag / setup }
201           { activate-mc,activate-tree,activate-struct }
202         \__tag_add_document_structure:n {#1}
203       },
204     activate .default:n = Document
205   }
```

(*End definition for* __tag_add_document_structure:n *and* activate (setup-key)*. This function is documented on page 32.*)

11.2 Structure destinations

In TeXlive 2022 pdftex and luatex will offer support for structure destinations. The pdfmanagement has already backend support. We activate them if the prerequisites are there: structures should be activated, the code in the pdfmanagement must be there. Structure destinations are actually PDF 2.0 only but they don't harm in older PDF and can improve html export.

```
206 \AddToHook{begindocument/before}
207   {
208     \bool_lazy_all:nT
209       {
210         { \g__tag_active_struct_dest_bool }
211         { \g__tag_active_struct_bool }
212         { \cs_if_exist_p:N \pdf_activate_structure_destination: }
213       }
214       {
215         \tl_set:Nn \l_pdf_current_structure_destination_tl { __tag/struct/\g__tag_struct_stack
216         \pdf_activate_structure_destination:
217       }
218   }
```

11.3 Fake space

\pdffakespace We need a luatex variant for `\pdffakespace`. This should probably go into the kernel at some time.

```
219 \sys_if_engine_luatex:T
220   {
221     \NewDocumentCommand\pdffakespace { }
222       {
223         \__tag_fakespace:
224       }
225   }
```

(*End definition for* `\pdffakespace`. *This function is documented on page 34.*)

11.4 Paratagging

The following are some simple commands to enable/disable paratagging. Probably one should add some checks if we are already in a paragraph.

\l__tag_para_bool
\l__tag_para_flattened_bool
\l__tag_para_show_bool
\g__tag_para_begin_int
\g__tag_para_end_int
\g__tag_para_main_begin_int
\g__tag_para_main_end_int
\l__tag_para_tag_default_tl
\l__tag_para_tag_tl
\l__tag_para_main_tag_tl

At first some variables.

```
226 \bool_new:N \l__tag_para_bool
227 ⟨/package⟩
228 ⟨base⟩\bool_new:N \l__tag_para_flattened_bool
229 ⟨*package⟩
230 \bool_new:N \l__tag_para_show_bool
231 \int_new:N  \g__tag_para_begin_int
232 \int_new:N  \g__tag_para_end_int
233 \int_new:N  \g__tag_para_main_begin_int
234 \int_new:N  \g__tag_para_main_end_int
235 \tl_new:N   \l__tag_para_tag_default_tl
236 \tl_set:Nn  \l__tag_para_tag_default_tl { text }
237 \tl_new:N   \l__tag_para_tag_tl
238 \tl_set:Nn  \l__tag_para_tag_tl { \l__tag_para_tag_default_tl }
239 \tl_new:N   \l__tag_para_main_tag_tl
240 \tl_set:Nn  \l__tag_para_main_tag_tl {text-unit}
```

(*End definition for* `\l__tag_para_bool` *and others.*)

paratagging␣(setup-key)
paratagging-show␣(setup-key)
paratag␣(setup-key)
paratag␣(tool-key)
unittag␣(tool-key)
para-flattened␣(tool-key)

These keys enable/disable locally paratagging, and the debug modus. It can affect the typesetting if `paratagging-show` is used. The small numbers are boxes and they have a (small) height. The `paratag` key sets the tag used by the next automatic paratagging, it can also be changed with `\tag_tool:n`

```
241 \keys_define:nn { __tag / setup }
242   {
243     paratagging      .bool_set:N = \l__tag_para_bool,
244     paratagging-show .bool_set:N = \l__tag_para_show_bool,
245     paratag          .tl_set:N   = \l__tag_para_tag_tl
246   }
247 \keys_define:nn { tag / tool}
248   {
249     paratag .tl_set:N = \l__tag_para_tag_tl,
250     unittag .tl_set:N = \l__tag_para_main_tag_tl,
251     para-flattened .bool_set:N = \l__tag_para_flattened_bool
252   }
```

41

(*End definition for* paratagging (setup-key) *and others. These functions are documented on page [34](#).*)

This fills the para hooks with the needed code.

```
253 \cs_new_protected:Npn \__tag_check_para_begin_show:nn #1 #2
254 %#1 color, #2 prefix
255   {
256     \bool_if:NT \l__tag_para_show_bool
257       {
258         \tag_mc_begin:n{artifact}
259         \llap{\color_select:n{#1}\tiny#2\int_use:N\g__tag_para_begin_int\ }
260         \tag_mc_end:
261       }
262   }
263
264 \cs_new_protected:Npn \__tag_check_para_end_show:nn #1 #2
265 %#1  color, #2 prefix
266   {
267     \bool_if:NT \l__tag_para_show_bool
268       {
269         \tag_mc_begin:n{artifact}
270         \rlap{\color_select:n{#1}\tiny\ #2\int_use:N\g__tag_para_end_int}
271         \tag_mc_end:
272       }
273   }
274
275 \AddToHook{para/begin}
276   {
277     \bool_if:NT \l__tag_para_bool
278       {
279         \bool_if:NF \l__tag_para_flattened_bool
280           {
281             \int_gincr:N \g__tag_para_main_begin_int
282             \tag_struct_begin:n
283               {
284                 tag=\l__tag_para_main_tag_tl,
285               }
286           }
287         \int_gincr:N \g__tag_para_begin_int
288         \tag_struct_begin:n {tag=\l__tag_para_tag_tl}
289         \__tag_check_para_begin_show:nn {green}{}
290         \tag_mc_begin:n {}
291       }
292   }
293 \AddToHook{para/end}
294   {
295     \bool_if:NT \l__tag_para_bool
296       {
297         \int_gincr:N \g__tag_para_end_int
298         \tag_mc_end:
299         \__tag_check_para_end_show:nn {red}{}
300         \tag_struct_end:
301         \bool_if:NF \l__tag_para_flattened_bool
302           {
303             \int_gincr:N \g__tag_para_main_end_int
304             \tag_struct_end:
```

```
305              }
306           }
307        }
308    \AddToHook{enddocument/info}
309      {
310        \int_compare:nNnF {\g__tag_para_main_begin_int}={\g__tag_para_main_end_int}
311          {
312            \msg_error:nnxxx
313              {tag}
314              {para-hook-count-wrong}
315              {\int_use:N\g__tag_para_main_begin_int}
316              {\int_use:N\g__tag_para_main_end_int}
317              {text-unit}
318          }
319        \int_compare:nNnF {\g__tag_para_begin_int}={\g__tag_para_end_int}
320          {
321            \msg_error:nnxxx
322              {tag}
323              {para-hook-count-wrong}
324              {\int_use:N\g__tag_para_begin_int}
325              {\int_use:N\g__tag_para_end_int}
326              {text}
327          }
328      }
```

In generic mode we need the additional code from the ptagging tests.

```
329    \AddToHook{begindocument/before}
330    {
331        \@ifundefined{@mult@ptagging@hook}{\RequirePackage{output-patches-tmp-ltx}}{} %
332        \bool_if:NF \g__tag_mode_lua_bool
333          {
334            \cs_if_exist:NT \@kernel@before@footins
335              {
336                \tl_put_right:Nn \@kernel@before@footins
337                  { \__tag_add_missing_mcs_to_stream:Nn \footins {footnote} }
338                \tl_put_right:Nn \@kernel@before@cclv
339                  {
340                    \__tag_check_typeout_v:n {====>~In~\token_to_str:N \@makecol\c_space_tl\the\c@p
341                    \__tag_add_missing_mcs_to_stream:Nn \@cclv {main}
342                  }
343                \tl_put_right:Nn \@mult@ptagging@hook
344                  {
345                    \__tag_check_typeout_v:n {====>~In~\string\page@sofar}
346                    \process@cols\mult@firstbox
347                      {
348                        \__tag_add_missing_mcs_to_stream:Nn \count@ {multicol}
349                      }
350                    \__tag_add_missing_mcs_to_stream:Nn \mult@rightbox {multicol}
351                  }
352              }
353          }
354    }
355 ⟨/package⟩
```

\tagpdfparaOn   This two command switch para mode on and off. \tagpdfsetup could be used too but
\tagpdfparaOff

is longer. An alternative is \tag_tool:n{para=false}

```
356 ⟨base⟩\newcommand\tagpdfparaOn {}
357 ⟨base⟩\newcommand\tagpdfparaOff{}
358 ⟨∗package⟩
359 \renewcommand\tagpdfparaOn {\bool_set_true:N \l__tag_para_bool}
360 \renewcommand\tagpdfparaOff{\bool_set_false:N \l__tag_para_bool}
361 \keys_define:nn { tag / tool}
362   {
363     para .bool_set:N = \l__tag_para_bool,
364     para-flattened .bool_set:N = \l__tag_para_flattened_bool,
365   }
```

(*End definition for* \tagpdfparaOn *and* \tagpdfparaOff. *These functions are documented on page* *34.*)

\tagpdfsuppressmarks   This command allows to suppress the creation of the marks. It takes an argument which should normally be one of the mc-commands, puts a group around it and suppress the marks creation in this group. This command should be used if the begin and end command are at different boxing levels. E.g.

```
 \@hangfrom
  {
   \tagstructbegin{tag=H1}%
   \tagmcbegin    {tag=H1}%
   #2
  }
  {#3\tagpdfsuppressmarks{\tagmcend}\tagstructend}%
```

```
366 \NewDocumentCommand\tagpdfsuppressmarks{m}
367   {{\use:c{__tag_mc_disable_marks:} #1}}
```

(*End definition for* \tagpdfsuppressmarks. *This function is documented on page* *34.*)

11.5   Header and footer

Header and footer should normally be tagged as artifacts. The following code requires the new hooks. For now we allow to disable this function, but probably the code should always there at the end. TODO check if Pagination should be changeable.

```
368 \cs_new_protected:Npn\__tag_hook_kernel_before_head:{}
369 \cs_new_protected:Npn\__tag_hook_kernel_after_head:{}
370 \cs_new_protected:Npn\__tag_hook_kernel_before_foot:{}
371 \cs_new_protected:Npn\__tag_hook_kernel_after_foot:{}
372
373 \AddToHook{begindocument}
374   {
375     \cs_if_exist:NT \@kernel@before@head
376       {
377         \tl_put_right:Nn \@kernel@before@head {\__tag_hook_kernel_before_head:}
378         \tl_put_left:Nn  \@kernel@after@head  {\__tag_hook_kernel_after_head:}
379         \tl_put_right:Nn \@kernel@before@foot {\__tag_hook_kernel_before_foot:}
380         \tl_put_left:Nn  \@kernel@after@foot  {\__tag_hook_kernel_after_foot:}
381       }
382   }
383
384 \bool_new:N \g__tag_saved_in_mc_bool
```

```
385  \cs_new_protected:Npn \__tag_exclude_headfoot_begin:
386  {
387      \bool_set_false:N  \l__tag_para_bool
388      \bool_if:NTF \g__tag_mode_lua_bool
389        {
390         \tag_mc_end_push:
391        }
392        {
393           \bool_gset_eq:NN    \g__tag_saved_in_mc_bool \g__tag_in_mc_bool
394           \bool_gset_false:N \g__tag_in_mc_bool
395        }
396      \tag_mc_begin:n {artifact}
397  }
398  \cs_new_protected:Npn \__tag_exclude_headfoot_end:
399  {
400      \tag_mc_end:
401      \bool_if:NTF \g__tag_mode_lua_bool
402        {
403         \tag_mc_begin_pop:n{}
404        }
405        {
406           \bool_gset_eq:NN \g__tag_in_mc_bool\g__tag_saved_in_mc_bool
407        }
408  }
```

This version allows to use an Artifact structure

```
409  \__tag_attr_new_entry:nn {__tag/attr/pagination}{/O/Artifact/Type/Pagination}
410  \cs_new_protected:Npn \__tag_exclude_struct_headfoot_begin:n #1
411  {
412      \bool_set_false:N  \l__tag_para_bool
413      \bool_if:NTF \g__tag_mode_lua_bool
414        {
415         \tag_mc_end_push:
416        }
417        {
418           \bool_gset_eq:NN    \g__tag_saved_in_mc_bool \g__tag_in_mc_bool
419           \bool_gset_false:N \g__tag_in_mc_bool
420        }
421      \tag_struct_begin:n{tag=Artifact,attribute-class=__tag/attr/#1}
422      \tag_mc_begin:n {artifact=#1}
423  }
424
425  \cs_new_protected:Npn \__tag_exclude_struct_headfoot_end:
426  {
427      \tag_mc_end:
428      \tag_struct_end:
429      \bool_if:NTF \g__tag_mode_lua_bool
430        {
431         \tag_mc_begin_pop:n{}
432        }
433        {
434           \bool_gset_eq:NN \g__tag_in_mc_bool\g__tag_saved_in_mc_bool
435        }
436  }
```

And now the keys

```
437 \keys_define:nn { __tag / setup }
438   {
439     exclude-header-footer .choice:,
440     exclude-header-footer / true .code:n =
441      {
442        \cs_set_eq:NN \__tag_hook_kernel_before_head: \__tag_exclude_headfoot_begin:
443        \cs_set_eq:NN \__tag_hook_kernel_before_foot: \__tag_exclude_headfoot_begin:
444        \cs_set_eq:NN \__tag_hook_kernel_after_head:  \__tag_exclude_headfoot_end:
445        \cs_set_eq:NN \__tag_hook_kernel_after_foot:  \__tag_exclude_headfoot_end:
446      },
447     exclude-header-footer / pagination .code:n =
448      {
449        \cs_set:Nn \__tag_hook_kernel_before_head: { \__tag_exclude_struct_headfoot_begin:n {pa
450        \cs_set:Nn \__tag_hook_kernel_before_foot: { \__tag_exclude_struct_headfoot_begin:n {pa
451        \cs_set_eq:NN \__tag_hook_kernel_after_head:  \__tag_exclude_struct_headfoot_end:
452        \cs_set_eq:NN \__tag_hook_kernel_after_foot:  \__tag_exclude_struct_headfoot_end:
453      },
454     exclude-header-footer / false .code:n =
455      {
456        \cs_set_eq:NN \__tag_hook_kernel_before_head: \prg_do_nothing:
457        \cs_set_eq:NN \__tag_hook_kernel_before_foot: \prg_do_nothing:
458        \cs_set_eq:NN \__tag_hook_kernel_after_head:  \prg_do_nothing:
459        \cs_set_eq:NN \__tag_hook_kernel_after_foot:  \prg_do_nothing:
460      },
461    exclude-header-footer .default:n = true,
462    exclude-header-footer .initial:n = true
463   }
```

(*End definition for* `exclude-header-footer` *(setup-key). This function is documented on page 34.*)

11.6   Links

We need to close and reopen mc-chunks around links. Currently we handle URI and GoTo (internal) links. Links should have an alternative text in the Contents key. It is unclear which text this should be and how to get it.

```
464 \hook_gput_code:nnn
465   {pdfannot/link/URI/before}
466   {tagpdf}
467   {
468     \tag_mc_end_push:
469     \tag_struct_begin:n { tag=Link }
470     \tag_mc_begin:n { tag=Link }
471     \pdfannot_dict_put:nnx
472       { link/URI }
473       { StructParent }
474       { \tag_struct_parent_int: }
475   }
476
477 \hook_gput_code:nnn
478   {pdfannot/link/URI/after}
479   {tagpdf}
```

```
480    {
481      \tag_struct_insert_annot:xx {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
482      \tag_mc_end:
483      \tag_struct_end:
484      \tag_mc_begin_pop:n{}
485    }
486
487  \hook_gput_code:nnn
488    {pdfannot/link/GoTo/before}
489    {tagpdf}
490    {
491      \tag_mc_end_push:
492      \tag_struct_begin:n{tag=Link}
493      \tag_mc_begin:n{tag=Link}
494      \pdfannot_dict_put:nnx
495        { link/GoTo }
496        { StructParent }
497        { \tag_struct_parent_int: }
498    }
499
500  \hook_gput_code:nnn
501    {pdfannot/link/GoTo/after}
502    {tagpdf}
503    {
504      \tag_struct_insert_annot:xx {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
505      \tag_mc_end:
506      \tag_struct_end:
507      \tag_mc_begin_pop:n{}
508
509    }
510
511  % "alternative descriptions " for PAX3. How to get better text here??
512  \pdfannot_dict_put:nnn
513    { link/URI }
514    { Contents }
515    { (url) }
516
517  \pdfannot_dict_put:nnn
518    { link/GoTo }
519    { Contents }
520    { (ref) }
521
    </package>
```

Part III
The **tagpdf-tree** module
Commands trees and main dictionaries
Part of the tagpdf package

```
1 ⟨@@=tag⟩
2 ⟨*header⟩
3 \ProvidesExplPackage {tagpdf-tree-code} {2023-03-07} {0.98e}
4   {part of tagpdf - code related to writing trees and dictionaries to the pdf}
5 ⟨/header⟩
```

1 Trees, pdfmanagement and finalization code

The code to finish the structure is in a hook. This will perhaps at the end be a kernel hook. TODO check right place for the code The pdfmanagement code is the kernel hook after shipout/lastpage so all code affecting it should be before. Objects can be written later, at least in pdf mode.

```
6  ⟨*package⟩
7  \hook_gput_code:nnn{begindocument}{tagpdf}
8    {
9      \bool_if:NT \g__tag_active_tree_bool
10       {
11         \sys_if_output_pdf:TF
12           {
13             \AddToHook{enddocument/end} { \__tag_finish_structure: }
14           }
15           {
16             \AddToHook{shipout/lastpage} { \__tag_finish_structure: }
17           }
18       }
19    }
```

1.1 Check structure

`\__tag_tree_final_checks:`

```
20 \cs_new_protected:Npn \__tag_tree_final_checks:
21   {
22     \int_compare:nNnF {\seq_count:N\g__tag_struct_stack_seq}={1}
23       {
24         \msg_warning:nn {tag}{tree-struct-still-open}
25         \int_step_inline:nnn{2}{\seq_count:N\g__tag_struct_stack_seq}
26         {\tag_struct_end:}
27       }
28   }
```

(*End definition for* `\__tag_tree_final_checks:`.)

1.2 Catalog: MarkInfo and StructTreeRoot

The StructTreeRoot and the MarkInfo entry must be added to the catalog. We do it late so that we can win, but before the pdfmanagement hook.

`__tag/struct/0` This is the object for the root object, the StructTreeRoot

```
29 \pdf_object_new:n { __tag/struct/0 }
```

(*End definition for* `__tag/struct/0`.)

```
30 \hook_gput_code:nnn{shipout/lastpage}{tagpdf}
31   {
32     \bool_if:NT \g__tag_active_tree_bool
33       {
34         \pdfmanagement_add:nnn { Catalog / MarkInfo } { Marked } { true }
35         \pdfmanagement_add:nnx
36           { Catalog }
37           { StructTreeRoot }
38           { \pdf_object_ref:n { __tag/struct/0 } }
39       }
40   }
```

1.3 Writing the IDtree

The ID are currently quite simple: every structure has an ID build from the prefix ID together with the structure number padded with enough zeros to that we get directly an lexical order. We ship them out in bundles At first a seq to hold the references for the kids

`\g__tag_tree_id_pad_int`

```
41 \int_new:N\g__tag_tree_id_pad_int
```

(*End definition for* `\g__tag_tree_id_pad_int`.)
    Now we get the needed padding

```
42 \cs_generate_variant:Nn \tl_count:n {e}
43 \hook_gput_code:nnn{begindocument}{tagpdf}
44   {
45     \int_gset:Nn\g__tag_tree_id_pad_int
46       {\tl_count:e { \__tag_ref_value_lastpage:nn{tagstruct}{1000}}+1}
47   }
48
```

This is the main code to write the tree it basically splits the existing structure numbers in chunks of length 50 TODO consider is 50 is a good length.

```
49 \cs_new_protected:Npn \__tag_tree_write_idtree:
50   {
51     \tl_clear:N \l__tag_tmpa_tl
52     \tl_clear:N \l__tag_tmpb_tl
53     \int_zero:N \l__tag_tmpa_int
54     \int_step_inline:nn {\c@g__tag_struct_abs_int}
55       {
56         \int_incr:N\l__tag_tmpa_int
57         \tl_put_right:Nx \l__tag_tmpa_tl
58           {
59             \__tag_struct_get_id:n{##1}~\pdf_object_ref:n{__tag/struct/##1}~
```

```
60              }
61          \int_compare:nNnF {\l__tag_tmpa_int}<{50} %
62            {
63              \pdf_object_unnamed_write:nx {dict}
64                { /Limits~[\__tag_struct_get_id:n{##1-\l__tag_tmpa_int+1}~\__tag_struct_get_id:
65                  /Names~[\l__tag_tmpa_tl]
66                }
67              \tl_put_right:Nx\l__tag_tmpb_tl {\pdf_object_ref_last:\c_space_tl}
68              \int_zero:N \l__tag_tmpa_int
69              \tl_clear:N \l__tag_tmpa_tl
70            }
71        }
72      \tl_if_empty:NF \l__tag_tmpa_tl
73        {
74          \pdf_object_unnamed_write:nx {dict}
75            {
76              /Limits~
77                [\__tag_struct_get_id:n{\c@g__tag_struct_abs_int-\l__tag_tmpa_int+1}~
78                  \__tag_struct_get_id:n{\c@g__tag_struct_abs_int}]
79              /Names~[\l__tag_tmpa_tl]
80            }
81          \tl_put_right:Nx\l__tag_tmpb_tl {\pdf_object_ref_last:}
82        }
83      \pdf_object_unnamed_write:nx {dict}{/Kids~[\l__tag_tmpb_tl]}
84      \__tag_prop_gput:cnx
85          { g__tag_struct_0_prop }
86          { IDTree }
87          { \pdf_object_ref_last: }
88    }
```

1.4 Writing structure elements

The following commands are needed to write out the structure.

__tag_tree_write_structtreeroot: This writes out the root object.

```
89  \pdf_version_compare:NnTF < {2.0}
90    {
91    \cs_new_protected:Npn \__tag_tree_write_structtreeroot:
92      {
93          \__tag_prop_gput:cnx
94            { g__tag_struct_0_prop }
95            { ParentTree }
96            { \pdf_object_ref:n { __tag/tree/parenttree } }
97          \__tag_prop_gput:cnx
98            { g__tag_struct_0_prop }
99            { RoleMap }
100           { \pdf_object_ref:n { __tag/tree/rolemap } }
101         \__tag_struct_fill_kid_key:n { 0 }
102         \__tag_struct_get_dict_content:nN { 0 } \l__tag_tmpa_tl
103         \pdf_object_write:nnx
104             { __tag/struct/0 }
105             {dict}
106             {
107               \l__tag_tmpa_tl
```

50

```
108              }
109          }
110      }
```
no RoleMap in pdf 2.0
```
111  {
112      \cs_new_protected:Npn \__tag_tree_write_structtreeroot:
113          {
114              \__tag_prop_gput:cnx
115                  { g__tag_struct_0_prop }
116                  { ParentTree }
117                  { \pdf_object_ref:n { __tag/tree/parenttree } }
118              \__tag_struct_fill_kid_key:n { 0 }
119              \__tag_struct_get_dict_content:nN { 0 } \l__tag_tmpa_tl
120              \pdf_object_write:nnx
121                  { __tag/struct/0 }
122                  {dict}
123                  {
124                      \l__tag_tmpa_tl
125                  }
126          }
127  }
```

(*End definition for* `\__tag_tree_write_structtreeroot:`.)

`\__tag_tree_write_structelements:`   This writes out the other struct elems, the absolute number is in the counter.

```
128  \cs_new_protected:Npn \__tag_tree_write_structelements:
129  {
130      \int_step_inline:nnnn {1}{1}{\c@g__tag_struct_abs_int}
131      {
132          \__tag_struct_write_obj:n { ##1 }
133      }
134  }
```

(*End definition for* `\__tag_tree_write_structelements:`.)

1.5   ParentTree

`__tag/tree/parenttree`   The object which will hold the parenttree

```
135  \pdf_object_new:n { __tag/tree/parenttree }
```

(*End definition for* `__tag/tree/parenttree`.)

The ParentTree maps numbers to objects or (if the number represents a page) to arrays of objects. The numbers refer to two dictinct types of entries: page streams and real objects like annotations. The numbers must be distinct and ordered. So we rely on abspage for the pages and put the real objects at the end. We use a counter to have a chance to get the correct number if code is processed twice.

`\c@g__tag_parenttree_obj_int`   This is a counter for the real objects. It starts at the absolute last page value. It relies on l3ref.

```
136  \newcounter  { g__tag_parenttree_obj_int }
137  \hook_gput_code:nnn{begindocument}{tagpdf}
138  {
139      \int_gset:Nn
```

```
140          \c@g__tag_parenttree_obj_int
141          { \__tag_ref_value_lastpage:nn{abspage}{100}  }
142    }
```

(*End definition for* \c@g__tag_parenttree_obj_int.)

We store the number/object references in a tl-var. If more structure is needed one could switch to a seq.

\g__tag_parenttree_objr_tl

```
143 \tl_new:N \g__tag_parenttree_objr_tl
```

(*End definition for* \g__tag_parenttree_objr_tl.)

__tag_parenttree_add_objr:nn   This command stores a StructParent number and a objref into the tl var. This is only for objects like annotations, pages are handled elsewhere.

```
144 \cs_new_protected:Npn \__tag_parenttree_add_objr:nn #1 #2 %#1 StructParent number, #2 objref
145    {
146      \tl_gput_right:Nx \g__tag_parenttree_objr_tl
147        {
148          #1 \c_space_tl #2 ^^J
149        }
150    }
```

(*End definition for* __tag_parenttree_add_objr:nn.)

\l__tag_parenttree_content_tl   A tl-var which will get the page related parenttree content.

```
151 \tl_new:N \l__tag_parenttree_content_tl
```

(*End definition for* \l__tag_parenttree_content_tl.)

__tag_tree_fill_parenttree: This is the main command to assemble the page related entries of the parent tree. It wanders through the pages and the mcid numbers and collects all mcid of one page.

```
152
153 \cs_new_protected:Npn \__tag_tree_fill_parenttree:
154    {
155      \int_step_inline:nnnn{1}{1}{\__tag_ref_value_lastpage:nn{abspage}{-1}} %not quite clear i
156        { %page ##1
157          \prop_clear:N \l__tag_tmpa_prop
158          \int_step_inline:nnnn{1}{1}{\__tag_ref_value_lastpage:nn{tagmcabs}{-1}}
159            {
160              %mcid####1
161              \int_compare:nT
162                {\__tag_ref_value:enn{mcid-####1}{tagabspage}{-1}=##1} %mcid is on current page
163                {% yes
164                  \prop_put:Nxx
165                    \l__tag_tmpa_prop
166                    {\__tag_ref_value:enn{mcid-####1}{tagmcid}{-1}}
167                    {\prop_item:Nn \g__tag_mc_parenttree_prop {####1}}
168                }
169            }
170          \tl_put_right:Nx\l__tag_parenttree_content_tl
171            {
172              \int_eval:n {##1-1}\c_space_tl
173              [\c_space_tl %]
174            }
```

52

```
175        \int_step_inline:nnnn
176          {0}
177          {1}
178          { \prop_count:N \l__tag_tmpa_prop -1 }
179          {
180            \prop_get:NnNTF \l__tag_tmpa_prop {####1} \l__tag_tmpa_tl
181              {% page#1:mcid##1:\l__tag_tmpa_tl :content
182                \tl_put_right:Nx \l__tag_parenttree_content_tl
183                  {
184                    \pdf_object_if_exist:eT { __tag/struct/\l__tag_tmpa_tl }
185                     {
186                       \pdf_object_ref:e { __tag/struct/\l__tag_tmpa_tl }
187                     }
188                    \c_space_tl
189                  }
190              }
191              {
192                \msg_warning:nn { tag } {tree-mcid-index-wrong}
193              }
194          }
195        \tl_put_right:Nn
196          \l__tag_parenttree_content_tl
197          {%[
198          ]^^J
199          }
200      }
201    }
```

(*End definition for* `\__tag_tree_fill_parenttree:`.)

`\__tag_tree_lua_fill_parenttree:` This is a special variant for luatex. lua mode must/can do it differently.

```
202 \cs_new_protected:Npn \__tag_tree_lua_fill_parenttree:
203   {
204     \tl_set:Nn \l__tag_parenttree_content_tl
205       {
206         \lua_now:e
207           {
208             ltx.__tag.func.output_parenttree
209               (
210                 \int_use:N\g_shipout_readonly_int
211               )
212           }
213       }
214   }
```

(*End definition for* `\__tag_tree_lua_fill_parenttree:`.)

`\__tag_tree_write_parenttree:` This combines the two parts and writes out the object. TODO should the check for lua be moved into the backend code?

```
215 \cs_new_protected:Npn \__tag_tree_write_parenttree:
216   {
217     \bool_if:NTF \g__tag_mode_lua_bool
218       {
219         \__tag_tree_lua_fill_parenttree:
```

```
220        }
221      {
222        \__tag_tree_fill_parenttree:
223      }
224    \tl_put_right:NV \l__tag_parenttree_content_tl\g__tag_parenttree_objr_tl
225    \pdf_object_write:nnx  { __tag/tree/parenttree }{dict}
226      {
227        /Nums\c_space_tl [\l__tag_parenttree_content_tl]
228      }
229  }
```

(*End definition for* `\__tag_tree_write_parenttree:`.)

1.6   Rolemap dictionary

The Rolemap dictionary describes relations between new tags and standard types. The main part here is handled in the role module, here we only define the command which writes it to the PDF.

`__tag/tree/rolemap`   At first we reserve again an object.

```
230  \pdf_version_compare:NnT < {2.0}
231    {
232      \pdf_object_new:n { __tag/tree/rolemap }
233    }
```

(*End definition for* `__tag/tree/rolemap`.)

`\__tag_tree_write_rolemap:`   This writes out the rolemap, basically it simply pushes out the dictionary which has been filled in the role module.

```
234  \pdf_version_compare:NnTF < {2.0}
235    {
236     \cs_new_protected:Npn \__tag_tree_write_rolemap:
237       {
238         \prop_map_inline:Nn\g__tag_role_rolemap_prop
239           {
240             \tl_if_eq:nnF {##1}{##2}
241               {
242                 \pdfdict_gput:nnx {g__tag_role/RoleMap_dict}
243                 {##1}
244                 {\pdf_name_from_unicode_e:n{##2}}
245               }
246           }
247         \pdf_object_write:nnx  { __tag/tree/rolemap }{dict}
248           {
249             \pdfdict_use:n{g__tag_role/RoleMap_dict}
250           }
251       }
252    }
253    {
254     \cs_new_protected:Npn \__tag_tree_write_rolemap:{}
255    }
256
```

(*End definition for* `\__tag_tree_write_rolemap:`.)

1.7 Classmap dictionary

Classmap and attributes are setup in the struct module, here is only the code to write it out. It should only done if values have been used.

```
257 \cs_new_protected:Npn \__tag_tree_write_classmap:
258   {
259     \tl_clear:N \l__tag_tmpa_tl
260     \seq_gremove_duplicates:N \g__tag_attr_class_used_seq
261     \seq_set_map:NNn \l__tag_tmpa_seq \g__tag_attr_class_used_seq
262       {
263         ##1\c_space_tl
264         <<
265           \prop_item:Nn
266             \g__tag_attr_entries_prop
267             {##1}
268         >>
269       }
270     \tl_set:Nx \l__tag_tmpa_tl
271       {
272         \seq_use:Nn
273           \l__tag_tmpa_seq
274           { \iow_newline: }
275       }
276     \tl_if_empty:NF
277       \l__tag_tmpa_tl
278       {
279         \pdf_object_new:n { __tag/tree/classmap }
280         \pdf_object_write:nnx
281           { __tag/tree/classmap }
282           {dict}
283           { \l__tag_tmpa_tl }
284         \__tag_prop_gput:cnx
285           { g__tag_struct_0_prop }
286           { ClassMap }
287           { \pdf_object_ref:n { __tag/tree/classmap }  }
288       }
289   }
```

(*End definition for* __tag_tree_write_classmap:.)

1.8 Namespaces

Namespaces are handle in the role module, here is the code to write them out. Namespaces are only relevant for pdf2.0.

```
290 \pdf_object_new:n { __tag/tree/namespaces }
```

(*End definition for* __tag/tree/namespaces.)

```
291 \cs_new_protected:Npn \__tag_tree_write_namespaces:
292   {
```

```
293    \pdf_version_compare:NnF < {2.0}
294      {
295        \prop_map_inline:Nn \g__tag_role_NS_prop
296          {
297            \pdfdict_if_empty:nF {g__tag_role/RoleMapNS_##1_dict}
298              {
299                \pdf_object_write:nnx {__tag/RoleMapNS/##1}{dict}
300                  {
301                    \pdfdict_use:n {g__tag_role/RoleMapNS_##1_dict}
302                  }
303                \pdfdict_gput:nnx{g__tag_role/Namespace_##1_dict}
304                  {RoleMapNS}{\pdf_object_ref:n {__tag/RoleMapNS/##1}}
305              }
306            \pdf_object_write:nnx{tag/NS/##1}{dict}
307              {
308                \pdfdict_use:n {g__tag_role/Namespace_##1_dict}
309              }
310          }
311        \pdf_object_write:nnx {__tag/tree/namespaces}{array}
312          {
313            \prop_map_tokens:Nn \g__tag_role_NS_prop{\use_ii:nn}
314          }
315      }
316  }
```

(End definition for __tag_tree_write_namespaces:*.)*

1.9   Finishing the structure

This assembles the various parts. TODO (when tabular are done or if someone requests it): IDTree

```
317  \hook_new:n {tagpdf/finish/before}
318  \cs_new_protected:Npn \__tag_finish_structure:
319    {
320      \bool_if:NT\g__tag_active_tree_bool
321        {
322          \hook_use:n {tagpdf/finish/before}
323          \__tag_tree_final_checks:
324          \__tag_tree_write_parenttree:
325          \__tag_tree_write_idtree:
326          \__tag_tree_write_rolemap:
327          \__tag_tree_write_classmap:
328          \__tag_tree_write_namespaces:
329          \__tag_tree_write_structelements: %this is rather slow!!
330          \__tag_tree_write_structtreeroot:
331        }
332    }
```

(End definition for __tag_finish_structure:*.)*

1.10 StructParents entry for Page

We need to add to the Page resources the `StructParents` entry, this is simply the absolute page number.

```
333 \hook_gput_code:nnn{begindocument}{tagpdf}
334   {
335     \bool_if:NT\g__tag_active_tree_bool
336       {
337         \hook_gput_code:nnn{shipout/before} { tagpdf/structparents }
338           {
339             \pdfmanagement_add:nnx
340               { Page }
341               { StructParents }
342               { \int_eval:n { \g_shipout_readonly_int} }
343           }
344       }
345   }
346 ⟨/package⟩
```

Part IV
The **tagpdf-mc-shared** module
Code related to Marked Content
(mc-chunks), code shared by all
modes
Part of the tagpdf package

1 Public Commands

`\tag_mc_begin:n`
`\tag_mc_end:`

`\tag_mc_begin:n{⟨key-values⟩}`
`\tag_mc_end:`

These commands insert the end code of the marked content. They don't end a group and in generic mode it doesn't matter if they are in another group as the starting commands. In generic mode both commands check if they are correctly nested and issue a warning if not.

`\tag_mc_use:n`

`\tag_mc_use:n{⟨label⟩}`

These command allow to record a marked content that was stashed away before into the current structure. A marked content can be used only once – the command will issue a warning if an mc is use a second time.

`\tag_mc_artifact_group_begin:n`
`\tag_mc_artifact_group_end:`

`\tag_mc_artifact_group_begin:n {⟨name⟩}`
`   \tag_mc_artifact_group_end:`

New: 2019-11-20

This command pair creates a group with an artifact marker at the begin and the end. Inside the group the tagging commands are disabled. It allows to mark a complete region as artifact without having to worry about user commands with tagging commands. ⟨name⟩ should be a value allowed also for the `artifact` key. It pushes and pops mc-chunks at the begin and end. TODO: document is in tagpdf.tex

`\tag_mc_end_push:`
`\tag_mc_begin_pop:n`

`\tag_mc_end_push:`
`\tag_mc_begin_pop:n{⟨key-values⟩}`

New: 2021-04-22

If there is an open mc chunk, `\tag_mc_end_push:` ends it and pushes its tag of the (global) stack. If there is no open chunk, it puts −1 on the stack (for debugging) `\tag_-mc_begin_pop:n` removes a value from the stack. If it is different from −1 it opens a tag with it. The reopened mc chunk looses info like the alt text for now.

`\tag_mc_if_in_p:` ⋆
`\tag_mc_if_in:TF` ⋆

`\tag_mc_if_in:TF {⟨true code⟩} {⟨false code⟩}`

Determines if a mc-chunk is open.

2 Public keys

The following keys can be used with \tag_mc_begin:n, \tagmcbegin, \tag_mc_begin_pop:n,

tag␣(mc-key)   This key is required, unless artifact is used. The value is a tag like P or H1 without a slash at the begin, this is added by the code. It is possible to setup new tags. The value of the key is expanded, so it can be a command. The expansion is passed unchanged to the PDF, so it should with a starting slash give a valid PDF name (some ascii with numbers like H4 is fine).

artifact␣(mc-key)   This will setup the marked content as an artifact. The key should be used for content that should be ignored. The key can take one of the values pagination, layout, page, background and notype (this is the default).

raw␣(mc-key)   This key allows to add more entries to the properties dictionary. The value must be correct, low-level PDF. E.g. raw=/Alt (Hello) will insert an alternative Text.

alt␣(mc-key)   This key inserts an /Alt value in the property dictionary of the BDC operator. The value is handled as verbatim string, commands are not expanded. The value will be expanded first once.

actualtext␣(mc-key)   This key inserts an /ActualText value in the property dictionary of the BDC operator. The value is handled as verbatim string, commands are not expanded. The value will be expanded first once.

label␣(mc-key)   This key sets a label by which one can call the marked content later in another structure (if it has been stashed with the stash key). Internally the label name will start with tagpdf-.

stash␣(mc-key)   This "stashes" an mc-chunk: it is not inserted into the current structure. It should be normally be used along with a label to be able to use the mc-chunk in another place.
    The code is splitted into three parts: code shared by all engines, code specific to luamode and code not used by luamode.

3 Marked content code − shared

```
1 ⟨@@=tag⟩
2 ⟨∗header⟩
3 \ProvidesExplPackage {tagpdf-mc-code-shared} {2023-03-07} {0.98e}
4   {part of tagpdf - code related to marking chunks -
5    code shared by generic and luamode }
6 ⟨/header⟩
```

3.1 Variables and counters

MC chunks must be counted. I use a latex counter for the absolute count, so that it is added to \cl@@ckpt and restored e.g. in tabulars and align. \int_new:N \c@g_@@_MCID_int and \tl_put_right:Nn\cl@@ckpt{\@elt{g_uf_test_int}} would work too, but as the name is not expl3 then too, why bother? The absolute counter can be used to label and to check if the page counter needs a reset.

g__tag_MCID_abs_int

```
7  ⟨*shared⟩
8  \newcounter { g__tag_MCID_abs_int }
```

(*End definition for* g__tag_MCID_abs_int.)

__tag_get_mc_abs_cnt: A (expandable) function to get the current value of the cnt.

```
9  \cs_new:Npn \__tag_get_mc_abs_cnt: { \int_use:N \c@g__tag_MCID_abs_int }
```

(*End definition for* __tag_get_mc_abs_cnt:.)

\g__tag_MCID_tmp_bypage_int   The following hold the temporary by page number assigned to a mc. It must be defined in the shared code to avoid problems with labels.

```
10  \int_new:N \g__tag_MCID_tmp_bypage_int
```

(*End definition for* \g__tag_MCID_tmp_bypage_int.)

\g__tag_in_mc_bool   This booleans record if a mc is open, to test nesting.

```
11  \bool_new:N \g__tag_in_mc_bool
```

(*End definition for* \g__tag_in_mc_bool.)

\g__tag_mc_parenttree_prop   For every chunk we need to know the structure it is in, to record this in the parent tree. We store this in a property.
key: absolute number of the mc (tagmcabs)
value: the structure number the mc is in

```
12  \__tag_prop_new:N \g__tag_mc_parenttree_prop
```

(*End definition for* \g__tag_mc_parenttree_prop.)

\g__tag_mc_parenttree_prop   Some commands (e.g. links) want to close a previous mc and reopen it after they did their work. For this we create a stack:

```
13  \seq_new:N \g__tag_mc_stack_seq
```

(*End definition for* \g__tag_mc_parenttree_prop.)

\l__tag_mc_artifact_type_tl   Artifacts can have various types like Pagination or Layout. This stored in this variable.

```
14  \tl_new:N \l__tag_mc_artifact_type_tl
```

(*End definition for* \l__tag_mc_artifact_type_tl.)

\l__tag_mc_key_stash_bool
\l__tag_mc_artifact_bool   This booleans store the stash and artifact status of the mc-chunk.

```
15  \bool_new:N \l__tag_mc_key_stash_bool
16  \bool_new:N \l__tag_mc_artifact_bool
```

(*End definition for* \l__tag_mc_key_stash_bool *and* \l__tag_mc_artifact_bool.)

\l__tag_mc_key_tag_tl	Variables used by the keys. `\l_@@_mc_key_properties_tl` will collect a number of
\g__tag_mc_key_tag_tl	values. TODO: should this be a pdfdict now?
\l__tag_mc_key_label_tl	
\l__tag_mc_key_properties_tl	

```
17 \tl_new:N \l__tag_mc_key_tag_tl
18 \tl_new:N \g__tag_mc_key_tag_tl
19 \tl_new:N \l__tag_mc_key_label_tl
20 \tl_new:N \l__tag_mc_key_properties_tl
```

(*End definition for* \l__tag_mc_key_tag_tl *and others.*)

3.2 Functions

__tag_mc_handle_mc_label:n The commands labels a mc-chunk. It is used if the user explicitly labels the mc-chunk with the `label` key. The argument is the value provided by the user. It stores the attributes

`tagabspage`: the absolute page, `\g_shipout_readonly_int`,

`tagmcabs`: the absolute mc-counter `\c@g_@@_MCID_abs_int`,

`tagmcid`: the ID of the chunk on the page `\g_@@_MCID_tmp_bypage_int`, this typically settles down after a second compilation. The reference command is defined in tagpdf.dtx and is based on l3ref.

```
21 \cs_new:Nn \__tag_mc_handle_mc_label:n
22   {
23     \__tag_ref_label:en{tagpdf-#1}{mc}
24   }
```

(*End definition for* __tag_mc_handle_mc_label:n.)

__tag_mc_set_label_used:n Unlike with structures we can't check if a labeled mc has been used by looking at the P key, so we use a dedicated csname for the test

```
25 \cs_new_protected:Npn \__tag_mc_set_label_used:n #1 %#1 labelname
26   {
27     \tl_new:c { g__tag_mc_label_\tl_to_str:n{#1}_used_tl }
28   }
29 ⟨/shared⟩
```

(*End definition for* __tag_mc_set_label_used:n.)

\tag_mc_use:n These command allow to record a marked content that was stashed away before into the current structure. A marked content can be used only once – the command will issue a warning if an mc is use a second time. The argument is a label name set with the `label` key.

TODO: is testing for struct the right test?

```
30 ⟨base⟩\cs_new_protected:Npn \tag_mc_use:n #1 { \__tag_whatsits: }
31 ⟨*shared⟩
32 \cs_set_protected:Npn \tag_mc_use:n #1 %#1: label name
33   {
34     \__tag_check_if_active_struct:T
35       {
36         \tl_set:Nx  \l__tag_tmpa_tl { \__tag_ref_value:nnn{tagpdf-#1}{tagmcabs}{} }
37         \tl_if_empty:NTF\l__tag_tmpa_tl
38           {
39             \msg_warning:nnn {tag} {mc-label-unknown} {#1}
40           }
41           {
```

```
42                    \cs_if_free:cTF { g__tag_mc_label_\tl_to_str:n{#1}_used_tl }
43                      {
44                        \__tag_mc_handle_stash:x { \l__tag_tmpa_tl }
45                        \__tag_mc_set_label_used:n {#1}
46                      }
47                      {
48                        \msg_warning:nnn {tag}{mc-used-twice}{#1}
49                      }
50                }
51            }
52        }
53 ⟨/shared⟩
```

(*End definition for* `\tag_mc_use:n`. *This function is documented on page 58.*)

`\tag_mc_artifact_group_begin:n`
`\tag_mc_artifact_group_end:` This opens an artifact of the type given in the argument, and then stops all tagging. It creates a group. It pushes and pops mc-chunks at the begin and end.

```
54 ⟨base⟩\cs_new_protected:Npn \tag_mc_artifact_group_begin:n #1 {}
55 ⟨base⟩\cs_new_protected:Npn \tag_mc_artifact_group_end:{}
56 ⟨*shared⟩
57 \cs_set_protected:Npn \tag_mc_artifact_group_begin:n #1
58  {
59    \tag_mc_end_push:
60    \tag_mc_begin:n {artifact=#1}
61    \tag_stop_group_begin:
62  }
63
64 \cs_set_protected:Npn \tag_mc_artifact_group_end:
65  {
66    \tag_stop_group_end:
67    \tag_mc_end:
68    \tag_mc_begin_pop:n{}
69  }
70 ⟨/shared⟩
```

(*End definition for* `\tag_mc_artifact_group_begin:n` *and* `\tag_mc_artifact_group_end:`. *These functions are documented on page 58.*)

`\tag_mc_end_push:`
`\tag_mc_begin_pop:n`
```
71 ⟨base⟩\cs_new_protected:Npn \tag_mc_end_push: {}
72 ⟨base⟩\cs_new_protected:Npn \tag_mc_begin_pop:n #1 {}
73 ⟨*shared⟩
74 \cs_set_protected:Npn \tag_mc_end_push:
75   {
76     \__tag_check_if_active_mc:T
77       {
78         \__tag_mc_if_in:TF
79           {
80             \seq_gpush:Nx \g__tag_mc_stack_seq { \tag_get:n {mc_tag} }
81             \__tag_check_mc_pushed_popped:nn
82               { pushed }
83               { \tag_get:n {mc_tag} }
84             \tag_mc_end:
85           }
86           {
```

```
87        \seq_gpush:Nn \g__tag_mc_stack_seq {-1}
88        \__tag_check_mc_pushed_popped:nn { pushed }{-1}
89      }
90    }
91  }
92
93 \cs_set_protected:Npn \tag_mc_begin_pop:n #1
94   {
95     \__tag_check_if_active_mc:T
96       {
97         \seq_gpop:NNTF \g__tag_mc_stack_seq \l__tag_tmpa_tl
98           {
99             \tl_if_eq:NnTF \l__tag_tmpa_tl {-1}
100              {
101                \__tag_check_mc_pushed_popped:nn {popped}{-1}
102              }
103              {
104                \__tag_check_mc_pushed_popped:nn {popped}{\l__tag_tmpa_tl}
105                \tag_mc_begin:n {tag=\l__tag_tmpa_tl,#1}
106              }
107          }
108          {
109            \__tag_check_mc_pushed_popped:nn {popped}{empty~stack,~nothing}
110          }
111      }
112  }
```

(*End definition for* `\tag_mc_end_push:` *and* `\tag_mc_begin_pop:n`*. These functions are documented on page* *.*)

3.3 Keys

This are the keys where the code can be shared between the modes.

stash␣(mc-key) the two internal artifact keys are use to define the public `artifact`. For now we add
__artifact-bool support for the subtypes Header and Footer. Watermark,PageNum, LineNum,Redac-
__artifact-type tion,Bates will be added if some use case emerges. If some use case for /BBox and
/Attached emerges, it will be perhaps necessary to adapt the code.

```
113 \keys_define:nn { __tag / mc }
114   {
115     stash                      .bool_set:N    = \l__tag_mc_key_stash_bool,
116     __artifact-bool            .bool_set:N    = \l__tag_mc_artifact_bool,
117     __artifact-type            .choice:,
118     __artifact-type / pagination .code:n    =
119       {
120         \tl_set:Nn \l__tag_mc_artifact_type_tl { Pagination }
121       },
122     __artifact-type / pagination/header .code:n    =
123       {
124         \tl_set:Nn \l__tag_mc_artifact_type_tl { Pagination/Subtype/Header }
125       },
126     __artifact-type / pagination/footer .code:n    =
127       {
128         \tl_set:Nn \l__tag_mc_artifact_type_tl { Pagination/Subtype/Footer }
```

```
129         },
130       __artifact-type / layout      .code:n    =
131         {
132           \tl_set:Nn \l__tag_mc_artifact_type_tl { Layout }
133         },
134       __artifact-type / page        .code:n    =
135         {
136           \tl_set:Nn \l__tag_mc_artifact_type_tl { Page }
137         },
138       __artifact-type / background .code:n    =
139         {
140           \tl_set:Nn \l__tag_mc_artifact_type_tl { Background }
141         },
142       __artifact-type / notype      .code:n    =
143         {
144           \tl_set:Nn \l__tag_mc_artifact_type_tl {}
145         },
146       __artifact-type /            .code:n    =
147         {
148           \tl_set:Nn \l__tag_mc_artifact_type_tl {}
149         },
150   }
```

(*End definition for* stash (mc-key)*,* __artifact-bool*, and* __artifact-type*. This function is documented on page 59.*)

```
151 ⟨/shared⟩
```

Part V

The **tagpdf-mc-generic** module
Code related to Marked Content
(mc-chunks), generic mode
Part of the tagpdf package

1 Marked content code – generic mode

```
1 ⟨@@=tag⟩
2 ⟨*generic⟩
3 \ProvidesExplPackage {tagpdf-mc-code-generic} {2023-03-07} {0.98e}
4  {part of tagpdf - code related to marking chunks - generic mode}
5 ⟨/generic⟩
6 ⟨*debug⟩
7 \ProvidesExplPackage {tagpdf-debug-generic} {2023-03-07} {0.98e}
8  {part of tagpdf - debugging code related to marking chunks - generic mode}
9 ⟨/debug⟩
```

1.1 Variables

\g__tag_MCID_byabspage_prop

This property will hold the current maximum on a page it will contain key-value of type $\langle abspagenum \rangle = \langle max\ mcid \rangle$

```
10 ⟨*generic⟩
11 \__tag_prop_new:N \g__tag_MCID_byabspage_prop
```

(*End definition for* \g__tag_MCID_byabspage_prop.)

\l__tag_mc_ref_abspage_tl

We need a ref-label system to ensure that the MCID cnt restarts at 0 on a new page This will be used to store the tagabspage attribute retrieved from a label.

```
12 \tl_new:N \l__tag_mc_ref_abspage_tl
```

(*End definition for* \l__tag_mc_ref_abspage_tl.)

\l__tag_mc_tmpa_tl

temporary variable

```
13 \tl_new:N \l__tag_mc_tmpa_tl
```

(*End definition for* \l__tag_mc_tmpa_tl.)

\g__tag_mc_marks

a marks register to keep track of the mc's at page breaks and a sequence to keep track of the data for the continuation extra-tmb. We probably will need to track mc-marks in more than one stream, so the seq contains the name of the stream.

```
14 \newmarks  \g__tag_mc_marks
```

(*End definition for* \g__tag_mc_marks.)

`\g__tag_mc_main_marks_seq`
`\g_tag_mc_footnote_marks_seq`
`\g_tag_mc_multicol_marks_seq`

Each stream has an associated global seq variable holding the bottom marks from the/a previous chunk in the stream. We provide three by default: main, footnote and multicol. TODO: perhaps an interface for more streams will be needed.

```
15 \seq_new:N \g__tag_mc_main_marks_seq
16 \seq_new:N \g__tag_mc_footnote_marks_seq
17 \seq_new:N \g__tag_mc_multicol_marks_seq
```

(*End definition for* `\g__tag_mc_main_marks_seq`, `\g__tag_mc_footnote_marks_seq`, *and* `\g__tag_mc_-multicol_marks_seq`.)

`\l__tag_mc_firstmarks_seq`
`\l__tag_mc_botmarks_seq`

The marks content contains a number of data which we will have to access and compare, so we will store it locally in two sequences. topmarks is unusable in LaTeX so we ignore it.

```
18 \seq_new:N  \l__tag_mc_firstmarks_seq
19 \seq_new:N  \l__tag_mc_botmarks_seq
```

(*End definition for* `\l__tag_mc_firstmarks_seq` *and* `\l__tag_mc_botmarks_seq`.)

1.2 Functions

`\__tag_mc_begin_marks:nn`
`\__tag_mc_artifact_begin_marks:n`
`\__tag_mc_end_marks:`

Generic mode need to set marks for the page break and split stream handling. We always set two marks to be able to detect the case when no mark is on a page/galley. MC-begin commands will set (b,-,data) and (b,+,data), MC-end commands will set (e,-,data) and (e,+,data).

```
20 \cs_new_protected:Npn \__tag_mc_begin_marks:nn #1 #2 %#1 tag, #2 label
21   {
22     \tex_marks:D \g__tag_mc_marks
23       {
24         b-, %first of begin pair
25         \int_use:N\c@g__tag_MCID_abs_int, %mc-num
26         \g__tag_struct_stack_current_tl,  %structure num
27         #1, %tag
28         \bool_if:NT \l__tag_mc_key_stash_bool{stash}, % stash info
29         #2, %label
30       }
31     \tex_marks:D \g__tag_mc_marks
32       {
33         b+, % second of begin pair
34         \int_use:N\c@g__tag_MCID_abs_int, %mc-num
35         \g__tag_struct_stack_current_tl,  %structure num
36         #1, %tag
37         \bool_if:NT \l__tag_mc_key_stash_bool{stash}, % stash info
38         #2, %label
39       }
40   }
41 \cs_generate_variant:Nn \__tag_mc_begin_marks:nn {oo}
42 \cs_new_protected:Npn \__tag_mc_artifact_begin_marks:n #1 %#1 type
43   {
44     \tex_marks:D \g__tag_mc_marks
45       {
46         b-, %first of begin pair
47         \int_use:N\c@g__tag_MCID_abs_int, %mc-num
48         -1, %structure num
```

```
49        #1 %type
50      }
51    \tex_marks:D \g__tag_mc_marks
52      {
53        b+, %first of begin pair
54        \int_use:N\c@g__tag_MCID_abs_int, %mc-num
55        -1,  %structure num
56        #1  %Type
57      }
58  }
59
60 \cs_new_protected:Npn \__tag_mc_end_marks:
61  {
62    \tex_marks:D \g__tag_mc_marks
63      {
64        e-, %first of end pair
65        \int_use:N\c@g__tag_MCID_abs_int, %mc-num
66        \g__tag_struct_stack_current_tl,  %structure num
67      }
68    \tex_marks:D \g__tag_mc_marks
69      {
70        e+, %second of end pair
71        \int_use:N\c@g__tag_MCID_abs_int, %mc-num
72        \g__tag_struct_stack_current_tl,  %structure num
73      }
74  }
```

(*End definition for* `\__tag_mc_begin_marks:nn`, `\__tag_mc_artifact_begin_marks:n`, *and* `\__tag_mc_-`
`end_marks:`.)

`\__tag_mc_disable_marks:` This disables the marks. They can't be reenabled, so it should only be used in groups.

```
75 \cs_new_protected:Npn \__tag_mc_disable_marks:
76  {
77    \cs_set_eq:NN \__tag_mc_begin_marks:nn \use_none:nn
78    \cs_set_eq:NN \__tag_mc_artifact_begin_marks:n \use_none:n
79    \cs_set_eq:NN \__tag_mc_end_marks: \prg_do_nothing:
80  }
```

(*End definition for* `\__tag_mc_disable_marks:`.)

`\__tag_mc_get_marks:` This stores the current content of the marks in the sequences. It naturally should only be used in places where it makes sense.

```
81 \cs_new_protected:Npn \__tag_mc_get_marks:
82  {
83    \exp_args:NNx
84    \seq_set_from_clist:Nn \l__tag_mc_firstmarks_seq
85      { \tex_firstmarks:D \g__tag_mc_marks }
86    \exp_args:NNx
87    \seq_set_from_clist:Nn \l__tag_mc_botmarks_seq
88      { \tex_botmarks:D    \g__tag_mc_marks }
89  }
```

(*End definition for* `\__tag_mc_get_marks:`.)

`\__tag_mc_store:nnn` This inserts the mc-chunk ⟨*mc-num*⟩ into the structure struct-num after the ⟨*mc-prev*⟩. The structure must already exist. The additional mcid dictionary is stored in a property. The item is retrieved when the kid entry is built. We test if there is already an addition and append if needed.

```
90 \cs_new_protected:Npn \__tag_mc_store:nnn #1 #2 #3 %#1 mc-prev, #2 mc-num #3 structure-
   num
91   {
92     %\prop_show:N \g__tag_struct_cont_mc_prop
93     \prop_get:NnNTF \g__tag_struct_cont_mc_prop {#1} \l__tag_tmpa_tl
94       {
95         \prop_gput:Nnx \g__tag_struct_cont_mc_prop {#1}{ \l__tag_tmpa_tl \__tag_struct_mcid_di
96       }
97       {
98         \prop_gput:Nnx \g__tag_struct_cont_mc_prop {#1}{ \__tag_struct_mcid_dict:n {#2}}
99       }
100     \prop_gput:Nxx \g__tag_mc_parenttree_prop
101       {#2}
102       {#3}
103   }
104 \cs_generate_variant:Nn \__tag_mc_store:nnn {xxx}
```

(*End definition for* `\__tag_mc_store:nnn`.)

`\__tag_mc_insert_extra_tmb:n`
`\__tag_mc_insert_extra_tme:n` These two functions should be used in the output routine at the place where a mc-literal could be missing due to a page break or some other split. They check (with the help of the marks) if a extra-tmb or extra-tme is needed. The tmb command stores also the mc into the structure, the tme has to store the data for a following extra-tmb. The argument takes a stream name like main or footnote to allow different handling there. The content of the marks must be stored before (with `\@@_mc_get_marks:` or manually) into `\l_@@_mc_firstmarks_seq` and `\l_@@_mc_botmarks_seq` so that the tests can use them.

```
105 \cs_new_protected:Npn \__tag_mc_insert_extra_tmb:n #1 % #1 stream: e.g. main or footnote
106   {
107     \__tag_check_typeout_v:n {=>~ first~ \seq_use:Nn \l__tag_mc_firstmarks_seq {,~}}
108     \__tag_check_typeout_v:n {=>~ bot~ \seq_use:Nn \l__tag_mc_botmarks_seq {,~}}
109     \__tag_check_if_mc_tmb_missing:TF
110       {
111         \__tag_check_typeout_v:n {=>~ TMB~ ~ missing~ ---~ inserted}
112         %test if artifact
113         \int_compare:nNnTF { \seq_item:cn { g__tag_mc_#1_marks_seq } {3} } = {-
   1}
114           {
115             \tl_set:Nx \l__tag_tmpa_tl { \seq_item:cn { g__tag_mc_#1_marks_seq } {4} }
116             \__tag_mc_handle_artifact:N \l__tag_tmpa_tl
117           }
118           {
119             \exp_args:Nx
120             \__tag_mc_bdc_mcid:n
121               {
122                 \seq_item:cn { g__tag_mc_#1_marks_seq } {4}
123               }
124             \str_if_eq:eeTF
125               {
```

```
126                    \seq_item:cn { g__tag_mc_#1_marks_seq } {5}
127                 }
128                 {}
129                 {
130                   %store
131                   \__tag_mc_store:xxx
132                     {
133                       \seq_item:cn { g__tag_mc_#1_marks_seq } {2}
134                     }
135                     { \int_eval:n{\c@g__tag_MCID_abs_int} }
136                     {
137                       \seq_item:cn { g__tag_mc_#1_marks_seq } {3}
138                     }
139                 }
140                 {
141                   %stashed -> warning!!
142                 }
143            }
144        }
145        {
146          \__tag_check_typeout_v:n {=>~ TMB~ not~ missing}
147        }
148    }
149
150 \cs_new_protected:Npn \__tag_mc_insert_extra_tme:n #1 % #1 stream, eg. main or footnote
151 {
152    \__tag_check_if_mc_tme_missing:TF
153      {
154        \__tag_check_typeout_v:n {=>~ TME~ ~ missing~ ---~ inserted}
155        \__tag_mc_emc:
156        \seq_gset_eq:cN
157          {  g__tag_mc_#1_marks_seq }
158          \l__tag_mc_botmarks_seq
159      }
160      {
161        \__tag_check_typeout_v:n {=>~ TME~ not~ missing}
162      }
163 }
```

(*End definition for* __tag_mc_insert_extra_tmb:n *and* __tag_mc_insert_extra_tme:n.)

1.3 Looking at MC marks in boxes

__tag_add_missing_mcs:Nn Assumptions:

- test for tagging active outside;

- mark retrieval also outside.

This takes a box register as its first argument (or the register number in a count register, as used by multicol). It adds an extra tmb at the top of the box if necessary and similarly an extra tme at the end. This is done by adding hboxes in a way that the positioning and the baseline of the given box is not altered. The result is written back to the box.

The second argument is the stream this box belongs to und is currently either `main` for the main galley, `footnote` for footnote note text, or `multicol` for boxes produced for columns in that environment. Other streams may follow over time.

```
164 \cs_new_protected:Npn\__tag_add_missing_mcs:Nn #1 #2 {
165   \vbadness \@M
166   \vfuzz    \c_max_dim
167   \vbox_set_to_ht:Nnn #1 { \box_ht:N #1 } {
168     \hbox_set:Nn \l__tag_tmpa_box { \__tag_mc_insert_extra_tmb:n {#2} }
169     \hbox_set:Nn \l__tag_tmpb_box { \__tag_mc_insert_extra_tme:n {#2} }
170     \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
171         {
172           \seq_log:c { g__tag_mc_#2_marks_seq}
173         }
```

The box placed on the top gets zero size and thus will not affect the box dimensions of the box we are modifying.

```
174     \box_set_ht:Nn \l__tag_tmpa_box \c_zero_dim
175     \box_set_dp:Nn \l__tag_tmpa_box \c_zero_dim
```

The box added at the bottom will get the depth of the original box. This way we can arrange that from the outside everything looks as before.

```
176     \box_set_ht:Nn \l__tag_tmpb_box \c_zero_dim
177     \box_set_dp:Nn \l__tag_tmpb_box { \box_dp:N #1 }
```

We need to set `\boxmaxdepth` in case the original box has an unusually large depth, otherwise that depth is not preserved when we string things together.

```
178     \boxmaxdepth \@maxdepth
179     \box_use_drop:N        \l__tag_tmpa_box
180     \vbox_unpack_drop:N     #1
```

Back up by the depth of the box as we add that later again.

```
181     \tex_kern:D -\box_dp:N \l__tag_tmpb_box
```

And we don't want any glue added when we add the box.

```
182     \nointerlineskip
183     \box_use_drop:N \l__tag_tmpb_box
184   }
185 }
```

(*End definition for* `\__tag_add_missing_mcs:Nn`.)

`\__tag_add_missing_mcs_to_stream:Nn`   This is the main command to add mc to the stream. It is therefor guarded by the mc-boolean.

If we aren't in the main stream then processing is a bit more complicated because to get at the marks in the box we need to artifically split it and then look at the split marks.

First argument is the box to update and the second is the "stream". In lua mode the command is a no-op.

```
186 \cs_new_protected:Npn \__tag_add_missing_mcs_to_stream:Nn #1#2
187   {
188     \__tag_check_if_active_mc:T {
```

First set up a temp box for trial splitting.

```
189     \vbadness\maxdimen
190     \box_set_eq:NN \l__tag_tmpa_box #1
```

Split the box to the largest size available. This should give us all content (but to be sure that there is no issue we could test out test box is empty now (not done).

```
191        \vbox_set_split_to_ht:NNn \l__tag_tmpa_box \l__tag_tmpa_box \c_max_dim
```

As a side effect of this split we should now have the first and bottom split marks set up. We use this to set up `\l__tag_mc_firstmarks_seq`

```
192        \exp_args:NNx
193        \seq_set_from_clist:Nn \l__tag_mc_firstmarks_seq
194          { \tex_splitfirstmarks:D \g__tag_mc_marks }
```

Some debugging info:

```
195 %      \iow_term:n { First~ mark~ from~ this~ box: }
196 %      \seq_log:N \l__tag_mc_firstmarks_seq
```

If this mark was empty then clearly the bottom mark will too be empty. Thus in this case we make use of the saved bot mark from the previous chunk. Note that if this is the first chunk in the stream the global seq would contain a random value, but then we can't end in this branch because the basis assumption is that streams are properly marked up so the first chunk would always have a mark at the beginning!

```
197        \seq_if_empty:NTF \l__tag_mc_firstmarks_seq
198          {
199            \__tag_check_typeout_v:n
200              {
201                No~ marks~ so~ use~ saved~ bot~ mark:~
202                \seq_use:cn {g__tag_mc_#2_marks_seq} {,~} \iow_newline:
203              }
204            \seq_set_eq:Nc \l__tag_mc_firstmarks_seq {g__tag_mc_#2_marks_seq}
```

We also update the bot mark to the same value so that we can later apply `\__tag_add_-missing_mcs:Nn` with the data structures in place (see assumptions made there).

```
205            \seq_set_eq:NN \l__tag_mc_botmarks_seq \l__tag_mc_firstmarks_seq
206          }
```

If there was a first mark then there is also a bot mark (and it can't be the same as our marks always come in pairs). So if that branch is chosen we update `\l__tag_mc_-botmarks_seq` from the bot mark.

```
207          {
208            \__tag_check_typeout_v:n
209              {
210                Pick~ up~ new~ bot~ mark!
211              }
212            \exp_args:NNx
213            \seq_set_from_clist:Nn \l__tag_mc_botmarks_seq
214              { \tex_splitbotmarks:D   \g__tag_mc_marks }
215          }
```

Finally we call `\__tag_add_missing_mcs:Nn` to add any missing tmb/tme as needed,

```
216      \__tag_add_missing_mcs:Nn #1 {#2}
217 %%
218      \seq_gset_eq:cN  {g__tag_mc_#2_marks_seq} \l__tag_mc_botmarks_seq
219 %%
220    }
221 }
```

(*End definition for* `\__tag_add_missing_mcs_to_stream:Nn`.)

`\__tag_mc_if_in_p:`	This is a test if a mc is open or not. It depends simply on a global boolean: mc-chunks
`\__tag_mc_if_in:TF`	are added linearly so nesting should not be relevant.
`\tag_mc_if_in_p:`	
`\tag_mc_if_in:TF`	

One exception are header and footer (perhaps they are more, but for now it doesn't seem so, so there are no dedicated code to handle this situation): When they are built and added to the page we could be both inside or outside a mc-chunk. But header and footer should ignore this and not push/pop or warn about nested mc. It is therefore important there to set and reset the boolean manually. See the tagpddocu-patches.sty for an example.

```
222 \prg_new_conditional:Nnn \__tag_mc_if_in: {p,T,F,TF}
223   {
224     \bool_if:NTF \g__tag_in_mc_bool
225       { \prg_return_true:  }
226       { \prg_return_false: }
227   }
228
229 \prg_new_eq_conditional:NNn \tag_mc_if_in: \__tag_mc_if_in: {p,T,F,TF}
```

(*End definition for* `\__tag_mc_if_in:TF` *and* `\tag_mc_if_in:TF`. *This function is documented on page 58.*)

`\__tag_mc_bmc:n`	These are the low-level commands. There are now equal to the pdfmanagement com-
`\__tag_mc_emc:`	mands generic mode, but we use an indirection in case luamode need something else.
`\__tag_mc_bdc:nn`	change 04.08.2018: the commands do not check the validity of the arguments or try to
`\__tag_mc_bdc:nx`	escape them, this should be done before using them.

```
230 % #1 tag, #2 properties
231 \cs_set_eq:NN \__tag_mc_bmc:n  \pdf_bmc:n
232 \cs_set_eq:NN \__tag_mc_emc:   \pdf_emc:
233 \cs_set_eq:NN \__tag_mc_bdc:nn \pdf_bdc:nn
234 \cs_generate_variant:Nn \__tag_mc_bdc:nn {nx}
```

(*End definition for* `\__tag_mc_bmc:n`, `\__tag_mc_emc:`, *and* `\__tag_mc_bdc:nn`.)

`\__tag_mc_bdc_mcid:nn`	This create a BDC mark with an `/MCID` key. Most of the work here is to get the current
`\__tag_mc_bdc_mcid:n`	number value for the MCID: they must be numbered by page starting with 0 and then
`\__tag_mc_handle_mcid:nn`	successively. The first argument is the tag, e.g. `P` or `Span`, the second is used to pass
`\__tag_mc_handle_mcid:VV`	more properties. We also define a wrapper around the low-level command as luamode
	will need something different.

```
235 \cs_new_protected:Npn \__tag_mc_bdc_mcid:nn #1 #2
236   {
237     \int_gincr:N \c@g__tag_MCID_abs_int
238     \tl_set:Nx \l__tag_mc_ref_abspage_tl
239       {
240         \__tag_ref_value:enn %3 args
241           {
242             mcid-\int_use:N \c@g__tag_MCID_abs_int
243           }
244           { tagabspage }
245           {-1}
246       }
247     \prop_get:NoNTF
248       \g__tag_MCID_byabspage_prop
249       {
250         \l__tag_mc_ref_abspage_tl
```

```
251          }
252          \l__tag_mc_tmpa_tl
253          {
254            %key already present, use value for MCID and add 1 for the next
255            \int_gset:Nn \g__tag_MCID_tmp_bypage_int { \l__tag_mc_tmpa_tl }
256            \__tag_prop_gput:Nxx
257              \g__tag_MCID_byabspage_prop
258              { \l__tag_mc_ref_abspage_tl }
259              { \int_eval:n {\l__tag_mc_tmpa_tl +1} }
260          }
261          {
262            %key not present, set MCID to 0 and insert 1
263            \int_gzero:N \g__tag_MCID_tmp_bypage_int
264            \__tag_prop_gput:Nxx
265              \g__tag_MCID_byabspage_prop
266              { \l__tag_mc_ref_abspage_tl }
267              {1}
268          }
269        \__tag_ref_label:en
270          {
271            mcid-\int_use:N \c@g__tag_MCID_abs_int
272          }
273          { mc }
274        \__tag_mc_bdc:nx
275          {#1}
276          { /MCID~\int_eval:n { \g__tag_MCID_tmp_bypage_int }~ \exp_not:n { #2 } }
277      }
278  \cs_new_protected:Npn \__tag_mc_bdc_mcid:n #1
279    {
280      \__tag_mc_bdc_mcid:nn {#1} {}
281    }
282
283  \cs_new_protected:Npn \__tag_mc_handle_mcid:nn #1 #2 %#1 tag, #2 properties
284    {
285      \__tag_mc_bdc_mcid:nn {#1} {#2}
286    }
287
288  \cs_generate_variant:Nn \__tag_mc_handle_mcid:nn {VV}
```

(*End definition for* __tag_mc_bdc_mcid:nn, __tag_mc_bdc_mcid:n, *and* __tag_mc_handle_mcid:nn.)

__tag_mc_handle_stash:n
__tag_mc_handle_stash:x

This is the handler which puts a mc into the the current structure. The argument is the number of the mc. Beside storing the mc into the structure, it also has to record the structure for the parent tree. The name is a bit confusing, it does *not* handle mc with the stash key . . . . TODO: why does luamode use it for begin + use, but generic mode only for begin?

```
289  \cs_new_protected:Npn \__tag_mc_handle_stash:n #1 %1 mcidnum
290    {
291      \__tag_check_mc_used:n {#1}
292      \__tag_struct_kid_mc_gput_right:nn
293        { \g__tag_struct_stack_current_tl }
294        {#1}
295      \prop_gput:Nxx \g__tag_mc_parenttree_prop
296        {#1}
```

73

```
297       { \g__tag_struct_stack_current_tl }
298     }
299 \cs_generate_variant:Nn \__tag_mc_handle_stash:n { x }
```

(*End definition for* __tag_mc_handle_stash:n.)

__tag_mc_bmc_artifact: Two commands to create artifacts, one without type, and one with. We define also a
__tag_mc_bmc_artifact:n wrapper handler as luamode will need a different definition. TODO: perhaps later: more
__tag_mc_handle_artifact:N properties for artifacts

```
300 \cs_new_protected:Npn  \__tag_mc_bmc_artifact:
301   {
302     \__tag_mc_bmc:n {Artifact}
303   }
304 \cs_new_protected:Npn \__tag_mc_bmc_artifact:n #1
305   {
306     \__tag_mc_bdc:nn {Artifact}{/Type/#1}
307   }
308 \cs_new_protected:Npn \__tag_mc_handle_artifact:N #1
309     % #1 is a var containing the artifact type
310   {
311     \int_gincr:N \c@g__tag_MCID_abs_int
312     \tl_if_empty:NTF #1
313       { \__tag_mc_bmc_artifact: }
314       { \exp_args:NV\__tag_mc_bmc_artifact:n #1 }
315   }
```

(*End definition for* __tag_mc_bmc_artifact: , __tag_mc_bmc_artifact:n , *and* __tag_mc_handle_-
artifact:N.)

__tag_get_data_mc_tag: This allows to retrieve the active mc-tag. It is use by the get command.

```
316 \cs_new:Nn \__tag_get_data_mc_tag: { \g__tag_mc_key_tag_tl }
317 ⟨/generic⟩
```

(*End definition for* __tag_get_data_mc_tag:.)

\tag_mc_begin:n These are the core public commands to open and close an mc. They don't need to be
\tag_mc_end: in the same group or grouping level, but the code expect that they are issued linearly.
The tag and the state is passed to the end command through a global var and a global
boolean.

```
318 ⟨base⟩\cs_new_protected:Npn \tag_mc_begin:n #1 { \__tag_whatsits: }
319 ⟨base⟩\cs_new_protected:Nn \tag_mc_end:{ \__tag_whatsits: }
320 ⟨*generic | debug⟩
321 ⟨*generic⟩
322 \cs_set_protected:Npn \tag_mc_begin:n #1 %#1 keyval
323   {
324     \__tag_check_if_active_mc:T
325       {
326 ⟨/generic⟩
327 ⟨*debug⟩
328 \cs_set_protected:Npn \tag_mc_begin:n #1 %#1 keyval
329   {
330     \__tag_check_if_active_mc:TF
331       {
332         \__tag_debug_mc_begin_insert:n { #1 }
```

```
333 ⟨/debug⟩
334        \group_begin: %hm
335        \__tag_check_mc_if_nested:
336        \bool_gset_true:N \g__tag_in_mc_bool
```
set default MC tags to structure:
```
337        \tl_set_eq:NN \l__tag_mc_key_tag_tl \g__tag_struct_tag_tl
338        \tl_gset_eq:NN\g__tag_mc_key_tag_tl \g__tag_struct_tag_tl
339        \keys_set:nn { __tag / mc } {#1}
340        \bool_if:NTF \l__tag_mc_artifact_bool
341          { %handle artifact
342            \__tag_mc_handle_artifact:N \l__tag_mc_artifact_type_tl
343            \exp_args:NV
344            \__tag_mc_artifact_begin_marks:n \l__tag_mc_artifact_type_tl
345          }
346          { %handle mcid type
347            \__tag_check_mc_tag:N  \l__tag_mc_key_tag_tl
348            \__tag_mc_handle_mcid:VV
349               \l__tag_mc_key_tag_tl
350               \l__tag_mc_key_properties_tl
351            \__tag_mc_begin_marks:oo{\l__tag_mc_key_tag_tl}{\l__tag_mc_key_label_tl}
352            \tl_if_empty:NF {\l__tag_mc_key_label_tl}
353              {
354                \exp_args:NV
355                \__tag_mc_handle_mc_label:n \l__tag_mc_key_label_tl
356              }
357            \bool_if:NF \l__tag_mc_key_stash_bool
358              {
359                \exp_args:NV\__tag_struct_get_tag_info:nNN
360                   \g__tag_struct_stack_current_tl
361                   \l__tag_tmpa_tl
362                   \l__tag_tmpb_tl
363                \__tag_check_parent_child:VVnnN
364                \l__tag_tmpa_tl \l__tag_tmpb_tl
365                {MC}{}
366                \l__tag_parent_child_check_tl
367                \int_compare:nNnT {\l__tag_parent_child_check_tl}<{0}
368                {
369                    \msg_warning:nnxxx
370                    { tag }
371                    {role-parent-child}
372                    { \g__tag_struct_tag_tl/\g__tag_struct_tag_NS_tl  }
373                    { MC~(=~real content) }
374                    { 'not~allowed'. }
375                }
376                \__tag_mc_handle_stash:x { \int_use:N \c@g__tag_MCID_abs_int }
377              }
378          }
379        \group_end:
380      }
381 ⟨*debug⟩
382      {
383        \__tag_debug_mc_begin_ignore:n { #1 }
384      }
385 ⟨/debug⟩
```

```
386       }
387  ⟨∗generic⟩
388  \cs_set_protected:Nn \tag_mc_end:
389    {
390       \__tag_check_if_active_mc:T
391         {
392  ⟨/generic⟩
393  ⟨∗debug⟩
394  \cs_set_protected:Nn \tag_mc_end:
395    {
396       \__tag_check_if_active_mc:TF
397         {
398            \__tag_debug_mc_end_insert:
399  ⟨/debug⟩
400            \__tag_check_mc_if_open:
401            \bool_gset_false:N \g__tag_in_mc_bool
402            \tl_gset:Nn  \g__tag_mc_key_tag_tl { }
403            \__tag_mc_emc:
404            \__tag_mc_end_marks:
405         }
406  ⟨∗debug⟩
407         {
408            \__tag_debug_mc_end_ignore:
409         }
410  ⟨/debug⟩
411    }
412  ⟨/generic | debug⟩
```

(*End definition for* `\tag_mc_begin:n` *and* `\tag_mc_end:`. *These functions are documented on page 58.*)

1.4   Keys

Definitions are different in luamode. `tag` and `raw` are expanded as `\lua_now:e` in lua does it too and we assume that their values are safe.

tag␣(mc-key)
raw␣(mc-key)
alt␣(mc-key)
actualtext␣(mc-key)
label␣(mc-key)
artifact␣(mc-key)

```
413  ⟨∗generic⟩
414  \keys_define:nn { __tag / mc }
415    {
416      tag .code:n = % the name (H,P,Span) etc
417        {
418           \tl_set:Nx   \l__tag_mc_key_tag_tl { #1 }
419           \tl_gset:Nx  \g__tag_mc_key_tag_tl { #1 }
420        },
421      raw  .code:n =
422        {
423           \tl_put_right:Nx \l__tag_mc_key_properties_tl { #1 }
424        },
425      alt .code:n      = % Alt property
426        {
427           \str_set_convert:Noon
428             \l__tag_tmpa_str
429             { #1 }
430             { default }
```

76

```
431            { utf16/hex }
432          \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
433          \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
434        },
435      alttext .meta:n = {alt=#1},
436      actualtext .code:n      = % ActualText property
437        {
438          \str_set_convert:Noon
439            \l__tag_tmpa_str
440            { #1 }
441            { default }
442            { utf16/hex }
443          \tl_put_right:Nn \l__tag_mc_key_properties_tl { /ActualText~< }
444          \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
445        },
446      label .tl_set:N         = \l__tag_mc_key_label_tl,
447      artifact .code:n        =
448        {
449          \exp_args:Nnx
450            \keys_set:nn
451              { __tag / mc }
452              { __artifact-bool, __artifact-type=#1 }
453        },
454      artifact .default:n     = {notype}
455    }
456 ⟨/generic⟩
```

(End definition for tag *(*mc-key*) and others. These functions are documented on page* *59.)*

Part VI

The **tagpdf-mc-luacode** module Code related to Marked Content (mc-chunks), luamode-specific Part of the tagpdf package

The code is splitted into three parts: code shared by all engines, code specific to luamode and code not used by luamode.

1 Marked content code – luamode code

luamode uses attributes to mark mc-chunks. The two attributes used are defined in the backend file. The backend also load the lua file, as it can contain functions needed elsewhere. The attributes for mc are global (between 0.6 and 0.81 they were local but this was reverted). The attributes are setup only in lua, and one should use the lua functions to set and get them.

`g_@@_mc_type_attr`: the value represent the type

`g_@@_mc_cnt_attr`: will hold the `\c@g_@@_MCID_abs_int` value

Handling attribute needs a different system to number the page wise mcid's: a `\tagmcbegin ... \tagmcend` pair no longer surrounds exactly one mc chunk: it can be split at page breaks. We know the included mcid(s) only after the ship out. So for the `struct -> mcid` mapping we need to record `struct -> mc-cnt` (in `\g_@@_mc_parenttree_prop` and/or a lua table and at shipout `mc-cnt-> {mcid, mcid, ...}` and when building the trees connect both.

Key definitions are overwritten for luatex to store that data in lua-tables. The data for the mc are in `ltx.@@.mc[absnum]`. The fields of the table are:

tag : the type (a string)

raw : more properties (string)

label: a string.

artifact: the presence indicates an artifact, the value (string) is the type.

kids: a array of tables

`{1={kid=num2,page=pagenum1}, 2={kid=num2,page=pagenum2},...}`,

this describes the chunks the mc has been split to by the traversing code

parent: the number of the structure it is in. Needed to build the parent tree.

```
1 ⟨@@=tag⟩
2 ⟨*luamode⟩
3 \ProvidesExplPackage {tagpdf-mc-code-lua} {2023-03-07} {0.98e}
4   {tagpdf - mc code only for the luamode }
5 ⟨/luamode⟩
```

The main function which wanders through the shipout box to inject the literals. if the new callback is there, it is used.

```
6 ⟨*luamode⟩
7 \hook_gput_code:nnn{begindocument}{tagpdf/mc}
8   {
```

```
 9    \bool_if:NT\g__tag_active_space_bool
10      {
11        \lua_now:e
12          {
13            if~luatexbase.callbacktypes.pre_shipout_filter~then~
14              luatexbase.add_to_callback("pre_shipout_filter", function(TAGBOX)~
15              ltx.__tag.func.space_chars_shipout(TAGBOX)~return~true~
16              end, "tagpdf")~
17            end
18          }
19        \lua_now:e
20          {
21            if~luatexbase.callbacktypes.pre_shipout_filter~then~
22            token.get_next()~
23            end
24          }\@secondoftwo\@gobble
25          {
26            \hook_gput_code:nnn{shipout/before}{tagpdf/lua}
27              {
28                \lua_now:e
29                  { ltx.__tag.func.space_chars_shipout (tex.box["ShipoutBox"]) }
30              }
31          }
32      }
33    \bool_if:NT\g__tag_active_mc_bool
34      {
35        \lua_now:e
36          {
37            if~luatexbase.callbacktypes.pre_shipout_filter~then~
38              luatexbase.add_to_callback("pre_shipout_filter", function(TAGBOX)~
39              ltx.__tag.func.mark_shipout(TAGBOX)~return~true~
40              end, "tagpdf")~
41            end
42          }
43        \lua_now:e
44          {
45            if~luatexbase.callbacktypes.pre_shipout_filter~then~
46            token.get_next()~
47            end
48          }\@secondoftwo\@gobble
49          {
50            \hook_gput_code:nnn{shipout/before}{tagpdf/lua}
51              {
52                \lua_now:e
53                  { ltx.__tag.func.mark_shipout (tex.box["ShipoutBox"]) }
54              }
55          }
56      }
57  }
```

1.1 Commands

`\__tag_add_missing_mcs_to_stream:Nn` This command is used in the output routine by the ptagging code. It should do nothing in luamode.

79

```
58 \cs_new_protected:Npn \__tag_add_missing_mcs_to_stream:Nn #1#2 {}
```

(*End definition for* __tag_add_missing_mcs_to_stream:Nn*.*)

__tag_mc_if_in_p:
__tag_mc_if_in:*TF*
\tag_mc_if_in_p:
\tag_mc_if_in:*TF*

This tests, if we are in an mc, for attributes this means to check against a number.

```
59 \prg_new_conditional:Nnn \__tag_mc_if_in: {p,T,F,TF}
60   {
61     \int_compare:nNnTF
62       { -2147483647 }
63       =
64       {\lua_now:e
65         {
66           tex.print(\int_use:N \c_document_cctab,tex.getattribute(luatexbase.attributes.g__ta
67         }
68       }
69       { \prg_return_false:  }
70       { \prg_return_true: }
71   }
72
73 \prg_new_eq_conditional:NNn \tag_mc_if_in: \__tag_mc_if_in: {p,T,F,TF}
```

(*End definition for* __tag_mc_if_in:*TF* *and* \tag_mc_if_in:*TF*. *This function is documented on page 58.*)

__tag_mc_lua_set_mc_type_attr:n
__tag_mc_lua_set_mc_type_attr:o
__tag_mc_lua_unset_mc_type_attr:

This takes a tag name, and sets the attributes globally to the related number.

```
74 \cs_new:Nn \__tag_mc_lua_set_mc_type_attr:n % #1 is a tag name
75   {
76     %TODO ltx.__tag.func.get_num_from("#1") seems not to return a suitable number??
77     \tl_set:Nx\l__tag_tmpa_tl{\lua_now:e{ltx.__tag.func.output_num_from ("#1")} }
78     \lua_now:e
79       {
80         tex.setattribute
81           (
82           "global",
83           luatexbase.attributes.g__tag_mc_type_attr,
84           \l__tag_tmpa_tl
85           )
86       }
87     \lua_now:e
88       {
89         tex.setattribute
90           (
91           "global",
92           luatexbase.attributes.g__tag_mc_cnt_attr,
93           \__tag_get_mc_abs_cnt:
94           )
95       }
96   }
97
98 \cs_generate_variant:Nn\__tag_mc_lua_set_mc_type_attr:n { o }
99
100 \cs_new:Nn \__tag_mc_lua_unset_mc_type_attr:
101   {
102     \lua_now:e
103       {
```

```
104        tex.setattribute
105          (
106            "global",
107            luatexbase.attributes.g__tag_mc_type_attr,
108            -2147483647
109          )
110      }
111    \lua_now:e
112      {
113        tex.setattribute
114          (
115            "global",
116            luatexbase.attributes.g__tag_mc_cnt_attr,
117            -2147483647
118          )
119      }
120  }
121
```

(*End definition for* __tag_mc_lua_set_mc_type_attr:n *and* __tag_mc_lua_unset_mc_type_attr:.)

__tag_mc_insert_mcid_kids:n
__tag_mc_insert_mcid_single_kids:n

These commands will in the finish code replace the dummy for a mc by the real mcid kids we need a variant for the case that it is the only kid, to get the array right

```
122 \cs_new:Nn \__tag_mc_insert_mcid_kids:n
123   {
124     \lua_now:e { ltx.__tag.func.mc_insert_kids (#1,0) }
125   }
126
127 \cs_new:Nn \__tag_mc_insert_mcid_single_kids:n
128   {
129     \lua_now:e {ltx.__tag.func.mc_insert_kids (#1,1) }
130   }
```

(*End definition for* __tag_mc_insert_mcid_kids:n *and* __tag_mc_insert_mcid_single_kids:n.)

__tag_mc_handle_stash:n
__tag_mc_handle_stash:x

This is the lua variant for the command to put an mcid absolute number in the current structure.

```
131 \cs_new:Nn \__tag_mc_handle_stash:n %1 mcidnum
132   {
133     \__tag_check_mc_used:n { #1 }
134     \seq_gput_right:cn % Don't fill a lua table due to the command in the item,
135                        % so use the kernel command
136       { g__tag_struct_kids_\g__tag_struct_stack_current_tl _seq }
137       {
138         \__tag_mc_insert_mcid_kids:n {#1}%
139       }
140     \lua_now:e
141       {
142         ltx.__tag.func.store_struct_mcabs
143           (
144             \g__tag_struct_stack_current_tl,#1
145           )
146       }
147     \prop_gput:Nxx
```

```
148          \g__tag_mc_parenttree_prop
149          { #1 }
150          { \g__tag_struct_stack_current_tl }
151      }
152
153  \cs_generate_variant:Nn \__tag_mc_handle_stash:n { x }
```

(*End definition for* `\__tag_mc_handle_stash:n`.)

`\tag_mc_begin:n`   This is the lua version of the user command. We currently don't check if there is nesting as it doesn't matter so much in lua.

```
154  \cs_set_protected:Nn \tag_mc_begin:n
155    {
156        \__tag_check_if_active_mc:T
157          {
158              \group_begin:
159              %\__tag_check_mc_if_nested:
160              \bool_gset_true:N \g__tag_in_mc_bool
161              \bool_set_false:N\l__tag_mc_artifact_bool
162              \tl_clear:N \l__tag_mc_key_properties_tl
163              \int_gincr:N \c@g__tag_MCID_abs_int
```

set the default tag to the structure:

```
164              \tl_set_eq:NN \l__tag_mc_key_tag_tl \g__tag_struct_tag_tl
165              \tl_gset_eq:NN\g__tag_mc_key_tag_tl \g__tag_struct_tag_tl
166              \lua_now:e
167                {
168                  ltx.__tag.func.store_mc_data(\__tag_get_mc_abs_cnt:,"tag","\g__tag_struct_tag_tl")
169                }
170              \keys_set:nn { __tag / mc }{ label={}, #1 }
171              %check that a tag or artifact has been used
172              \__tag_check_mc_tag:N \l__tag_mc_key_tag_tl
173              %set the attributes:
174              \__tag_mc_lua_set_mc_type_attr:o  { \l__tag_mc_key_tag_tl }
175              \bool_if:NF \l__tag_mc_artifact_bool
176                { % store the absolute num name in a label:
177                  \tl_if_empty:NF {\l__tag_mc_key_label_tl}
178                    {
179                        \exp_args:NV
180                          \__tag_mc_handle_mc_label:n  \l__tag_mc_key_label_tl
181                    }
182                % if not stashed record the absolute number
183                  \bool_if:NF \l__tag_mc_key_stash_bool
184                    {
185                        \exp_args:NV\__tag_struct_get_tag_info:nNN
186                          \g__tag_struct_stack_current_tl
187                          \l__tag_tmpa_tl
188                          \l__tag_tmpb_tl
189                      \__tag_check_parent_child:VVnnN
190                        \l__tag_tmpa_tl \l__tag_tmpb_tl
191                        {MC}{}
192                        \l__tag_parent_child_check_tl
193                        \int_compare:nNnT {\l__tag_parent_child_check_tl}<{0}
194                          {
195                              \msg_warning:nnxxxx
```

82

```
196                           { tag }
197                           {role-parent-child}
198                           { \g__tag_struct_tag_tl/\g__tag_struct_tag_NS_tl  }
199                           { MC~(=~real content) }
200                           { 'not~allowed'. }
201                       }
202                     \__tag_mc_handle_stash:x { \__tag_get_mc_abs_cnt: }
203                   }
204               }
205           \group_end:
206       }
207   }
```

(*End definition for* \tag_mc_begin:n. *This function is documented on page 58.*)

\tag_mc_end: TODO: check how the use command must be guarded.

```
208 \cs_set_protected:Nn \tag_mc_end:
209   {
210     \__tag_check_if_active_mc:T
211       {
212         %\__tag_check_mc_if_open:
213         \bool_gset_false:N \g__tag_in_mc_bool
214         \bool_set_false:N\l__tag_mc_artifact_bool
215         \__tag_mc_lua_unset_mc_type_attr:
216         \tl_set:Nn  \l__tag_mc_key_tag_tl { }
217         \tl_gset:Nn \g__tag_mc_key_tag_tl { }
218       }
219   }
```

(*End definition for* \tag_mc_end:. *This function is documented on page 58.*)

__tag_get_data_mc_tag: The command to retrieve the current mc tag. TODO: Perhaps this should use the
attribute instead.

```
220 \cs_new:Npn \__tag_get_data_mc_tag: { \g__tag_mc_key_tag_tl }
```

(*End definition for* __tag_get_data_mc_tag:.)

1.2   Key definitions

tag␣(mc-key)   TODO: check conversion, check if local/global setting is right.
raw␣(mc-key)
alt␣(mc-key)
actualtext␣(mc-key)
label␣(mc-key)
artifact␣(mc-key)
```
221 \keys_define:nn { __tag / mc }
222   {
223     tag .code:n = %
224       {
225         \tl_set:Nx   \l__tag_mc_key_tag_tl { #1 }
226         \tl_gset:Nx  \g__tag_mc_key_tag_tl { #1 }
227         \lua_now:e
228           {
229             ltx.__tag.func.store_mc_data(\__tag_get_mc_abs_cnt:,"tag","#1")
230           }
231       },
232     raw .code:n =
233       {
234         \tl_put_right:Nx \l__tag_mc_key_properties_tl { #1 }
```

83

```
235        \lua_now:e
236          {
237            ltx.__tag.func.store_mc_data(\__tag_get_mc_abs_cnt:,"raw","#1")
238          }
239        },
240     alt .code:n       = % Alt property
241       {
242         \str_set_convert:Noon
243           \l__tag_tmpa_str
244           { #1 }
245           { default }
246           { utf16/hex }
247         \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
248         \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
249         \lua_now:e
250           {
251             ltx.__tag.func.store_mc_data
252               (
253                 \__tag_get_mc_abs_cnt:,"alt","/Alt~<\str_use:N \l__tag_tmpa_str>"
254               )
255           }
256       },
257     alttext .meta:n = {alt=#1},
258     actualtext .code:n       = % Alt property
259       {
260         \str_set_convert:Noon
261           \l__tag_tmpa_str
262           { #1 }
263           { default }
264           { utf16/hex }
265         \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
266         \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
267         \lua_now:e
268           {
269             ltx.__tag.func.store_mc_data
270               (
271                 \__tag_get_mc_abs_cnt:,
272                 "actualtext",
273                 "/ActualText~<\str_use:N \l__tag_tmpa_str>"
274               )
275           }
276       },
277     label .code:n =
278       {
279         \tl_set:Nn\l__tag_mc_key_label_tl { #1 }
280         \lua_now:e
281           {
282             ltx.__tag.func.store_mc_data
283               (
284                 \__tag_get_mc_abs_cnt:,"label","#1"
285               )
286           }
287       },
288     __artifact-store .code:n =
```

84

```
289        {
290          \lua_now:e
291            {
292              ltx.__tag.func.store_mc_data
293                (
294                  \__tag_get_mc_abs_cnt:,"artifact","#1"
295                )
296            }
297        },
298      artifact .code:n        =
299        {
300          \exp_args:Nnx
301            \keys_set:nn
302              { __tag / mc}
303              { __artifact-bool, __artifact-type=#1, tag=Artifact }
304          \exp_args:Nnx
305            \keys_set:nn
306              { __tag / mc }
307              { __artifact-store=\l__tag_mc_artifact_type_tl }
308        },
309      artifact .default:n    = { notype }
310    }
311
312 ⟨/luamode⟩
```

(*End definition for* `tag` `(mc-key)` *and others. These functions are documented on page* .)

Part VII
The **tagpdf-struct** module
Commands to create the structure
Part of the tagpdf package

1   Public Commands

\tag_struct_begin:n
\tag_struct_end:
\tag_struct_end:n

`\tag_struct_begin:n{⟨key-values⟩}`
`\tag_struct_end:`
`\tag_struct_end:n{⟨tag⟩}`

These commands start and end a new structure. They don't start a group. They set all their values globally. `\tag_struct_end:n` does nothing special normally (apart from swallowing its argument, but if `tagpdf-debug` is loaded, it will check if the `{⟨tag⟩}` (after expansion) is identical to the current structure on the stack. The tag is not role mapped!

\tag_struct_use:n

`\tag_struct_use:n{⟨label⟩}`

These commands insert a structure previously stashed away as kid into the currently active structure. A structure should be used only once, if the structure already has a parent a warning is issued.

\tag_struct_object_ref:n
\tag_struct_object_ref:e

`\tag_struct_object_ref:n{⟨struct number⟩}`

This is a small wrapper around `\pdf_object_ref:n` to retrieve the object reference of the structure with the number ⟨*struct number*⟩. This number can be retrieved and stored for the current structure for example with `\tag_get:n{⟨struct$_n$um⟩}`. Be aware that it can only be used if the structure has already been created and that it doesn't check if the object actually exists!

The following two functions are used to add annotations. They must be used together and with care to get the same numbers. Perhaps some improvements are needed here.

\tag_struct_insert_annot:nn

`\tag_struct_insert_annot:nn{⟨object reference⟩}{⟨struct parent number⟩}`

This inserts an annotation in the structure. ⟨*object reference*⟩ is there reference to the annotation. ⟨*struct parent number*⟩ should be the same number as had been inserted with `\tag_struct_parent_int:` as `StructParent` value to the dictionary of the annotion. The command will increase the value of the counter used by `\tag_struct_parent_int:`.

\tag_struct_parent_int:

`\tag_struct_parent_int:`

This gives back the next free /StructParent number (assuming that it is together with `\tag_struct_insert_annot:nn` which will increase the number.

2 Public keys

2.1 Keys for the structure commands

tag␣(struct-key) This is required. The value of the key is normally one of the standard types listed in the main tagpdf documentation. It is possible to setup new tags/types. The value can also be of the form `type/NS`, where `NS` is the shorthand of a declared name space. Currently the names spaces `pdf`, `pdf2`, `mathml` and `user` are defined. This allows to use a different name space than the one connected by default to the tag. But normally this should not be needed.

stash␣(struct-key) Normally a new structure inserts itself as a kid into the currently active structure. This key prohibits this. The structure is nevertheless from now on "the current active structure" and parent for following marked content and structures.

label␣(struct-key) This key sets a label by which one can refer to the structure. It is e.g. used by `\tag_-struct_use:n` (where a real label is actually not needed as you can only use structures already defined), and by the `ref` key (which can refer to future structures). Internally the label name will start with `tagpdfstruct-` and it stores the two attributes `tagstruct` (the structure number) and `tagstructobj` (the object reference).

parent␣(struct-key) By default a structure is added as kid to the currently active structure. With the parent key one can choose another parent. The value is a structure number which must refer to an already existing, previously created structure. Such a structure number can for example be have been stored with `\tag_get:n`, but one can also use a label on the parent structure and then use `\ref_value:nn{tagpdfstruct-label}{tagstruct}` to retrieve it.

title␣(struct-key)
title-o␣(struct-key) This keys allows to set the dictionary entry `/Title` in the structure object. The value is handled as verbatim string and hex encoded. Commands are not expanded. `title-o` will expand the value once.

alt␣(struct-key) This key inserts an `/Alt` value in the dictionary of structure object. The value is handled as verbatim string and hex encoded. The value will be expanded first once.

actualtext␣(struct-key) This key inserts an `/ActualText` value in the dictionary of structure object. The value is handled as verbatim string and hex encoded. The value will be expanded first once.

lang␣(struct-key) This key allows to set the language for a structure element. The value should be a bcp-identifier, e.g. `de-De`.

`ref`␣(struct-key) This key allows to add references to other structure elements, it adds the `/Ref` array to the structure. The value should be a comma separated list of structure labels set with the `label` key. e.g. `ref={label1,label2}`.

`E`␣(struct-key) This key sets the `/E` key, the expanded form of an abbreviation or an acronym (I couldn't think of a better name, so I sticked to E).

`AF`␣(struct-key)
`AFinline`␣(struct-key)
`AFinline-o`␣(struct-key)

`AF = ⟨object name⟩`
`AF-inline = ⟨text content⟩`

These keys allows to reference an associated file in the structure element. The value ⟨*object name*⟩ should be the name of an object pointing to the `/Filespec` dictionary as expected by `\pdf_object_ref:n` from a current `l3kernel`.

The value `AF-inline` is some text, which is embedded in the PDF as a text file with mime type text/plain. `AF-inline-o` is like `AF-inline` but expands the value once.

Future versions will perhaps extend this to more mime types, but it is still a research task to find out what is really needed.

`AF` can be used more than once, to associate more than one file. The inline keys can be used only once per structure. Additional calls are ignored.

`attribute`␣(struct-key) This key takes as argument a comma list of attribute names (use braces to protect the commas from the external key-val parser) and allows to add one or more attribute dictionary entries in the structure object. As an example

> `\tagstructbegin{tag=TH,attribute= TH-row}`

Attribute names and their content must be declared first in `\tagpdfsetup`.

`attribute-class`␣(struct-key)

This key takes as argument a comma list of attribute class names (use braces to protect the commas from the external key-val parser) and allows to add one or more attribute classes to the structure object.

Attribute class names and their content must be declared first in `\tagpdfsetup`.

2.2 Setup keys

<table>
<tr><td><code>newattribute␣(setup-key)</code></td><td><code>newattribute = {⟨name⟩}{⟨Content⟩}</code></td></tr>
</table>

This key can be used in the setup command `\tagpdfsetup` and allow to declare a new attribute, which can be used as attribute or attribute class. The value are two brace groups, the first contains the name, the second the content.

```
\tagpdfsetup
 {
  newattribute =
   {TH-col}{/O /Table /Scope /Column},
  newattribute =
   {TH-row}{/O /Table /Scope /Row},
  }
```

<table>
<tr><td><code>root-AF␣(setup-key)</code></td><td><code>root-AF = ⟨object name⟩</code></td></tr>
</table>

This key can be used in the setup command `\tagpdfsetup` and allows to add associated files to the root structure. Like `AF` it can be used more than once to add more than one file.

```
1 ⟨@@=tag⟩
2 ⟨*header⟩
3 \ProvidesExplPackage {tagpdf-struct-code} {2023-03-07} {0.98e}
4  {part of tagpdf - code related to storing structure}
5 ⟨/header⟩
```

3 Variables

`\c@g__tag_struct_abs_int` Every structure will have a unique, absolute number. I will use a latex counter for the structure count to have a chance to avoid double structures in align etc.

```
6 ⟨base⟩\newcounter  { g__tag_struct_abs_int }
7 ⟨base⟩\int_gzero:N \c@g__tag_struct_abs_int
```

(*End definition for* `\c@g__tag_struct_abs_int`.)

`\g__tag_struct_objR_seq` a sequence to store mapping between the structure number and the object number. We assume that structure numbers are assign consecutively and so the index of the seq can be used. A seq allows easy mapping over the structures.

```
8 ⟨*package⟩
9 \__tag_seq_new:N  \g__tag_struct_objR_seq
```

(*End definition for* `\g__tag_struct_objR_seq`.)

`\g__tag_struct_cont_mc_prop` in generic mode it can happen after a page break that we have to inject into a structure sequence an additional mc after. We will store this additional info in a property. The key is the absolut mc num, the value the pdf directory.

```
10 \__tag_prop_new:N  \g__tag_struct_cont_mc_prop
```

(*End definition for* `\g__tag_struct_cont_mc_prop`.)

`\g__tag_struct_stack_seq` A stack sequence for the structure stack. When a sequence is opened it's number is put on the stack.

```
11 \seq_new:N      \g__tag_struct_stack_seq
12 \seq_gpush:Nn \g__tag_struct_stack_seq {0}
```

(*End definition for* `\g__tag_struct_stack_seq`.)

`\g__tag_struct_tag_stack_seq` We will perhaps also need the tags. While it is possible to get them from the numbered stack, lets build a tag stack too.

```
13 \seq_new:N      \g__tag_struct_tag_stack_seq
14 \seq_gpush:Nn \g__tag_struct_tag_stack_seq {{Root}{StructTreeRoot}}
```

(*End definition for* `\g__tag_struct_tag_stack_seq`.)

`\g__tag_struct_tag_NS_prop` For the parent-child check, we need the tag and NS of every structure

```
15 \prop_new:N\g__tag_struct_tag_NS_prop
```

(*End definition for* `\g__tag_struct_tag_NS_prop`.)

`\g_tag_struct_stack_current_tl`
`\l_tag_struct_stack_parent_tmpa_tl`
The global variable will hold the current structure number. It is already defined in `tagpdf-base`. The local temporary variable will hold the parent when we fetch it from the stack.

```
16 ⟨/package⟩
17 ⟨base⟩\tl_new:N  \g__tag_struct_stack_current_tl
18 ⟨base⟩\tl_gset:Nn \g__tag_struct_stack_current_tl {\int_use:N\c@g__tag_struct_abs_int}
19 ⟨*package⟩
20 \tl_new:N      \l__tag_struct_stack_parent_tmpa_tl
```

(*End definition for* `\g__tag_struct_stack_current_tl` *and* `\l__tag_struct_stack_parent_tmpa_tl`.)

I will need at least one structure: the StructTreeRoot normally it should have only one kid, e.g. the document element.

The data of the StructTreeRoot and the StructElem are in properties: `\g_@@_struct_0_prop` for the root and `\g_@@_struct_N_prop`, $N \geq= 1$ for the other.

This creates quite a number of properties, so perhaps we will have to do this more efficiently in the future.

All properties have at least the keys

Type StructTreeRoot or StructElem

and the keys from the two following lists (the root has a special set of properties). the values of the prop should be already escaped properly when the entries are created (title,lange,alt,E,actualtext)

`\c_tag_struct_StructTreeRoot_entries_seq`
`\c__tag_struct_StructElem_entries_seq`
These seq contain the keys we support in the two object types. They are currently no longer used, but are provided as documentation and for potential future checks. They should be adapted if there are changes in the PDF format.

```
21 \seq_const_from_clist:Nn \c__tag_struct_StructTreeRoot_entries_seq
22   {%p. 857/858
23    Type,              % always /StructTreeRoot
24    K,                 % kid, dictionary or array of dictionaries
25    IDTree,            % currently unused
26    ParentTree,        % required,obj ref to the parent tree
27    ParentTreeNextKey, % optional
28    RoleMap,
```

90

```
29    ClassMap,
30    Namespaces,
31    AF                  %pdf 2.0
32  }
33
34 \seq_const_from_clist:Nn \c__tag_struct_StructElem_entries_seq
35  {%p 858 f
36    Type,              %always /StructElem
37    S,                 %tag/type
38    P,                 %parent
39    ID,                %optional
40    Ref,               %optional, pdf 2.0 Use?
41    Pg,                %obj num of starting page, optional
42    K,                 %kids
43    A,                 %attributes, probably unused
44    C,                 %class ""
45    %R,                %attribute revision number, irrelevant for us as we
46                       % don't update/change existing PDF and (probably)
47                       % deprecated in PDF 2.0
48    T,                 %title, value in () or <>
49    Lang,              %language
50    Alt,               % value in () or <>
51    E,                 % abreviation
52    ActualText,
53    AF,                %pdf 2.0, array of dict, associated files
54    NS,                %pdf 2.0, dict, namespace
55    PhoneticAlphabet,  %pdf 2.0
56    Phoneme            %pdf 2.0
57  }
```

(*End definition for* \c__tag_struct_StructTreeRoot_entries_seq *and* \c__tag_struct_StructElem_-entries_seq.)

3.1 Variables used by the keys

\g__tag_struct_tag_tl
\g__tag_struct_tag_NS_tl

Use by the tag key to store the tag and the namespace.

```
58 \tl_new:N \g__tag_struct_tag_tl
59 \tl_new:N \g__tag_struct_tag_NS_tl
```

(*End definition for* \g__tag_struct_tag_tl *and* \g__tag_struct_tag_NS_tl.)

\l__tag_struct_key_label_tl

This will hold the label value.

```
60 \tl_new:N \l__tag_struct_key_label_tl
```

(*End definition for* \l__tag_struct_key_label_tl.)

\l__tag_struct_elem_stash_bool

This will keep track of the stash status

```
61 \bool_new:N \l__tag_struct_elem_stash_bool
```

(*End definition for* \l__tag_struct_elem_stash_bool.)

3.2 Variables used by tagging code of basic elements

`\g__tag_struct_dest_num_prop` This variable records for (some or all, not clear yet) destination names the related structure number to allow to reference them in a Ref. The key is the destination. It is currently used by the toc-tagging and sec-tagging code.

```
62 ⟨/package⟩
63 ⟨base⟩\prop_new:N \g__tag_struct_dest_num_prop
64 ⟨*package⟩
```

(*End definition for* `\g__tag_struct_dest_num_prop`.)

`\g_tag_struct_ref_by_dest_prop` This variable contains structures whose Ref key should be updated at the end to point to structured related with this destination. As this is probably need in other places too, it is not only a toc-variable.

```
65 \prop_new:N \g__tag_struct_ref_by_dest_prop
```

(*End definition for* `\g__tag_struct_ref_by_dest_prop`.)

4 Commands

The properties must be in some places handled expandably. So I need an output handler for each prop, to get expandable output see https://tex.stackexchange.com/questions/424208. There is probably room here for a more efficient implementation. TODO check if this can now be implemented with the pdfdict commands. The property contains currently non pdf keys, but e.g. object numbers are perhaps no longer needed as we have named object anyway.

`\__tag_struct_output_prop_aux:nn`
`\__tag_new_output_prop_handler:n`

```
66 \cs_new:Npn \__tag_struct_output_prop_aux:nn #1 #2 %#1 num, #2 key
67   {
68     \prop_if_in:cnT
69       { g__tag_struct_#1_prop }
70       { #2 }
71       {
72         \c_space_tl/#2~ \prop_item:cn{ g__tag_struct_#1_prop } { #2 }
73       }
74   }
75
76 \cs_new_protected:Npn \__tag_new_output_prop_handler:n #1
77   {
78     \cs_new:cn { __tag_struct_output_prop_#1:n }
79       {
80         \__tag_struct_output_prop_aux:nn {#1}{##1}
81       }
82   }
```

(*End definition for* `\__tag_struct_output_prop_aux:nn` *and* `\__tag_new_output_prop_handler:n`.)

4.1 Initialization of the StructTreeRoot

The first structure element, the StructTreeRoot is special, so created manually. The underlying object is `@@/struct/0` which is currently created in the tree code (TODO move it here). The `ParentTree` and `RoleMap` entries are added at begin document in the tree code as they refer to object which are setup in other parts of the code. This avoid timing issues.

```
83 \tl_gset:Nn \g__tag_struct_stack_current_tl {0}
```

`\__tag_pdf_name_e:n`

```
84 \cs_new:Npn \__tag_pdf_name_e:n #1{\pdf_name_from_unicode_e:n{#1}}
```

(*End definition for* `\__tag_pdf_name_e:n`.)

`g__tag_struct_0_prop`
`g__tag_struct_kids_0_seq`

```
85 \__tag_prop_new:c { g__tag_struct_0_prop }
86 \__tag_new_output_prop_handler:n {0}
87 \__tag_seq_new:c  { g__tag_struct_kids_0_seq }
88
89 \__tag_prop_gput:cnx
90   { g__tag_struct_0_prop }
91   { Type }
92   { \pdf_name_from_unicode_e:n {StructTreeRoot} }
93
94 \__tag_prop_gput:cnx
95   { g__tag_struct_0_prop }
96   { S }
97   { \pdf_name_from_unicode_e:n {StructTreeRoot} }
98
99 \prop_gput:Nnn \g__tag_struct_tag_NS_prop {0}{{StructTreeRoot}{pdf}}
```

Namespaces are pdf 2.0 but it doesn't harm to have an empty entry. We could add a test, but if the code moves into the kernel, timing could get tricky.

```
100 \__tag_prop_gput:cnx
101   { g__tag_struct_0_prop }
102   { Namespaces }
103   { \pdf_object_ref:n { __tag/tree/namespaces } }
```

(*End definition for* `g__tag_struct_0_prop` *and* `g__tag_struct_kids_0_seq`.)

4.2 Adding the /ID key

Every structure gets automatically an ID which is currently simply calculated from the structure number.

`\__tag_struct_get_id:n`

```
104 \cs_new:Npn \__tag_struct_get_id:n #1 %#1=struct num
105   {
106     (
107       ID.
108       \prg_replicate:nn
109       { \int_abs:n{\g__tag_tree_id_pad_int - \tl_count:e { \int_to_arabic:n { #1 } }} }
110       { 0 }
111       \int_to_arabic:n { #1 }
```

```
112      )
113    }
```

(*End definition for* `\__tag_struct_get_id:n.`)

4.3 Filling in the tag info

`\__tag_struct_set_tag_info:nnn` This adds or updates the tag info to a structure given by a number. We need also the original data, so we store both.

```
114 \pdf_version_compare:NnTF < {2.0}
115 {
116   \cs_new_protected:Npn \__tag_struct_set_tag_info:nnn #1 #2 #3
117     %#1 structure number, #2 tag, #3 NS
118     {
119       \__tag_prop_gput:cnx
120         { g__tag_struct_#1_prop }
121         { S }
122         { \pdf_name_from_unicode_e:n {#2}  } %
123       \prop_gput:Nnn \g__tag_struct_tag_NS_prop {#1}{{#2}{#3}}
124     }
125 }
126 {
127   \cs_new_protected:Npn \__tag_struct_set_tag_info:nnn #1 #2 #3
128     {
129       \__tag_prop_gput:cnx
130         { g__tag_struct_#1_prop }
131         { S }
132         { \pdf_name_from_unicode_e:n {#2} } %
133       \prop_get:NnNT \g__tag_role_NS_prop {#3} \l__tag_get_tmpc_tl
134         {
135           \__tag_prop_gput:cnx
136             { g__tag_struct_#1_prop }
137             { NS }
138             { \l__tag_get_tmpc_tl } %
139         }
140       \prop_gput:Nnn \g__tag_struct_tag_NS_prop {#1}{{#2}{#3}}
141     }
142 }
143 \cs_generate_variant:Nn \__tag_struct_set_tag_info:nnn {eVV}
```

(*End definition for* `\__tag_struct_set_tag_info:nnn.`)

`\__tag_struct_get_tag_info:nNN` We also need a way to get the tag info back from parent structures.

```
144 \cs_new_protected:Npn \__tag_struct_get_tag_info:nNN #1 #2 #3
145   %#1 struct num, #2 tlvar for tag , #3 tlvar for NS
146   {
147     \prop_get:NnNTF \g__tag_struct_tag_NS_prop {#1}\l__tag_get_tmpc_tl
148       {
149         \tl_set:Nx #2{\exp_last_unbraced:NV\use_i:nn  \l__tag_get_tmpc_tl}
150         \tl_set:Nx #3{\exp_last_unbraced:NV\use_ii:nn \l__tag_get_tmpc_tl}
151       }
152       {
153         \tl_clear:N#2
154         \tl_clear:N#3
```

```
155                 }
156             }
157 \cs_generate_variant:Nn\__tag_struct_get_tag_info:nNN {eNN}
```

(*End definition for* `\__tag_struct_get_tag_info:nNN.`)

4.4 Handlings kids

Commands to store the kids. Kids in a structure can be a reference to a mc-chunk, an object reference to another structure element, or a object reference to an annotation (through an OBJR object).

`\__tag_struct_kid_mc_gput_right:nn`
`\__tag_struct_kid_mc_gput_right:nx`

The command to store an mc-chunk, this is a dictionary of type MCR. It would be possible to write out the content directly as unnamed object and to store only the object reference, but probably this would be slower, and the PDF is more readable like this. The code doesn't try to avoid the use of the /Pg key by checking page numbers. That imho only slows down without much gain. In generic mode the page break code will perhaps to have to insert an additional mcid after an existing one. For this we use a property list At first an auxiliary to write the MCID dict. This should normally be expanded!

```
158 \cs_new:Npn \__tag_struct_mcid_dict:n #1 %#1 MCID absnum
159   {
160     <<
161      /Type \c_space_tl /MCR \c_space_tl
162      /Pg
163        \c_space_tl
164      \pdf_pageobject_ref:n { \__tag_ref_value:enn{mcid-#1}{tagabspage}{1} }
165      /MCID \c_space_tl \__tag_ref_value:enn{mcid-#1}{tagmcid}{1}
166     >>
167   }
168 \cs_new_protected:Npn \__tag_struct_kid_mc_gput_right:nn #1 #2 %#1 structure num, #2 MCID absr
169   {
170     \__tag_seq_gput_right:cx
171       { g__tag_struct_kids_#1_seq }
172       {
173         \__tag_struct_mcid_dict:n {#2}
174       }
175     \__tag_seq_gput_right:cn
176       { g__tag_struct_kids_#1_seq }
177       {
178         \prop_item:Nn \g__tag_struct_cont_mc_prop {#2}
179       }
180   }
181 \cs_generate_variant:Nn \__tag_struct_kid_mc_gput_right:nn {nx}
182
```

(*End definition for* `\__tag_struct_kid_mc_gput_right:nn.`)

`\__tag_struct_kid_struct_gput_right:nn`
`\__tag_struct_kid_struct_gput_right:xx`

This commands adds a structure as kid. We only need to record the object reference in the sequence.

```
183 \cs_new_protected:Npn\__tag_struct_kid_struct_gput_right:nn #1 #2 %#1 num of parent struct, #2
184   {
185     \__tag_seq_gput_right:cx
186       { g__tag_struct_kids_#1_seq }
```

```
187      {
188          \pdf_object_ref:n { __tag/struct/#2 }
189      }
190  }
191
192  \cs_generate_variant:Nn \__tag_struct_kid_struct_gput_right:nn {xx}
```

(*End definition for* `\__tag_struct_kid_struct_gput_right:nn`.)

`\__tag_struct_kid_OBJR_gput_right:nnn`
`\__tag_struct_kid_OBJR_gput_right:xxx`

At last the command to add an OBJR object. This has to write an object first. The first argument is the number of the parent structure, the second the (expanded) object reference of the annotation. The last argument is the page object reference

```
193  \cs_new_protected:Npn\__tag_struct_kid_OBJR_gput_right:nnn #1 #2 #3 %#1 num of parent struct,
194                                                                      %#2 obj reference
195                                                                      %#3 page object reference
196  {
197      \pdf_object_unnamed_write:nn
198        { dict }
199        {
200          /Type/OBJR/Obj~#2/Pg~#3
201        }
202      \__tag_seq_gput_right:cx
203        { g__tag_struct_kids_#1_seq }
204        {
205          \pdf_object_ref_last:
206        }
207  }
208
209  \cs_generate_variant:Nn\__tag_struct_kid_OBJR_gput_right:nnn { xxx }
210
```

(*End definition for* `\__tag_struct_kid_OBJR_gput_right:nnn`.)

`\__tag_struct_exchange_kid_command:N`
`\__tag_struct_exchange_kid_command:c`

In luamode it can happen that a single kid in a structure is split at a page break into two or more mcid. In this case the lua code has to convert put the dictionary of the kid into an array. See issue 13 at tagpdf repo. We exchange the dummy command for the kids to mark this case.

```
211  \cs_new_protected:Npn\__tag_struct_exchange_kid_command:N #1 %#1 = seq var
212  {
213      \seq_gpop_left:NN #1 \l__tag_tmpa_tl
214      \regex_replace_once:nnN
215        { \c{\__tag_mc_insert_mcid_kids:n} }
216        { \c{\__tag_mc_insert_mcid_single_kids:n} }
217        \l__tag_tmpa_tl
218      \seq_gput_left:NV #1 \l__tag_tmpa_tl
219  }
220
221  \cs_generate_variant:Nn\__tag_struct_exchange_kid_command:N { c }
```

(*End definition for* `\__tag_struct_exchange_kid_command:N`.)

`\__tag_struct_fill_kid_key:n`

This command adds the kid info to the K entry. In lua mode the content contains commands which are expanded later. The argument is the structure number.

```
222  \cs_new_protected:Npn \__tag_struct_fill_kid_key:n #1 %#1 is the struct num
```

```
223    {
224      \bool_if:NF\g__tag_mode_lua_bool
225        {
226          \seq_clear:N \l__tag_tmpa_seq
227          \seq_map_inline:cn { g__tag_struct_kids_#1_seq }
228           { \seq_put_right:Nx \l__tag_tmpa_seq { ##1 } }
229          %\seq_show:c { g__tag_struct_kids_#1_seq }
230          %\seq_show:N \l__tag_tmpa_seq
231          \seq_remove_all:Nn \l__tag_tmpa_seq {}
232          %\seq_show:N \l__tag_tmpa_seq
233          \seq_gset_eq:cN { g__tag_struct_kids_#1_seq } \l__tag_tmpa_seq
234        }
235
236      \int_case:nnF
237        {
238          \seq_count:c
239            {
240              g__tag_struct_kids_#1_seq
241            }
242        }
243        {
244          { 0 }
245           { } %no kids, do nothing
246          { 1 } % 1 kid, insert
247            {
248              % in this case we need a special command in
249              % luamode to get the array right. See issue #13
250              \bool_if:NT\g__tag_mode_lua_bool
251                {
252                  \__tag_struct_exchange_kid_command:c
253                  {g__tag_struct_kids_#1_seq}
254                }
255              \__tag_prop_gput:cnx { g__tag_struct_#1_prop } {K}
256                {
257                  \seq_item:cn
258                    {
259                      g__tag_struct_kids_#1_seq
260                    }
261                  {1}
262                }
263            } %
264        }
265        { %many kids, use an array
266          \__tag_prop_gput:cnx { g__tag_struct_#1_prop } {K}
267            {
268              [
269                \seq_use:cn
270                  {
271                    g__tag_struct_kids_#1_seq
272                  }
273                  {
274                    \c_space_tl
275                  }
276              ]
```

```
277            }
278        }
279    }
280
```

(*End definition for* `\__tag_struct_fill_kid_key:n`.)

4.5 Output of the object

`\__tag_struct_get_dict_content:nN`   This maps the dictionary content of a structure into a tl-var. Basically it does what `\pdfdict_use:n` does. TODO!! this looks over-complicated. Check if it can be done with pdfdict now.

```
281 \cs_new_protected:Npn \__tag_struct_get_dict_content:nN #1 #2 %#1: stucture num
282    {
283        \tl_clear:N #2
284        \seq_map_inline:cn
285          {
286            c__tag_struct_
287            \int_compare:nNnTF{#1}={0}{StructTreeRoot}{StructElem}
288            _entries_seq
289          }
290          {
291            \tl_put_right:Nx
292              #2
293              {
294                \prop_if_in:cnT
295                { g__tag_struct_#1_prop }
296                { ##1 }
297                {
298                    \c_space_tl/##1~
```

Some keys needs the option to format the key, e.g. add brackets for an array

```
299                    \cs_if_exist_use:cTF {__tag_struct_format_##1:e}
300                      {
301                        { \prop_item:cn{ g__tag_struct_#1_prop } { ##1 } } }
302                      }
303                      {
304                        \prop_item:cn{ g__tag_struct_#1_prop } { ##1 }
305                      }
306                }
307              }
308          }
309    }
```

(*End definition for* `\__tag_struct_get_dict_content:nN`.)

`\__tag_struct_format_Ref:n`   Ref is an array, we store only the content to be able to extend it so the formatting command adds the brackets:

```
310 \cs_new:Nn\__tag_struct_format_Ref:n{[#1]}
311 \cs_generate_variant:Nn\__tag_struct_format_Ref:n{e}
```

(*End definition for* `\__tag_struct_format_Ref:n`.)

`\__tag_struct_write_obj:n` This writes out the structure object. This is done in the finish code, in the tree module and guarded by the tree boolean.

```
312 \cs_new_protected:Npn \__tag_struct_write_obj:n #1 % #1 is the struct num
313   {
314     \pdf_object_if_exist:nTF { __tag/struct/#1 }
315       {
316         \__tag_struct_fill_kid_key:n { #1 }
317         \__tag_struct_get_dict_content:nN { #1 } \l__tag_tmpa_tl
318         \exp_args:Nx
319           \pdf_object_write:nnx
320             { __tag/struct/#1 }
321             {dict}
322             {
323               \l__tag_tmpa_tl\c_space_tl
324               /ID~\__tag_struct_get_id:n{#1}
325             }
326       }
327       {
328         \msg_error:nnn { tag } { struct-no-objnum } { #1}
329       }
330   }
```

(*End definition for* `\__tag_struct_write_obj:n`.)

`\__tag_struct_insert_annot:nn` This is the command to insert an annotation into the structure. It can probably be used for xform too.

Annotations used as structure content must

1. add a StructParent integer to their dictionary

2. push the object reference as OBJR object in the structure

3. Add a Structparent/obj-nr reference to the parent tree.

For a link this looks like this

```
        \tag_struct_begin:n { tag=Link }
        \tag_mc_begin:n { tag=Link }
(1)     \pdfannot_dict_put:nnx
          { link/URI }
          { StructParent }
          { \int_use:N\c@g_@@_parenttree_obj_int }
   <start link> link text <stop link>
(2+3)   \@@_struct_insert_annot:nn {obj ref}{parent num}
        \tag_mc_end:
        \tag_struct_end:
```

```
331 \cs_new_protected:Npn \__tag_struct_insert_annot:nn #1 #2 %#1 object reference to the annotat
332                                                            %#2 structparent number
333   {
334     \bool_if:NT \g__tag_active_struct_bool
335       {
336         %get the number of the parent structure:
337         \seq_get:NNF
338           \g__tag_struct_stack_seq
```

99

```
339              \l__tag_struct_stack_parent_tmpa_tl
340              {
341                \msg_error:nn { tag } { struct-faulty-nesting }
342              }
343          %put the obj number of the annot in the kid entry, this also creates
344          %the OBJR object
345          \ref_label:nn {__tag_objr_page_#2 }{ tagabspage }
346          \__tag_struct_kid_OBJR_gput_right:xxx
347              {
348                \l__tag_struct_stack_parent_tmpa_tl
349              }
350              {
351                #1 %
352              }
353              {
354                \pdf_pageobject_ref:n { \__tag_ref_value:nnn {__tag_objr_page_#2 }{ tagabspage }{1
355              }
356          % add the parent obj number to the parent tree:
357          \exp_args:Nnx
358          \__tag_parenttree_add_objr:nn
359              {
360                #2
361              }
362              {
363                \pdf_object_ref:e { __tag/struct/\l__tag_struct_stack_parent_tmpa_tl }
364              }
365          % increase the int:
366          \stepcounter{ g__tag_parenttree_obj_int }
367        }
368    }
```

(*End definition for* __tag_struct_insert_annot:nn.)

__tag_get_data_struct_tag: this command allows \tag_get:n to get the current structure tag with the keyword struct_tag.

```
369 \cs_new:Npn \__tag_get_data_struct_tag:
370   {
371     \exp_args:Ne
372     \tl_tail:n
373       {
374         \prop_item:cn {g__tag_struct_\g__tag_struct_stack_current_tl _prop}{S}
375       }
376   }
```

(*End definition for* __tag_get_data_struct_tag:.)

__tag_get_data_struct_id: this command allows \tag_get:n to get the current structure id with the keyword struct_id.

```
377 \cs_new:Npn \__tag_get_data_struct_id:
378   {
379     \__tag_struct_get_id:n {\g__tag_struct_stack_current_tl}
380   }
381 ⟨/package⟩
```

(*End definition for* __tag_get_data_struct_id:.)

`\__tag_get_data_struct_num:` this command allows `\tag_get:n` to get the current structure number with the keyword `struct_num`. We will need to handle nesting

```
382 ⟨∗base⟩
383 \cs_new:Npn \__tag_get_data_struct_num:
384   {
385     \g__tag_struct_stack_current_tl
386   }
387 ⟨/base⟩
```

(*End definition for* `\__tag_get_data_struct_num:`.)

5 Keys

This are the keys for the user commands. we store the tag in a variable. But we should be careful, it is only reliable at the begin.

label␣(struct-key)
stash␣(struct-key)
parent␣(struct-key)
tag␣(struct-key)
title␣(struct-key)
title-o␣(struct-key)
alt␣(struct-key)
actualtext␣(struct-key)
lang␣(struct-key)
ref␣(struct-key)
E␣(struct-key)

```
388 ⟨∗package⟩
389 \keys_define:nn { __tag / struct }
390   {
391     label .tl_set:N      = \l__tag_struct_key_label_tl,
392     stash .bool_set:N    = \l__tag_struct_elem_stash_bool,
393     parent .code:n       =
394       {
395         \bool_lazy_and:nnTF
396           {
397             \prop_if_exist_p:c { g__tag_struct_\int_eval:n {#1}_prop }
398           }
399           {
400             \int_compare_p:nNn {#1}<{\c@g__tag_struct_abs_int}
401           }
402           { \tl_set:Nx \l__tag_struct_stack_parent_tmpa_tl { \int_eval:n {#1} } }
403           {
404             \msg_warning:nnxx { tag } { struct-unknown }
405               { \int_eval:n {#1} }
406               { parent~key~ignored }
407           }
408       },
409     parent .default:n    = {-1},
410     tag   .code:n        = % S property
411       {
412         \seq_set_split:Nne \l__tag_tmpa_seq { / } {#1/\prop_item:Ne\g__tag_role_tags_NS_prop{#
413         \tl_gset:Nx \g__tag_struct_tag_tl   { \seq_item:Nn\l__tag_tmpa_seq {1} }
414         \tl_gset:Nx \g__tag_struct_tag_NS_tl{ \seq_item:Nn\l__tag_tmpa_seq {2} }
415         \__tag_check_structure_tag:N \g__tag_struct_tag_tl
416       },
417     title .code:n        = % T property
418       {
419         \str_set_convert:Nnnn
420           \l__tag_tmpa_str
421           { #1 }
422           { default }
423           { utf16/hex }
```

101

```
424        \__tag_prop_gput:cnx
425          { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
426          { T }
427          { <\l__tag_tmpa_str> }
428      },
429    title-o .code:n       = % T property
430      {
431        \str_set_convert:Nonn
432          \l__tag_tmpa_str
433          { #1 }
434          { default }
435          { utf16/hex }
436        \__tag_prop_gput:cnx
437          { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
438          { T }
439          { <\l__tag_tmpa_str> }
440      },
441    alt .code:n       = % Alt property
442      {
443        \str_set_convert:Noon
444          \l__tag_tmpa_str
445          { #1 }
446          { default }
447          { utf16/hex }
448        \__tag_prop_gput:cnx
449          { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
450          { Alt }
451          { <\l__tag_tmpa_str> }
452      },
453    alttext .meta:n = {alt=#1},
454    actualtext .code:n  = % ActualText property
455      {
456        \str_set_convert:Noon
457          \l__tag_tmpa_str
458          { #1 }
459          { default }
460          { utf16/hex }
461        \__tag_prop_gput:cnx
462          { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
463          { ActualText }
464          { <\l__tag_tmpa_str>}
465      },
466    lang .code:n       = % Lang property
467      {
468        \__tag_prop_gput:cnx
469          { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
470          { Lang }
471          { (#1) }
472      },
```

Ref is an array, the brackets are added through the formatting command.

```
473    ref .code:n       = % ref property
474      {
475        \tl_clear:N\l__tag_tmpa_tl
476        \clist_map_inline:on {#1}
```

```
477                 {
478                   \tl_put_right:Nx \l__tag_tmpa_tl
479                     {~\ref_value:nn{tagpdfstruct-##1}{tagstructobj} }
480                 }
481             \__tag_struct_gput_data_ref:ee { \int_eval:n {\c@g__tag_struct_abs_int} } {\l__tag_tmp
482           },
483        E .code:n        = % E property
484           {
485             \str_set_convert:Nnon
486               \l__tag_tmpa_str
487               { #1 }
488               { default }
489               { utf16/hex }
490             \__tag_prop_gput:cnx
491               { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
492               { E }
493               { <\l__tag_tmpa_str> }
494        },
495     }
```

(*End definition for* label (struct-key) *and others. These functions are documented on page 87.*)

AF␣(struct-key) keys for the AF keys (associated files). They use commands from l3pdffile! The stream
AFinline␣(struct-key) variants use txt as extension to get the mimetype. TODO: check if this should be
AFinline-o␣(struct-key) configurable. For math we will perhaps need another extension. AF is an array and can
be used more than once, so we store it in a tl. which is expanded. AFinline currently uses
the fix extention txt. texsource is a special variant which creates a tex-file, it expects a
tl-var as value (e.g. from math grabbing)

This variable is used to number the AF-object names

```
496  \int_new:N\g__tag_struct_AFobj_int
```

\g__tag_struct_AFobj_int
```
497  \cs_if_free:NTF \pdffile_embed_stream:nnN
498  {
499    \cs_new_protected:Npn \__tag_struct_add_inline_AF:nn #1 #2
500      % #1 content, #2 extension
501      {
502        \group_begin:
503        \int_gincr:N \g__tag_struct_AFobj_int
504        \pdf_object_if_exist:eF {__tag/fileobj\int_use:N\g__tag_struct_AFobj_int}
505          {
506            \pdffile_embed_stream:nxx
507              {#1}
508              {tag-AFfile\int_use:N\g__tag_struct_AFobj_int.#2}
509              {__tag/fileobj\int_use:N\g__tag_struct_AFobj_int}
510            \__tag_struct_add_AF:ee
511              { \int_eval:n {\c@g__tag_struct_abs_int} }
512              { \pdf_object_ref:e {__tag/fileobj\int_use:N\g__tag_struct_AFobj_int} }
513            \__tag_prop_gput:cnx
514              { g__tag_struct_\int_use:N\c@g__tag_struct_abs_int _prop }
515              { AF }
516              {
517                [
518                  \tl_use:c
```

```
519                       { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
520                   ]
521                 }
522               }
523           \group_end:
524         }
525     }
526     {
527       \cs_generate_variant:Nn \pdffile_embed_stream:nnN {nxN}
528       \cs_new_protected:Npn \__tag_struct_add_inline_AF:nn #1 #2
529       % #1 content, #2 extension
530        {
531           \group_begin:
532           \int_gincr:N \g__tag_struct_AFobj_int
533           \pdffile_embed_stream:nxN
534             {#1}
535             {tag-AFfile\int_use:N\g__tag_struct_AFobj_int.#2}
536             \l__tag_tmpa_tl
537           \__tag_struct_add_AF:ee
538             { \int_eval:n {\c@g__tag_struct_abs_int} }
539             { \l__tag_tmpa_tl }
540           \__tag_prop_gput:cnx
541             { g__tag_struct_\int_use:N\c@g__tag_struct_abs_int _prop }
542             { AF }
543             {
544               [
545                 \tl_use:c
546                   { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
547               ]
548             }
549         \group_end:
550       }
551     }
552 \cs_generate_variant:Nn \__tag_struct_add_inline_AF:nn {on}
553 \cs_new_protected:Npn \__tag_struct_add_AF:nn #1 #2 % #1 struct num #2 object reference
554   {
555     \tl_if_exist:cTF
556       {
557         g__tag_struct_#1_AF_tl
558       }
559       {
560         \tl_gput_right:cx
561           { g__tag_struct_#1_AF_tl }
562           {  \c_space_tl #2 }
563       }
564       {
565         \tl_new:c
566           { g__tag_struct_#1_AF_tl }
567         \tl_gset:cx
568           { g__tag_struct_#1_AF_tl }
569           { #2 }
570       }
571   }
572 \cs_generate_variant:Nn \__tag_struct_add_AF:nn {en,ee}
```

```
573  \keys_define:nn { __tag / struct }
574  {
575     AF .code:n        = % AF property
576        {
577           \pdf_object_if_exist:nTF {#1}
578              {
579                 \__tag_struct_add_AF:ee { \int_eval:n {\c@g__tag_struct_abs_int} }{\pdf_object_re
580                 \__tag_prop_gput:cnx
581                 { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
582                 { AF }
583                 {
584                    [
585                       \tl_use:c
586                          { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
587                    ]
588                 }
589              }
590              {

592              }
593        },
594     ,AFinline .code:n =
595        {
596           \__tag_struct_add_inline_AF:nn {#1}{txt}
597        }
598     ,AFinline-o .code:n =
599        {
600           \__tag_struct_add_inline_AF:on {#1}{txt}
601        }
602     ,texsource .code:n =
603        {
604           \group_begin:
605           \pdfdict_put:nnn { l_pdffile/Filespec }{AFRelationship} { /Source }
```

we set the mime type as pdfresources uses currently text/plain

```
606           \pdfdict_put:nnx
607              { l_pdffile }{Subtype}
608              { \pdf_name_from_unicode_e:n{application/x-tex} }
609           \__tag_struct_add_inline_AF:on {#1}{tex}
610           \group_end:
611        }
612  }
```

(*End definition for* AF (struct-key) *and others. These functions are documented on page 88.*)

root-AF␣(setup-key)   The root structure can take AF keys too, so we provide a key for it. This key is used with \tagpdfsetup, not in a structure!

```
613  \keys_define:nn { __tag / setup }
614  {
615     root-AF .code:n =
616        {
617           \pdf_object_if_exist:nTF {#1}
618              {
619                 \__tag_struct_add_AF:ee { 0 }{\pdf_object_ref:n {#1}}
```

```
620          \__tag_prop_gput:cnx
621            { g__tag_struct_0_prop }
622            { AF }
623            {
624              [
625                \tl_use:c
626                  { g__tag_struct_0_AF_tl }
627              ]
628            }
629          }
630          {

632          }
633        },
634      }
```
⟨/package⟩

(*End definition for* `root-AF` (setup-key). *This function is documented on page 89.*)

6   User commands

<div style="float:left">

`\tag_struct_begin:n`
`\tag_struct_end:`

</div>

```
636 ⟨base⟩\cs_new_protected:Npn \tag_struct_begin:n #1 {\int_gincr:N \c@g__tag_struct_abs_int}
637 ⟨base⟩\cs_new_protected:Npn \tag_struct_end:{}
638 ⟨base⟩\cs_new_protected:Npn \tag_struct_end:n{}
639 ⟨*package | debug⟩
640 ⟨package⟩\cs_set_protected:Npn \tag_struct_begin:n #1 %#1 key-val
641 ⟨debug⟩\cs_set_protected:Npn \tag_struct_begin:n #1 %#1 key-val
642    {
643 ⟨package⟩\__tag_check_if_active_struct:T
644 ⟨debug⟩\__tag_check_if_active_struct:TF
645      {
646        \group_begin:
647        \int_gincr:N \c@g__tag_struct_abs_int
648        \__tag_prop_new:c  { g__tag_struct_\int_eval:n { \c@g__tag_struct_abs_int }_prop }
649        \__tag_new_output_prop_handler:n {\int_eval:n { \c@g__tag_struct_abs_int }}
650        \__tag_seq_new:c  { g__tag_struct_kids_\int_eval:n { \c@g__tag_struct_abs_int }_seq}
651        \exp_args:Ne
652          \pdf_object_new:n
653            { __tag/struct/\int_eval:n { \c@g__tag_struct_abs_int } }
654        \__tag_prop_gput:cno
655          { g__tag_struct_\int_eval:n { \c@g__tag_struct_abs_int }_prop }
656          { Type }
657          { /StructElem }
658        \tl_set:Nn \l__tag_struct_stack_parent_tmpa_tl {-1}
659        \keys_set:nn { __tag / struct} { #1 }

660        \__tag_struct_set_tag_info:eVV
661          { \int_eval:n {\c@g__tag_struct_abs_int} }
662           \g__tag_struct_tag_tl
663           \g__tag_struct_tag_NS_tl
664        \__tag_check_structure_has_tag:n { \int_eval:n {\c@g__tag_struct_abs_int} }
665        \tl_if_empty:NF
666          \l__tag_struct_key_label_tl
```

```
667              {
668                \__tag_ref_label:en{tagpdfstruct-\l__tag_struct_key_label_tl}{struct}
669              }
```

The structure number of the parent is either taken from the stack or has been set with the parent key.

```
670          \int_compare:nNnT { \l__tag_struct_stack_parent_tmpa_tl } = { -1 }
671            {
672              \seq_get:NNF
673                \g__tag_struct_stack_seq
674                \l__tag_struct_stack_parent_tmpa_tl
675                {
676                  \msg_error:nn { tag } { struct-faulty-nesting }
677                }
678            }
679          \seq_gpush:NV \g__tag_struct_stack_seq          \c@g__tag_struct_abs_int
680          \__tag_role_get:VVN
681            \g__tag_struct_tag_tl
682            \g__tag_struct_tag_NS_tl
683            \l__tag_get_tmpc_tl
684          \seq_gpush:Nx \g__tag_struct_tag_stack_seq
685            {{\g__tag_struct_tag_tl}{\l__tag_get_tmpc_tl}}
686          \tl_gset:NV   \g__tag_struct_stack_current_tl \c@g__tag_struct_abs_int
687          %\seq_show:N   \g__tag_struct_stack_seq
688          \bool_if:NF
689            \l__tag_struct_elem_stash_bool
690            {
```

check if the tag can be used inside the parent. It only makes sense, if the structure is actually used here, so it is guarded by the stash boolean. For now we ignore the namespace!

```
691              \__tag_struct_get_tag_info:eNN
692                {\l__tag_struct_stack_parent_tmpa_tl}
693                \l__tag_get_parent_tmpa_tl
694                \l__tag_get_parent_tmpb_tl
695              \__tag_check_parent_child:VVVVN
696                \l__tag_get_parent_tmpa_tl
697                \l__tag_get_parent_tmpb_tl
698                \g__tag_struct_tag_tl
699                \g__tag_struct_tag_NS_tl
700                \l__tag_parent_child_check_tl
701            \int_compare:nNnT {\l__tag_parent_child_check_tl}<0
702              {
703                \msg_warning:nnxxx
704                  { tag }
705                  {role-parent-child}
706                  { \l__tag_get_parent_tmpa_tl/\l__tag_get_parent_tmpb_tl }
707                  { \g__tag_struct_tag_tl/\g__tag_struct_tag_NS_tl  }
708                  { not~allowed~(struct~\g__tag_struct_stack_current_tl) }
709                \cs_set_eq:NN \l__tag_role_remap_tag_tl \g__tag_struct_tag_tl
710                \cs_set_eq:NN \l__tag_role_remap_NS_tl \g__tag_struct_tag_NS_tl
711                \__tag_role_remap:
712                \cs_gset_eq:NN \g__tag_struct_tag_tl \l__tag_role_remap_tag_tl
713                \cs_gset_eq:NN  \g__tag_struct_tag_NS_tl \l__tag_role_remap_NS_tl
```

```
714                          \__tag_struct_set_tag_info:eVV
715                            { \int_eval:n {\c@g__tag_struct_abs_int} }
716                              \g__tag_struct_tag_tl
717                              \g__tag_struct_tag_NS_tl
718                        }
```
Set the Parent.
```
719                        \__tag_prop_gput:cnx
720                          { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
721                          { P }
722                          {
723                            \pdf_object_ref:e { __tag/struct/\l__tag_struct_stack_parent_tmpa_tl }
724                          }
725                        %record this structure as kid:
726                        %\tl_show:N \g__tag_struct_stack_current_tl
727                        %\tl_show:N \l__tag_struct_stack_parent_tmpa_tl
728                        \__tag_struct_kid_struct_gput_right:xx
729                          { \l__tag_struct_stack_parent_tmpa_tl }
730                          { \g__tag_struct_stack_current_tl }
731                        %\prop_show:c { g__tag_struct_\g__tag_struct_stack_current_tl _prop }
732                        %\seq_show:c {g__tag_struct_kids_\l__tag_struct_stack_parent_tmpa_tl _seq}
733                      }
734              %\prop_show:c { g__tag_struct_\g__tag_struct_stack_current_tl _prop }
735              %\seq_show:c {g__tag_struct_kids_\l__tag_struct_stack_parent_tmpa_tl _seq}
736  ⟨debug⟩ \__tag_debug_struct_begin_insert:n { #1 }
737              \group_end:
738          }
739  ⟨debug⟩{ \__tag_debug_struct_begin_ignore:n { #1 }}
740    }
741  ⟨package⟩\cs_set_protected:Nn \tag_struct_end:
742  ⟨debug⟩\cs_set_protected:Nn \tag_struct_end:
743    { %take the current structure num from the stack:
744      %the objects are written later, lua mode hasn't all needed info yet
745      %\seq_show:N \g__tag_struct_stack_seq
746  ⟨package⟩\__tag_check_if_active_struct:T
747  ⟨debug⟩\__tag_check_if_active_struct:TF
748        {
749          \seq_gpop:NN    \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
750          \seq_gpop:NNTF \g__tag_struct_stack_seq \l__tag_tmpa_tl
751            {
752              \__tag_check_info_closing_struct:o { \g__tag_struct_stack_current_tl }
753            }
754            { \__tag_check_no_open_struct: }
755          % get the previous one, shouldn't be empty as the root should be there
756          \seq_get:NNTF \g__tag_struct_stack_seq \l__tag_tmpa_tl
757            {
758              \tl_gset:NV    \g__tag_struct_stack_current_tl \l__tag_tmpa_tl
759            }
760            {
761              \__tag_check_no_open_struct:
762            }
763          \seq_get:NNT \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
764            {
765              \tl_gset:Nx \g__tag_struct_tag_tl
766                { \exp_last_unbraced:NV\use_i:nn \l__tag_tmpa_tl }
```

```
767                      }
768  ⟨debug⟩\__tag_debug_struct_end_insert:
769          }
770  ⟨debug⟩{\__tag_debug_struct_end_ignore:}
771    }
772
773  \cs_set_protected:Npn \tag_struct_end:n #1
774    {
775  ⟨debug⟩\__tag_debug_struct_end_check:n{#1}
776      \tag_struct_end:
777    }
778  ⟨/package | debug⟩
```

(*End definition for* \tag_struct_begin:n *and* \tag_struct_end:. *These functions are documented on page 86.*)

\tag_struct_use:n   This command allows to use a stashed structure in another place. TODO: decide how it should be guarded. Probably by the struct-check.

```
779  ⟨base⟩\cs_new_protected:Npn \tag_struct_use:n #1 {}
780  ⟨∗package⟩
781  \cs_set_protected:Npn \tag_struct_use:n #1 %#1 is the label
782    {
783      \__tag_check_if_active_struct:T
784        {
785          \prop_if_exist:cTF
786            { g__tag_struct_\__tag_ref_value:enn{tagpdfstruct-#1}{tagstruct}{unknown}_prop } %
787            {
788              \__tag_check_struct_used:n {#1}
789              %add the label structure as kid to the current structure (can be the root)
790              \__tag_struct_kid_struct_gput_right:xx
791                { \g__tag_struct_stack_current_tl }
792                { \__tag_ref_value:enn{tagpdfstruct-#1}{tagstruct}{0} }
793              %add the current structure to the labeled one as parents
794              \__tag_prop_gput:cnx
795                { g__tag_struct_\__tag_ref_value:enn{tagpdfstruct-#1}{tagstruct}{0}_prop }
796                { P }
797                {
798                  \pdf_object_ref:e { __tag/struct/\g__tag_struct_stack_current_tl }
799                }
```

check if the tag is allowed as child. Here we have to retrieve the tag info for the child, while the data for the parent is in the global tl-vars:

```
800                  \__tag_struct_get_tag_info:eNN
801                    {\__tag_ref_value:enn{tagpdfstruct-#1}{tagstruct}{0}}
802                    \l__tag_tmpa_tl
803                    \l__tag_tmpb_tl
804                  \__tag_check_parent_child:VVVVN
805                    \g__tag_struct_tag_tl
806                    \g__tag_struct_tag_NS_tl
807                    \l__tag_tmpa_tl
808                    \l__tag_tmpb_tl
809                    \l__tag_parent_child_check_tl
810                  \int_compare:nNnT {\l__tag_parent_child_check_tl}<0
811                    {
812                      \cs_set_eq:NN \l__tag_role_remap_tag_tl \g__tag_struct_tag_tl
```

```
813                    \cs_set_eq:NN \l__tag_role_remap_NS_tl \g__tag_struct_tag_NS_tl
814                    \__tag_role_remap:
815                    \cs_gset_eq:NN \g__tag_struct_tag_tl \l__tag_role_remap_tag_tl
816                    \cs_gset_eq:NN  \g__tag_struct_tag_NS_tl \l__tag_role_remap_NS_tl
817                    \__tag_struct_set_tag_info:eVV
818                      { \int_eval:n {\c@g__tag_struct_abs_int} }
819                        \g__tag_struct_tag_tl
820                        \g__tag_struct_tag_NS_tl
821                }
822            }
823            {
824                \msg_warning:nnn{ tag }{struct-label-unknown}{#1}
825            }
826        }
827    }
828 ⟨/package⟩
```

(*End definition for* \tag_struct_use:n. *This function is documented on page 86.*)

\tag_struct_object_ref:n   This is a command that allows to reference a structure. The argument is the number
                            which can be get for the current structure with \tag_get:n{struct_num} TODO check
                            if it should be in base too.

```
829 ⟨*package⟩
830 \cs_new:Npn \tag_struct_object_ref:n #1
831   {
832     \pdf_object_ref:n {__tag/struct/#1}
833   }
834 \cs_generate_variant:Nn \tag_struct_object_ref:n {e}
```

(*End definition for* \tag_struct_object_ref:n. *This function is documented on page 86.*)

\tag_struct_gput:nnn   This is a command that allows to update the data of a structure. The first argument is
                        the number of the structure, the second a keyword referring to a function, the third the
                        value. Currently the only keyword is ref

```
835 \cs_new_protected:Npn \tag_struct_gput:nnn #1 #2 #3
836   {
837     \cs_if_exist_use:cF {__tag_struct_gput_data_#2:nn}
838       { %warning??
839         \use_none:nn
840       }
841     {#1}{#3}
842   }
843 \cs_generate_variant:Nn \tag_struct_gput:nnn {ene,nne}
```

(*End definition for* \tag_struct_gput:nnn. *This function is documented on page* **??**.)

__tag_struct_gput_data_ref:nn

```
844 \cs_new_protected:Npn \__tag_struct_gput_data_ref:nn #1 #2
845   % #1 receiving struct num, #2 list of object ref
846   {
847     \prop_get:cnN
848       { g__tag_struct_#1_prop }
849       {Ref}
850       \l__tag_get_tmpc_tl
```

110

```
851        \__tag_prop_gput:cnx
852            { g__tag_struct_#1_prop }
853            { Ref }
854            { \quark_if_no_value:NF\l__tag_get_tmpc_tl { \l__tag_get_tmpc_tl\c_space_tl }#2 }
855        }
856  \cs_generate_variant:Nn \__tag_struct_gput_data_ref:nn {ee}
```

(*End definition for* `\__tag_struct_gput_data_ref:nn`.)

<div style="float:left">

`\tag_struct_insert_annot:nn`
`\tag_struct_insert_annot:xx`
`\tag_struct_parent_int:`

</div>

This are the user command to insert annotations. They must be used together to get the numbers right. They use a counter to the StructParent and `\tag_struct_insert_-annot:nn` increases the counter given back by `\tag_struct_parent_int:`.

It must be used together with `\tag_struct_parent_int:` to insert an annotation. TODO: decide how it should be guarded if tagging is deactivated.

```
857  \cs_new_protected:Npn \tag_struct_insert_annot:nn #1 #2 %#1 should be an object reference
858                                                          %#2 struct parent num
859    {
860      \__tag_check_if_active_struct:T
861        {
862          \__tag_struct_insert_annot:nn {#1}{#2}
863        }
864    }
865
866  \cs_generate_variant:Nn \tag_struct_insert_annot:nn {xx}
867  \cs_new:Npn \tag_struct_parent_int: {\int_use:c { c@g__tag_parenttree_obj_int }}
868
869  ⟨/package⟩
870
```

(*End definition for* `\tag_struct_insert_annot:nn` *and* `\tag_struct_parent_int:`. *These functions are documented on page 86.*)

7 Attributes and attribute classes

```
871  ⟨∗header⟩
872  \ProvidesExplPackage {tagpdf-attr-code} {2023-03-07} {0.98e}
873    {part of tagpdf - code related to attributes and attribute classes}
874  ⟨/header⟩
```

7.1 Variables

<div style="float:left">

`\g__tag_attr_entries_prop`
`\g__tag_attr_class_used_seq`
`\g__tag_attr_objref_prop`
`\l__tag_attr_value_tl`

</div>

`\g_@@_attr_entries_prop` will store attribute names and their dictionary content. `\g_@@_attr_class_used_seq` will hold the attributes which have been used as class name. `\l_@@_attr_value_tl` is used to build the attribute array or key. Everytime an attribute is used for the first time, and object is created with its content, the name-object reference relation is stored in `\g_@@_attr_objref_prop`

```
875  ⟨∗package⟩
876  \prop_new:N \g__tag_attr_entries_prop
877  \seq_new:N  \g__tag_attr_class_used_seq
878  \tl_new:N   \l__tag_attr_value_tl
879  \prop_new:N \g__tag_attr_objref_prop %will contain obj num of used attributes
```

(*End definition for* `\g__tag_attr_entries_prop` *and others.*)

7.2 Commands and keys

__tag_attr_new_entry:nn
newattribute␣(setup-key)

This allows to define attributes. Defined attributes are stored in a global property. newattribute expects two brace group, the name and the content. The content typically needs an /O key for the owner. An example look like this.

```
\tagpdfsetup
 {
  newattribute =
   {TH-col}{/O /Table /Scope /Column},
  newattribute =
   {TH-row}{/O /Table /Scope /Row},
 }
```

```
880 \cs_new_protected:Npn \__tag_attr_new_entry:nn #1 #2 %#1:name, #2: content
881   {
882     \prop_gput:Nen \g__tag_attr_entries_prop
883       {\pdf_name_from_unicode_e:n{#1}}{#2}
884   }
885
886 \keys_define:nn { __tag / setup }
887   {
888     newattribute .code:n =
889       {
890          \__tag_attr_new_entry:nn #1
891       }
892   }
```

(*End definition for* __tag_attr_new_entry:nn *and* newattribute (setup-key)*. This function is documented on page 89.*)

attribute-class␣(struct-key)

attribute-class has to store the used attribute names so that they can be added to the ClassMap later.

```
893 \keys_define:nn { __tag / struct }
894   {
895     attribute-class .code:n =
896      {
897        \clist_set:Nx \l__tag_tmpa_clist { #1 }
898        \seq_set_from_clist:NN \l__tag_tmpb_seq \l__tag_tmpa_clist
```

we convert the names into pdf names with slash

```
899        \seq_set_map_x:NNn \l__tag_tmpa_seq \l__tag_tmpb_seq
900          {
901            \pdf_name_from_unicode_e:n {##1}
902          }
903        \seq_map_inline:Nn \l__tag_tmpa_seq
904          {
905            \prop_if_in:NnF \g__tag_attr_entries_prop {##1}
906              {
907                \msg_error:nnn { tag } { attr-unknown } { ##1 }
908              }
909            \seq_gput_left:Nn\g__tag_attr_class_used_seq { ##1}
910          }
911        \tl_set:Nx \l__tag_tmpa_tl
912          {
```

```
913          \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[}
914          \seq_use:Nn \l__tag_tmpa_seq  { \c_space_tl  }
915          \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{]}
916        }
917      \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 0 }
918        {
919          \__tag_prop_gput:cnx
920            { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
921            { C }
922            { \l__tag_tmpa_tl }
923         %\prop_show:c  { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
924        }
925    }
926  }
```

(End definition for attribute-class (struct-key)*. This function is documented on page 88.)*

```
927 \keys_define:nn { __tag / struct }
928   {
929     attribute .code:n  = % A property (attribute, value currently a dictionary)
930       {
931         \clist_set:Nx            \l__tag_tmpa_clist { #1 }
932         \clist_if_empty:NF \l__tag_tmpa_clist
933           {
934             \seq_set_from_clist:NN \l__tag_tmpb_seq \l__tag_tmpa_clist
```
we convert the names into pdf names with slash
```
935             \seq_set_map_x:NNn \l__tag_tmpa_seq \l__tag_tmpb_seq
936               {
937                 \pdf_name_from_unicode_e:n {##1}
938               }
939             \tl_set:Nx \l__tag_attr_value_tl
940               {
941                 \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[}%]
942               }
943             \seq_map_inline:Nn \l__tag_tmpa_seq
944               {
945                 \prop_if_in:NnF \g__tag_attr_entries_prop {##1}
946                   {
947                     \msg_error:nnn { tag } { attr-unknown } { ##1 }
948                   }
949                 \prop_if_in:NnF \g__tag_attr_objref_prop {##1}
950                   {%\prop_show:N \g__tag_attr_entries_prop
951                     \pdf_object_unnamed_write:nx
952                       { dict }
953                       {
954                         \prop_item:Nn\g__tag_attr_entries_prop {##1}
955                       }
956                     \prop_gput:Nnx \g__tag_attr_objref_prop {##1} {\pdf_object_ref_last:}
957                   }
958                 \tl_put_right:Nx \l__tag_attr_value_tl
959                   {
960                     \c_space_tl
961                     \prop_item:Nn \g__tag_attr_objref_prop {##1}
```

```
962                           }
963     %      \tl_show:N \l__tag_attr_value_tl
964                        }
965              \tl_put_right:Nx \l__tag_attr_value_tl
966                { %[
967                  \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{]}%
968                }
969     %        \tl_show:N \l__tag_attr_value_tl
970              \__tag_prop_gput:cnx
971                { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
972                { A }
973                { \l__tag_attr_value_tl }
974         }
975     },
976   }
977 ⟨/package⟩
```

(End definition for `attribute (struct-key)`*. This function is documented on page 88.)*

Part VIII
The **tagpdf-luatex.def** Driver for luatex
Part of the tagpdf package

```
1 ⟨@@=tag⟩
2 ⟨∗luatex⟩
3 \ProvidesExplFile {tagpdf-luatex.def} {2023-03-07} {0.98e}
4   {tagpdf~driver~for~luatex}
```

1 Loading the lua

The space code requires that the fall back font has been loaded and initialized, so we force that first. But perhaps this could be done in the kernel.

```
5 {
6   \fontencoding{TU}\fontfamily{lmr}\fontseries{m}\fontshape{n}\fontsize{10pt}{10pt}\selectfont
7 }
8 \lua_now:e { tagpdf=require('tagpdf.lua') }
```

The following defines wrappers around prop and seq commands to store the data also in lua tables. I probably want also lua tables I put them in the ltx.@@.tables namespaces The tables will be named like the variables but without backslash To access such a table with a dynamical name create a string and then use ltx.@@.tables[string] Old code, I'm not quite sure if this was a good idea. Now I have mix of table in ltx.@@.tables and ltx.@@.mc/struct. And a lot is probably not needed. TODO: this should be cleaned up, but at least roles are currently using the table!

```
                           9 \cs_set_protected:Npn \__tag_prop_new:N #1
\__tag_prop_new:N         10   {
\__tag_seq_new:N          11     \prop_new:N #1
\__tag_prop_gput:Nnn      12     \lua_now:e { ltx.__tag.tables.\cs_to_str:N#1 = {} }
\__tag_seq_gput_right:Nn  13   }
\__tag_seq_item:cn        14
\__tag_prop_item:cn       15
\__tag_seq_show:N         16 \cs_set_protected:Npn \__tag_seq_new:N #1
\__tag_prop_show:N        17   {
                          18     \seq_new:N #1
                          19     \lua_now:e { ltx.__tag.tables.\cs_to_str:N#1 = {} }
                          20   }
                          21
                          22
                          23 \cs_set_protected:Npn \__tag_prop_gput:Nnn #1 #2 #3
                          24   {
                          25     \prop_gput:Nnn #1 { #2 } { #3 }
                          26     \lua_now:e { ltx.__tag.tables.\cs_to_str:N#1 ["#2"] = "#3" }
                          27   }
                          28
                          29
```

```
30 \cs_set_protected:Npn \__tag_seq_gput_right:Nn #1 #2
31   {
32     \seq_gput_right:Nn #1 { #2 }
33     \lua_now:e { table.insert(ltx.__tag.tables.\cs_to_str:N#1, "#2") }
34   }
35
36 %Hm not quite sure about the naming
37
38 \cs_set:Npn \__tag_seq_item:cn #1 #2
39   {
40     \lua_now:e { tex.print(ltx.__tag.tables.#1[#2]) }
41   }
42
43 \cs_set:Npn \__tag_prop_item:cn #1 #2
44   {
45     \lua_now:e { tex.print(ltx.__tag.tables.#1["#2"]) }
46   }
47
48 %for debugging commands that show both the seq/prop and the lua tables
49 \cs_set_protected:Npn \__tag_seq_show:N #1
50   {
51     \seq_show:N #1
52     \lua_now:e { ltx.__tag.trace.log ("lua~sequence~array~\cs_to_str:N#1",1) }
53     \lua_now:e { ltx.__tag.trace.show_seq (ltx.__tag.tables.\cs_to_str:N#1) }
54   }
55
56 \cs_set_protected:Npn \__tag_prop_show:N #1
57   {
58     \prop_show:N #1
59     \lua_now:e {ltx.__tag.trace.log  ("lua~property~table~\cs_to_str:N#1",1) }
60     \lua_now:e {ltx.__tag.trace.show_prop (ltx.__tag.tables.\cs_to_str:N#1) }
61   }
```

(*End definition for* `\__tag_prop_new:N` *and others.*)

```
62 ⟨/luatex⟩
```

The module declaration

```
63 ⟨*lua⟩
64 -- tagpdf.lua
65 -- Ulrike Fischer
66
67 local ProvidesLuaModule = {
68     name         = "tagpdf",
69     version      = "0.98e",       --TAGVERSION
70     date         = "2023-03-07", --TAGDATE
71     description  = "tagpdf lua code",
72     license      = "The LATEX Project Public License 1.3c"
73 }
74
75 if luatexbase and luatexbase.provides_module then
76   luatexbase.provides_module (ProvidesLuaModule)
77 end
78
79 --[[
```

```
80  The code has quite probably a number of problems
81   - more variables should be local instead of global
82   - the naming is not always consistent due to the development of the code
83   - the traversing of the shipout box must be tested with more complicated setups
84   - it should probably handle more node types
85   -
86  --]]
87
```

Some comments about the lua structure.

```
88  --[[
89  the main table is named ltx.__tag. It contains the functions and also the data
90  collected during the compilation.
91
92  ltx.__tag.mc      will contain mc connected data.
93  ltx.__tag.struct will contain structure related data.
94  ltx.__tag.page    will contain page data
95  ltx.__tag.tables contains also data from mc and struct (from older code). This needs cleaning
96              There are certainly dublettes, but I don't dare yet ...
97  ltx.__tag.func    will contain (public) functions.
98  ltx.__tag.trace  will contain tracing/loging functions.
99  local funktions starts with __
100 functions meant for users will be in ltx.tag
101
102 functions
103  ltx.__tag.func.get_num_from (tag):    takes a tag (string) and returns the id number
104  ltx.__tag.func.output_num_from (tag): takes a tag (string) and prints (to tex) the id number
105  ltx.__tag.func.get_tag_from (num):    takes a num and returns the tag
106  ltx.__tag.func.output_tag_from (num): takes a num and prints (to tex) the tag
107  ltx.__tag.func.store_mc_data (num,key,data): stores key=data in ltx.__tag.mc[num]
108  ltx.__tag.func.store_mc_label (label,num): stores label=num in ltx.__tag.mc.labels
109  ltx.__tag.func.store_mc_kid (mcnum,kid,page): stores the mc-kids of mcnum on page page
110  ltx.__tag.func.store_mc_in_page(mcnum,mcpagecnt,page): stores in the page table the number o
111  ltx.__tag.func.store_struct_mcabs (structnum,mcnum): stores relations structnum<->mcnum (abs)
112  ltx.__tag.func.mc_insert_kids (mcnum): inserts the /K entries for mcnum by wandering throught
113  ltx.__tag.func.mark_page_elements(box,mcpagecnt,mccntprev,mcopen,name,mctypeprev) : the main
114  ltx.__tag.func.mark_shipout (): a wrapper around the core function which inserts the last EMC
115  ltx.__tag.func.fill_parent_tree_line (page): outputs the entries of the parenttree for this p
116  ltx.__tag.func.output_parenttree(): outputs the content of the parenttree
117  ltx.__tag.func.pdf_object_ref(name): outputs the object reference for the object name
118  ltx.__tag.func.markspaceon(), ltx.__tag.func.markspaceoff(): (de)activates the marking of pos
119  ltx.__tag.trace.show_mc_data (num,loglevel): shows ltx.__tag.mc[num] is the current log level
120  ltx.__tag.trace.show_all_mc_data (max,loglevel): shows a maximum about mc's if the current lo
121  ltx.__tag.trace.show_seq: shows a sequence (array)
122  ltx.__tag.trace.show_struct_data (num): shows data of structure num
123  ltx.__tag.trace.show_prop: shows a prop
124  ltx.__tag.trace.log
125  ltx.__tag.trace.showspaces : boolean
126 --]]
127
```

This set-ups the main attribute registers. The mc_type attribute stores the type (P, Span etc) encoded as a num, The mc_cnt attribute stores the absolute number and allows so to see if a node belongs to the same mc-chunk.

The interwordspace attr is set by the function `@@_mark_spaces`, and marks the place where spaces should be inserted. The interwordfont attr is set by the function `@@_mark_spaces` too and stores the font, so that we can decide which font to use for the real space char.

```
128 local mctypeattributeid  = luatexbase.new_attribute ("g__tag_mc_type_attr")
129 local mccntattributeid   = luatexbase.new_attribute ("g__tag_mc_cnt_attr")
130 local iwspaceattributeid = luatexbase.new_attribute ("g__tag_interwordspace_attr")
131 local iwfontattributeid  = luatexbase.new_attribute ("g__tag_interwordfont_attr")
```

with this token we can query the state of the boolean and so detect if unmarked nodes should be marked as attributes

```
132 local tagunmarkedbool= token.create("g__tag_tagunmarked_bool")
133 local truebool        = token.create("c_true_bool")
```

Now a number of local versions from global tables. Not all is perhaps needed, most node variants were copied from lua-debug.

```
134 local catlatex        = luatexbase.registernumber("catcodetable@latex")
135 local tableinsert     = table.insert
136 local nodeid          = node.id
137 local nodecopy        = node.copy
138 local nodegetattribute = node.get_attribute
139 local nodesetattribute = node.set_attribute
140 local nodehasattribute = node.has_attribute
141 local nodenew         = node.new
142 local nodetail        = node.tail
143 local nodeslide       = node.slide
144 local noderemove      = node.remove
145 local nodetraverseid  = node.traverse_id
146 local nodetraverse    = node.traverse
147 local nodeinsertafter = node.insert_after
148 local nodeinsertbefore = node.insert_before
149 local pdfpageref      = pdf.pageref
150
151 local HLIST          = node.id("hlist")
152 local VLIST          = node.id("vlist")
153 local RULE           = node.id("rule")
154 local DISC           = node.id("disc")
155 local GLUE           = node.id("glue")
156 local GLYPH          = node.id("glyph")
157 local KERN           = node.id("kern")
158 local PENALTY        = node.id("penalty")
159 local LOCAL_PAR      = node.id("local_par")
160 local MATH           = node.id("math")
```

Now we setup the main table structure. ltx is used by other latex code too!

```
161 ltx              = ltx          or { }
162 ltx.__tag        = ltx.__tag        or { }
163 ltx.__tag.mc     = ltx.__tag.mc    or  { } -- mc data
164 ltx.__tag.struct = ltx.__tag.struct or { } -- struct data
165 ltx.__tag.tables = ltx.__tag.tables or { } -- tables created with new prop and new seq.
166                                    -- wasn't a so great idea ...
167                                    -- g__tag_role_tags_seq used by tag<-> is in this tabl
168                                    -- used for pure lua tables too now!
169 ltx.__tag.page   = ltx.__tag.page   or { } -- page data, currently only i->{0->mcnum,1->mc
170 ltx.__tag.trace  = ltx.__tag.trace  or { } -- show commands
```

118

```
171 ltx.__tag.func    = ltx.__tag.func   or  { } -- functions
172 ltx.__tag.conf    = ltx.__tag.conf   or  { } -- configuration variables
```

2 Logging functions

__tag_log
ltx.__tag.trace.log
This rather simple log function takes as argument a message (string) and a number and will output the message to the log/terminal if the current loglevel is greater or equal than num.

```
173 local __tag_log =
174 function (message,loglevel)
175   if (loglevel or 3) <= tex.count["l__tag_loglevel_int"] then
176     texio.write_nl("tagpdf: ".. message)
177   end
178 end
179
180 ltx.__tag.trace.log = __tag_log
```

(*End definition for* __tag_log *and* ltx.__tag.trace.log.)

ltx.__tag.trace.show_seq   This shows the content of a seq as stored in the tables table. It is used by the \@@_seq_show:N function. It is not used in user commands, only for debugging, and so requires log level >0.

```
181 function ltx.__tag.trace.show_seq (seq)
182   if (type(seq) == "table") then
183     for i,v in ipairs(seq) do
184       __tag_log ("[" .. i .. "] => " .. tostring(v),1)
185     end
186   else
187     __tag_log ("sequence " .. tostring(seq) .. " not found",1)
188   end
189 end
```

(*End definition for* ltx.__tag.trace.show_seq.)

__tag_pairs_prop
ltx.__tag.trace.show_prop
This shows the content of a prop as stored in the tables table. It is used by the \@@_prop_show:N function.

```
190 local __tag_pairs_prop =
191 function  (prop)
192     local a = {}
193     for n in pairs(prop) do tableinsert(a, n) end
194     table.sort(a)
195     local i = 0                  -- iterator variable
196     local iter = function ()   -- iterator function
197       i = i + 1
198       if a[i] == nil then return nil
199       else return a[i], prop[a[i]]
200       end
201     end
202     return iter
203   end
204
205
206 function ltx.__tag.trace.show_prop (prop)
```

119

```
207  if (type(prop) == "table") then
208    for i,v in __tag_pairs_prop (prop) do
209      __tag_log ("[" .. i .. "] => " .. tostring(v),1)
210    end
211  else
212    __tag_log ("prop " .. tostring(prop) .. " not found or not a table",1)
213  end
214  end
```

(*End definition for* `__tag_pairs_prop` *and* `ltx.__tag.trace.show_prop.`)

`ltx.__tag.trace.show_mc_data` This shows some data for a mc given by `num`. If something is shown depends on the log level. The function is used by the following function and then in `\ShowTagging`

```
215  function ltx.__tag.trace.show_mc_data (num,loglevel)
216  if ltx.__tag and ltx.__tag.mc and ltx.__tag.mc[num] then
217    for k,v in pairs(ltx.__tag.mc[num]) do
218      __tag_log  ("mc"..num..": "..tostring(k).."=>"..tostring(v),loglevel)
219    end
220    if ltx.__tag.mc[num]["kids"] then
221    __tag_log ("mc" .. num .. " has " .. #ltx.__tag.mc[num]["kids"] .. " kids",loglevel)
222      for k,v in ipairs(ltx.__tag.mc[num]["kids"]) do
223        __tag_log ("mc ".. num .. " kid "..k.." =>" .. v.kid.." on page " ..v.page,loglevel)
224      end
225    end
226  else
227    __tag_log  ("mc"..num.." not found",loglevel)
228  end
229  end
```

(*End definition for* `ltx.__tag.trace.show_mc_data.`)

`ltx.__tag.trace.show_all_mc_data` This shows data for the mc's between `min` and `max` (numbers). It is used by the `\ShowTagging` function.

```
230  function ltx.__tag.trace.show_all_mc_data (min,max,loglevel)
231  for i = min, max do
232    ltx.__tag.trace.show_mc_data (i,loglevel)
233  end
234  texio.write_nl("")
235  end
```

(*End definition for* `ltx.__tag.trace.show_all_mc_data.`)

`ltx.__tag.trace.show_struct_data` This function shows some struct data. Unused but kept for debugging.

```
236  function ltx.__tag.trace.show_struct_data (num)
237  if ltx.__tag and ltx.__tag.struct and ltx.__tag.struct[num] then
238    for k,v in ipairs(ltx.__tag.struct[num]) do
239      __tag_log  ("struct "..num..": "..tostring(k).."=>"..tostring(v),1)
240    end
241  else
242    __tag_log  ("struct "..num.." not found ",1)
243  end
244  end
```

(*End definition for* `ltx.__tag.trace.show_struct_data.`)

3 Helper functions

3.1 Retrieve data functions

__tag_get_mc_cnt_type_tag

This takes a node as argument and returns the mc-cnt, the mc-type and and the tag (calculated from the mc-cnt.

```
245 local __tag_get_mc_cnt_type_tag = function (n)
246   local mccnt       = nodegetattribute(n,mccntattributeid)  or -1
247   local mctype      = nodegetattribute(n,mctypeattributeid)  or -1
248   local tag         = ltx.__tag.func.get_tag_from(mctype)
249   return mccnt,mctype,tag
250 end
```

(*End definition for* `__tag_get_mc_cnt_type_tag`.)

__tag_get_mathsubtype

This function allows to detect if we are at the begin or the end of math. It takes as argument a mathnode.

```
251 local function __tag_get_mathsubtype  (mathnode)
252   if mathnode.subtype == 0 then
253     subtype = "beginmath"
254   else
255     subtype = "endmath"
256   end
257   return subtype
258 end
```

(*End definition for* `__tag_get_mathsubtype`.)

ltx.__tag.tables.role_tag_attribute

The first is a table with key a tag and value a number (the attribute) The second is an array with the attribute value as key.

```
259 ltx.__tag.tables.role_tag_attribute = {}
260 ltx.__tag.tables.role_attribute_tag = {}
```

(*End definition for* `ltx.__tag.tables.role_tag_attribute`.)

ltx.__tag.func.alloctag

```
261 local __tag_alloctag =
262   function (tag)
263     if not ltx.__tag.tables.role_tag_attribute[tag] then
264       table.insert(ltx.__tag.tables.role_attribute_tag,tag)
265       ltx.__tag.tables.role_tag_attribute[tag]=#ltx.__tag.tables.role_attribute_tag
266       __tag_log  ("Add "..tag.." "..ltx.__tag.tables.role_tag_attribute[tag],3)
267     end
268   end
269 ltx.__tag.func.alloctag = __tag_alloctag
```

(*End definition for* `ltx.__tag.func.alloctag`.)

__tag_get_num_from
ltx.__tag.func.get_num_from
ltx.__tag.func.output_num_from

These functions take as argument a string `tag`, and return the number under which is it recorded (and so the attribute value). The first function outputs the number for lua, while the `output` function outputs to tex.

```
270 local __tag_get_num_from =
271   function (tag)
272     if ltx.__tag.tables.role_tag_attribute[tag] then
```

121

```
273    a= ltx.__tag.tables.role_tag_attribute[tag]
274   else
275     a= -1
276   end
277   return a
278  end
279
280 ltx.__tag.func.get_num_from = __tag_get_num_from
281
282 function ltx.__tag.func.output_num_from (tag)
283   local num = __tag_get_num_from (tag)
284   tex.sprint(catlatex,num)
285   if num == -1 then
286    __tag_log ("Unknown tag "..tag.." used")
287   end
288 end
```

(*End definition for* __tag_get_num_from*,* ltx.__tag.func.get_num_from*, and* ltx.__tag.func.output_-num_from*.*)

__tag_get_tag_from
ltx.__tag.func.get_tag_from
ltx.__tag.func.output_tag_from

These functions are the opposites to the previous function: they take as argument a number (the attribute value) and return the string tag. The first function outputs the string for lua, while the output function outputs to tex.

```
289 local __tag_get_tag_from =
290  function  (num)
291   if ltx.__tag.tables.role_attribute_tag[num] then
292    a = ltx.__tag.tables.role_attribute_tag[num]
293   else
294    a= "UNKNOWN"
295   end
296  return a
297 end
298
299 ltx.__tag.func.get_tag_from = __tag_get_tag_from
300
301 function ltx.__tag.func.output_tag_from (num)
302   tex.sprint(catlatex,__tag_get_tag_from (num))
303 end
```

(*End definition for* __tag_get_tag_from*,* ltx.__tag.func.get_tag_from*, and* ltx.__tag.func.output_-tag_from*.*)

ltx.__tag.func.store_mc_data

This function stores for key=data for mc-chunk num. It is used in the tagpdf-mc code, to store for example the tag string, and the raw options.

```
304 function ltx.__tag.func.store_mc_data (num,key,data)
305  ltx.__tag.mc[num] = ltx.__tag.mc[num] or { }
306  ltx.__tag.mc[num][key] = data
307  __tag_log  ("INFO TEX-STORE-MC-DATA: "..num.." => "..tostring(key).." => "..tostring(data),3)
308 end
```

(*End definition for* ltx.__tag.func.store_mc_data*.*)

ltx.__tag.func.store_mc_label

This function stores the label=num relationship in the labels subtable. TODO: this is probably unused and can go.

```
309 function ltx.__tag.func.store_mc_label (label,num)
```

```
310  ltx.__tag.mc["labels"] = ltx.__tag.mc["labels"] or { }
311  ltx.__tag.mc.labels[label] = num
312  end
```

(*End definition for* `ltx.__tag.func.store_mc_label.`)

`ltx.__tag.func.store_mc_kid` This function is used in the traversing code. It stores a sub-chunk of a mc `mcnum` into the `kids` table.

```
313  function ltx.__tag.func.store_mc_kid (mcnum,kid,page)
314  ltx.__tag.trace.log("INFO TAG-STORE-MC-KID: "..mcnum.." => " .. kid.." on page " .. page,3)
315  ltx.__tag.mc[mcnum]["kids"] = ltx.__tag.mc[mcnum]["kids"] or { }
316  local kidtable = {kid=kid,page=page}
317  tableinsert(ltx.__tag.mc[mcnum]["kids"], kidtable )
318  end
```

(*End definition for* `ltx.__tag.func.store_mc_kid.`)

`ltx._tag.func.mc_num_of_kids` This function returns the number of kids a mc `mcnum` has. We need to account for the case that a mc can have no kids.

```
319  function ltx.__tag.func.mc_num_of_kids (mcnum)
320  local num = 0
321  if ltx.__tag.mc[mcnum] and ltx.__tag.mc[mcnum]["kids"] then
322    num = #ltx.__tag.mc[mcnum]["kids"]
323  end
324  ltx.__tag.trace.log ("INFO MC-KID-NUMBERS: " .. mcnum .. "has " .. num .. "KIDS",4)
325  return num
326  end
```

(*End definition for* `ltx.__tag.func.mc_num_of_kids.`)

3.2   Functions to insert the pdf literals

`__tag_insert_emc_node` This insert the emc node.

```
327  local function __tag_insert_emc_node (head,current)
328  local emcnode = nodenew("whatsit","pdf_literal")
329      emcnode.data = "EMC"
330      emcnode.mode=1
331      head = node.insert_before(head,current,emcnode)
332  return head
333  end
```

(*End definition for* `__tag_insert_emc_node.`)

`__tag_insert_bmc_node` This inserts a simple bmc node

```
334  local function __tag_insert_bmc_node (head,current,tag)
335  local bmcnode = nodenew("whatsit","pdf_literal")
336      bmcnode.data = "/"..tag.." BMC"
337      bmcnode.mode=1
338      head = node.insert_before(head,current,bmcnode)
339  return head
340  end
```

(*End definition for* `__tag_insert_bmc_node.`)

`__tag_insert_bdc_node` This inserts a bcd node with a fix dict. TODO: check if this is still used, now that we create properties.

```
341 local function __tag_insert_bdc_node (head,current,tag,dict)
342  local bdcnode = nodenew("whatsit","pdf_literal")
343       bdcnode.data = "/"..tag.."<<"..dict..">> BDC"
344       bdcnode.mode=1
345       head = node.insert_before(head,current,bdcnode)
346  return head
347 end
```

(*End definition for* `__tag_insert_bdc_node`.)

`__tag_pdf_object_ref`
`ltx.__tag.func.pdf_object_ref` This allows to reference a pdf object reserved with the l3pdf command by name. The return value is `n 0 R`, if the object doesn't exist, n is 0. TODO: is uses internal l3pdf commands, this should be properly supported by l3pdf

```
348 local function __tag_pdf_object_ref (name)
349    local tokenname = 'c__pdf_backend_object_'..name..'_int'
350    local object = token.create(tokenname).index..' 0 R'
351    return object
352 end
353 ltx.__tag.func.pdf_object_ref=__tag_pdf_object_ref
```

(*End definition for* `__tag_pdf_object_ref` *and* `ltx.__tag.func.pdf_object_ref`.)

4 Function for the real space chars

`__tag_show_spacemark` A debugging function, it is used to inserts red color markers in the places where space chars can go, it can have side effects so not always reliable, but ok.

```
354 local function __tag_show_spacemark (head,current,color,height)
355  local markcolor = color or "1 0 0"
356  local markheight = height or 10
357  local pdfstring = node.new("whatsit","pdf_literal")
358       pdfstring.data =
359       string.format("q "..markcolor.." RG "..markcolor.." rg 0.4 w 0 %g m 0 %g l S Q",-
   3,markheight)
360       head = node.insert_after(head,current,pdfstring)
361  return head
362 end
```

(*End definition for* `__tag_show_spacemark`.)

`__tag_fakespace`
`ltx.__tag.func.fakespace` This is used to define a lua version of \pdffakespace

```
363 local function __tag_fakespace()
364    tex.setattribute(iwspaceattributeid,1)
365    tex.setattribute(iwfontattributeid,font.current())
366 end
367 ltx.__tag.func.fakespace = __tag_fakespace
```

(*End definition for* `__tag_fakespace` *and* `ltx.__tag.func.fakespace`.)

`__tag_mark_spaces` a function to mark up places where real space chars should be inserted. It only sets attributes, these are then be used in a later traversing which inserts the actual spaces. When space handling is activated this function is inserted in some callbacks.

```
368 --[[ a function to mark up places where real space chars should be inserted
369     it only sets an attribute.
370 --]]
371
372 local function __tag_mark_spaces (head)
373   local inside_math = false
374   for n in nodetraverse(head) do
375     local id = n.id
376     if id == GLYPH then
377       local glyph = n
378       if glyph.next and (glyph.next.id == GLUE)
379         and not inside_math  and (glyph.next.width >0)
380       then
381         nodesetattribute(glyph.next,iwspaceattributeid,1)
382         nodesetattribute(glyph.next,iwfontattributeid,glyph.font)
383       -- for debugging
384        if ltx.__tag.trace.showspaces then
385         __tag_show_spacemark (head,glyph)
386        end
387      elseif glyph.next and (glyph.next.id==KERN) and not inside_math then
388       local kern = glyph.next
389       if kern.next and (kern.next.id== GLUE)  and (kern.next.width >0)
390       then
391        nodesetattribute(kern.next,iwspaceattributeid,1)
392        nodesetattribute(kern.next,iwfontattributeid,glyph.font)
393       end
394      end
395      --  look also back
396     if glyph.prev and (glyph.prev.id == GLUE)
397        and not inside_math
398        and (glyph.prev.width >0)
399        and not nodehasattribute(glyph.prev,iwspaceattributeid)
400      then
401        nodesetattribute(glyph.prev,iwspaceattributeid,1)
402        nodesetattribute(glyph.prev,iwfontattributeid,glyph.font)
403      -- for debugging
404       if ltx.__tag.trace.showspaces then
405        __tag_show_spacemark (head,glyph)
406       end
407      end
408     elseif id == PENALTY then
409       local glyph = n
410       -- ltx.__tag.trace.log ("PENALTY ".. n.subtype.."VALUE"..n.penalty,3)
411       if glyph.next and (glyph.next.id == GLUE)
412         and not inside_math  and (glyph.next.width >0) and n.subtype==0
413       then
414         nodesetattribute(glyph.next,iwspaceattributeid,1)
415       --  nodesetattribute(glyph.next,iwfontattributeid,glyph.font)
416       -- for debugging
417        if ltx.__tag.trace.showspaces then
418         __tag_show_spacemark (head,glyph)
```

```
419        end
420      end
421    elseif id == MATH then
422      inside_math = (n.subtype == 0)
423    end
424  end
425  return head
426 end
```

(*End definition for* `__tag_mark_spaces`.)

Theses functions add/remove the function which marks the spaces to the callbacks `pre_linebreak_filter` and `hpack_filter`

```
427 local function __tag_activate_mark_space ()
428  if not luatexbase.in_callback ("pre_linebreak_filter","markspaces") then
429    luatexbase.add_to_callback("pre_linebreak_filter",__tag_mark_spaces,"markspaces")
430    luatexbase.add_to_callback("hpack_filter",__tag_mark_spaces,"markspaces")
431  end
432 end
433
434 ltx.__tag.func.markspaceon=__tag_activate_mark_space
435
436 local function __tag_deactivate_mark_space ()
437  if luatexbase.in_callback ("pre_linebreak_filter","markspaces") then
438    luatexbase.remove_from_callback("pre_linebreak_filter","markspaces")
439    luatexbase.remove_from_callback("hpack_filter","markspaces")
440  end
441 end
442
443 ltx.__tag.func.markspaceoff=__tag_deactivate_mark_space
```

(*End definition for* `__tag_activate_mark_space`, `ltx.__tag.func.markspaceon`, *and* `ltx.__tag.func.markspaceoff`.)
    We need two local variable to setup a default space char.

```
444 local default_space_char = node.new(GLYPH)
445 local default_fontid     = font.id("TU/lmr/m/n/10")
446 default_space_char.char  = 32
447 default_space_char.font  = default_fontid
```

These is the main function to insert real space chars. It inserts a glyph before every glue which has been marked previously. The attributes are copied from the glue, so if the tagging is done later, it will be tagged like it.

```
448 local function __tag_space_chars_shipout (box)
449  local head = box.head
450   if head then
451    for n in node.traverse(head) do
452      local spaceattr = nodegetattribute(n,iwspaceattributeid)  or -1
453      if n.id == HLIST  then -- enter the hlist
454        __tag_space_chars_shipout (n)
455      elseif n.id == VLIST then -- enter the vlist
456        __tag_space_chars_shipout (n)
457      elseif n.id == GLUE then
458        if ltx.__tag.trace.showspaces and spaceattr==1  then
459          __tag_show_spacemark (head,n,"0 1 0")
460        end
```

126

```
461      if spaceattr==1  then
462        local space
463        local space_char = node.copy(default_space_char)
464        local curfont    = nodegetattribute(n,iwfontattributeid)
465        ltx.__tag.trace.log ("INFO SPACE-FUNCTION-FONT: ".. tostring(curfont),3)
466        if curfont and luaotfload.aux.slot_of_name(curfont,"space") then
467          space_char.font=curfont
468        end
469        head, space = node.insert_before(head, n, space_char) --
470        n.width     = n.width - space.width
471        space.attr  = n.attr
472      end
473    end
474  end
475  box.head = head
476  end
477 end
478
479 function ltx.__tag.func.space_chars_shipout (box)
480   __tag_space_chars_shipout (box)
481 end
```

(*End definition for* `__tag_space_chars_shipout` *and* `ltx.__tag.func.space_chars_shipout`.)

5 Function for the tagging

ltx.__tag.func.mc_insert_kids This is the main function to insert the K entry into a StructElem object. It is used in tagpdf-mc-luacode module. The `single` attribute allows to handle the case that a single mc on the tex side can have more than one kid after the processing here, and so we get the correct array/non array setup.

```
482 function ltx.__tag.func.mc_insert_kids (mcnum,single)
483   if ltx.__tag.mc[mcnum] then
484   ltx.__tag.trace.log("INFO TEX-MC-INSERT-KID-TEST: " .. mcnum,4)
485    if ltx.__tag.mc[mcnum]["kids"] then
486     if #ltx.__tag.mc[mcnum]["kids"] > 1 and single==1 then
487      tex.sprint("[")
488     end
489     for i,kidstable in ipairs( ltx.__tag.mc[mcnum]["kids"] ) do
490      local kidnum  = kidstable["kid"]
491      local kidpage = kidstable["page"]
492      local kidpageobjnum = pdfpageref(kidpage)
493      ltx.__tag.trace.log("INFO TEX-MC-INSERT-KID: " .. mcnum ..
494                     " insert KID " ..i..
495                     " with num " .. kidnum ..
496                     " on page " .. kidpage.."/"..kidpageobjnum,3)
497     tex.sprint(catlatex,"<</Type /MCR /Pg "..kidpageobjnum .. " 0 R /MCID "..kidnum.. ">> " )
498     end
499     if #ltx.__tag.mc[mcnum]["kids"] > 1 and single==1 then
500      tex.sprint("]")
501     end
502    else
503     -- this is typically not a problem, e.g. empty hbox in footer/header can
504     -- trigger this warning.
```

```
505   ltx.__tag.trace.log("WARN TEX-MC-INSERT-NO-KIDS: "..mcnum.." has no kids",2)
506   if single==1 then
507     tex.sprint("null")
508   end
509  end
510  else
511  ltx.__tag.trace.log("WARN TEX-MC-INSERT-MISSING: "..mcnum.." doesn't exist",0)
512  end
513 end
```

(*End definition for* `ltx.__tag.func.mc_insert_kids`.)

`ltx.__tag.func.store_struct_mcabs` This function is used in the tagpdf-mc-luacode. It store the absolute count of the mc into the current structure. This must be done ordered.

```
514 function ltx.__tag.func.store_struct_mcabs (structnum,mcnum)
515  ltx.__tag.struct[structnum]=ltx.__tag.struct[structnum] or { }
516  ltx.__tag.struct[structnum]["mc"]=ltx.__tag.struct[structnum]["mc"] or { }
517  -- a structure can contain more than on mc chunk, the content should be ordered
518  tableinsert(ltx.__tag.struct[structnum]["mc"],mcnum)
519  ltx.__tag.trace.log("INFO TEX-MC-INTO-STRUCT: "..
520                      mcnum.." inserted in struct "..structnum,3)
521  -- but every mc can only be in one structure
522  ltx.__tag.mc[mcnum]= ltx.__tag.mc[mcnum] or { }
523  ltx.__tag.mc[mcnum]["parent"] = structnum
524 end
525
```

(*End definition for* `ltx.__tag.func.store_struct_mcabs`.)

`ltx.__tag.func.store_mc_in_page` This is used in the traversing code and stores the relation between abs count and page count.

```
526 -- pay attention: lua counts arrays from 1, tex pages from one
527 -- mcid and arrays in pdf count from 0.
528 function ltx.__tag.func.store_mc_in_page (mcnum,mcpagecnt,page)
529  ltx.__tag.page[page] = ltx.__tag.page[page] or {}
530  ltx.__tag.page[page][mcpagecnt] = mcnum
531  ltx.__tag.trace.log("INFO TAG-MC-INTO-PAGE: page " .. page ..
532                      ": inserting MCID " .. mcpagecnt .. " => " .. mcnum,3)
533 end
```

(*End definition for* `ltx.__tag.func.store_mc_in_page`.)

`ltx.__tag.func.mark_page_elements` This is the main traversing function. See the lua comment for more details.

```
534 --[[
535     Now follows the core function
536     It wades through the shipout box and checks the attributes
537     ARGUMENTS
538     box: is a box,
539     mcpagecnt: num, the current page cnt of mc (should start at -1 in shipout box), needed fo
540     mccntprev: num, the attribute cnt of the previous node/whatever - if different we have a
541     mcopen: num, records if some bdc/emc is open
542     These arguments are only needed for log messages, if not present are replaces by fix stri
543     name: string to describe the box
544     mctypeprev: num, the type attribute of the previous node/whatever
545
```

```
546    there are lots of logging messages currently. Should be cleaned up in due course.
547    One should also find ways to make the function shorter.
548  --]]
549
550  function ltx.__tag.func.mark_page_elements (box,mcpagecnt,mccntprev,mcopen,name,mctypeprev)
551    local name = name or ("SOMEBOX")
552    local mctypeprev = mctypeprev or -1
553    local abspage = status.total_pages + 1  -- the real counter is increased
554                                            -- inside the box so one off
555                                            -- if the callback is not used. (???)
556    ltx.__tag.trace.log ("INFO TAG-ABSPAGE: " .. abspage,3)
557    ltx.__tag.trace.log ("INFO TAG-ARGS: pagecnt".. mcpagecnt..
558                     " prev "..mccntprev ..
559                     " type prev "..mctypeprev,4)
560    ltx.__tag.trace.log ("INFO TAG-TRAVERSING-BOX: ".. tostring(name)..
561                     " TYPE ".. node.type(node.getid(box)),3)
562    local head = box.head -- ShipoutBox is a vlist?
563    if head then
564      mccnthead, mctypehead,taghead = __tag_get_mc_cnt_type_tag (head)
565      ltx.__tag.trace.log ("INFO TAG-HEAD: " ..
566                       node.type(node.getid(head))..
567                       " MC"..tostring(mccnthead)..
568                       " => TAG " .. tostring(mctypehead)..
569                       " => ".. tostring(taghead),3)
570    else
571      ltx.__tag.trace.log ("INFO TAG-NO-HEAD: head is "..
572                        tostring(head),3)
573    end
574    for n in node.traverse(head) do
575      local mccnt, mctype, tag = __tag_get_mc_cnt_type_tag (n)
576      local spaceattr = nodegetattribute(n,iwspaceattributeid)  or -1
577      ltx.__tag.trace.log ("INFO TAG-NODE: "..
578                       node.type(node.getid(n))..
579                       " MC".. tostring(mccnt)..
580                       " => TAG ".. tostring(mctype)..
581                       " => " ..  tostring(tag),3)
582      if n.id == HLIST
583      then -- enter the hlist
584       mcopen,mcpagecnt,mccntprev,mctypeprev=
585        ltx.__tag.func.mark_page_elements (n,mcpagecnt,mccntprev,mcopen,"INTERNAL HLIST",mctypep
586      elseif n.id == VLIST then -- enter the vlist
587       mcopen,mcpagecnt,mccntprev,mctypeprev=
588        ltx.__tag.func.mark_page_elements (n,mcpagecnt,mccntprev,mcopen,"INTERNAL VLIST",mctypep
589      elseif n.id == GLUE and not n.leader then -- at glue real space chars are inserted, but th
590                                     -- been done if the previous shipout wandering, so here it
591      elseif n.id == LOCAL_PAR then  -- local_par is ignored
592      elseif n.id == PENALTY then    -- penalty is ignored
593      elseif n.id == KERN then       -- kern is ignored
594       ltx.__tag.trace.log ("INFO TAG-KERN-SUBTYPE: "..
595         node.type(node.getid(n)).." "..n.subtype,4)
596      else
597       -- math is currently only logged.
598       -- we could mark the whole as math
599       -- for inner processing the mlist_to_hlist callback is probably needed.
```

129

```
600    if n.id == MATH then
601     ltx.__tag.trace.log("INFO TAG-MATH-SUBTYPE: "..
602       node.type(node.getid(n)).." "..__tag_get_mathsubtype(n),4)
603    end
604    -- endmath
605    ltx.__tag.trace.log("INFO TAG-MC-COMPARE: current "..
606           mccnt.." prev "..mccntprev,4)
607    if mccnt~=mccntprev then -- a new mc chunk
608     ltx.__tag.trace.log ("INFO TAG-NEW-MC-NODE: "..
609                      node.type(node.getid(n))..
610                      " MC"..tostring(mccnt)..
611                      " <=> PREVIOUS "..tostring(mccntprev),4)
612     if mcopen~=0 then -- there is a chunk open, close it (hope there is only one ...
613      box.list=__tag_insert_emc_node (box.list,n)
614      mcopen = mcopen - 1
615      ltx.__tag.trace.log ("INFO TAG-INSERT-EMC: " ..
616        mcpagecnt .. " MCOPEN = " .. mcopen,3)
617      if mcopen ~=0 then
618       ltx.__tag.trace.log ("WARN TAG-OPEN-MC: " .. mcopen,1)
619      end
620     end
621     if ltx.__tag.mc[mccnt] then
622      if ltx.__tag.mc[mccnt]["artifact"] then
623       ltx.__tag.trace.log("INFO TAG-INSERT-ARTIFACT: "..
624                      tostring(ltx.__tag.mc[mccnt]["artifact"]),3)
625       if ltx.__tag.mc[mccnt]["artifact"] == "" then
626        box.list = __tag_insert_bmc_node (box.list,n,"Artifact")
627       else
628        box.list = __tag_insert_bdc_node (box.list,n,"Artifact", "/Type /"..ltx.__tag.mc[mccn
629       end
630      else
631       ltx.__tag.trace.log("INFO TAG-INSERT-TAG: "..
632                      tostring(tag),3)
633      mcpagecnt = mcpagecnt +1
634      ltx.__tag.trace.log ("INFO TAG-INSERT-BDC: "..mcpagecnt,3)
635      local dict= "/MCID "..mcpagecnt
636      if ltx.__tag.mc[mccnt]["raw"] then
637       ltx.__tag.trace.log("INFO TAG-USE-RAW: "..
638         tostring(ltx.__tag.mc[mccnt]["raw"]),3)
639       dict= dict .. " " .. ltx.__tag.mc[mccnt]["raw"]
640      end
641      if ltx.__tag.mc[mccnt]["alt"] then
642       ltx.__tag.trace.log("INFO TAG-USE-ALT: "..
643          tostring(ltx.__tag.mc[mccnt]["alt"]),3)
644       dict= dict .. " " .. ltx.__tag.mc[mccnt]["alt"]
645      end
646      if ltx.__tag.mc[mccnt]["actualtext"] then
647       ltx.__tag.trace.log("INFO TAG-USE-ACTUALTEXT: "..
648         tostring(ltx.__tag.mc[mccnt]["actualtext"]),3)
649       dict= dict .. " " .. ltx.__tag.mc[mccnt]["actualtext"]
650      end
651      box.list = __tag_insert_bdc_node (box.list,n,tag, dict)
652      ltx.__tag.func.store_mc_kid (mccnt,mcpagecnt,abspage)
653      ltx.__tag.func.store_mc_in_page(mccnt,mcpagecnt,abspage)
```

```
654        ltx.__tag.trace.show_mc_data (mccnt,3)
655      end
656    mcopen = mcopen + 1
657   else
658    if tagunmarkedbool.mode == truebool.mode then
659     ltx.__tag.trace.log("INFO TAG-NOT-TAGGED: this has not been tagged, using artifact",2)
660     box.list = __tag_insert_bmc_node (box.list,n,"Artifact")
661     mcopen = mcopen + 1
662    else
663     ltx.__tag.trace.log("WARN TAG-NOT-TAGGED: this has not been tagged",1)
664    end
665   end
666   mccntprev = mccnt
667  end
668  end -- end if
669  end -- end for
670  if head then
671   mccnthead, mctypehead,taghead = __tag_get_mc_cnt_type_tag (head)
672   ltx.__tag.trace.log ("INFO TAG-ENDHEAD: " ..
673                         node.type(node.getid(head))..
674                       " MC"..tostring(mccnthead)..
675                       " => TAG "..tostring(mctypehead)..
676                       " => "..tostring(taghead),4)
677  else
678    ltx.__tag.trace.log ("INFO TAG-ENDHEAD: ".. tostring(head),4)
679  end
680  ltx.__tag.trace.log ("INFO TAG-QUITTING-BOX "..
681                       tostring(name)..
682                      " TYPE ".. node.type(node.getid(box)),4)
683  return mcopen,mcpagecnt,mccntprev,mctypeprev
684 end
685
```

(*End definition for* `ltx.__tag.func.mark_page_elements`.)

`ltx.__tag.func.mark_shipout` This is the function used in the callback. Beside calling the traversing function it also checks if there is an open MC-chunk from a page break and insert the needed EMC literal.

```
686 function ltx.__tag.func.mark_shipout (box)
687  mcopen = ltx.__tag.func.mark_page_elements (box,-1,-100,0,"Shipout",-1)
688  if mcopen~=0 then -- there is a chunk open, close it (hope there is only one ...
689   local emcnode = nodenew("whatsit","pdf_literal")
690   local list = box.list
691   emcnode.data = "EMC"
692   emcnode.mode=1
693   if list then
694      list = node.insert_after (list,node.tail(list),emcnode)
695      mcopen = mcopen - 1
696      ltx.__tag.trace.log ("INFO SHIPOUT-INSERT-LAST-EMC: MCOPEN " .. mcopen,3)
697   else
698      ltx.__tag.trace.log ("WARN SHIPOUT-UPS: this shouldn't happen",0)
699   end
700   if mcopen ~=0 then
701      ltx.__tag.trace.log ("WARN SHIPOUT-MC-OPEN: " .. mcopen,1)
```

131

```
702     end
703   end
704 end
```

(*End definition for* `ltx.__tag.func.mark_shipout`.)

6   Parenttree

These functions create the parent tree. The second, main function is used in the tagpdf-tree code. TODO check if the tree code can move into the backend code.

```
705 function ltx.__tag.func.fill_parent_tree_line (page)
706     -- we need to get page-> i=kid -> mcnum -> structnum
707     -- pay attention: the kid numbers and the page number in the parent tree start with 0!
708     local numsentry =""
709     local pdfpage = page-1
710     if ltx.__tag.page[page] and ltx.__tag.page[page][0] then
711     mcchunks=#ltx.__tag.page[page]
712     ltx.__tag.trace.log("INFO PARENTTREE-NUM:  page "..
713                     page.." has "..mcchunks.."+1 Elements ",4)
714     for i=0,mcchunks do
715     -- what does this log??
716      ltx.__tag.trace.log("INFO PARENTTREE-CHUNKS:  "..
717        ltx.__tag.page[page][i],4)
718     end
719     if mcchunks == 0 then
720      -- only one chunk so no need for an array
721      local mcnum  = ltx.__tag.page[page][0]
722      local structnum = ltx.__tag.mc[mcnum]["parent"]
723      local propname  = "g__tag_struct_"..structnum.."_prop"
724      --local objref   = ltx.__tag.tables[propname]["objref"] or "XXXX"
725      local objref = __tag_pdf_object_ref('__tag/struct/'..structnum)
726      ltx.__tag.trace.log("INFO PARENTTREE-STRUCT-OBJREF:  =====>"..
727        tostring(objref),5)
728      numsentry = pdfpage .. " ["..  objref .. "]"
729      ltx.__tag.trace.log("INFO PARENTTREE-NUMENTRY: page " ..
730        page.. " num entry = ".. numsentry,3)
731     else
732      numsentry = pdfpage .. " ["
733       for i=0,mcchunks do
734        local mcnum  = ltx.__tag.page[page][i]
735        local structnum = ltx.__tag.mc[mcnum]["parent"] or 0
736        local propname  = "g__tag_struct_"..structnum.."_prop"
737        --local objref   = ltx.__tag.tables[propname]["objref"] or "XXXX"
738        local objref = __tag_pdf_object_ref('__tag/struct/'..structnum)
739        numsentry = numsentry .. " ".. objref
740       end
741      numsentry = numsentry .. "] "
742      ltx.__tag.trace.log("INFO PARENTTREE-NUMENTRY: page " ..
743        page.. " num entry = ".. numsentry,3)
744     end
745     else
746       ltx.__tag.trace.log ("INFO PARENTTREE-NO-DATA: page "..page,3)
747     end
```

132

```
748       return numsentry
749  end
750
751  function ltx.__tag.func.output_parenttree (abspage)
752   for i=1,abspage do
753    line = ltx.__tag.func.fill_parent_tree_line (i) .. "^^J"
754    tex.sprint(catlatex,line)
755   end
756  end
```

(*End definition for* `ltx.__tag.func.fill_parent_tree_line` *and* `ltx.__tag.func.output_parenttree`*.*)

757 ⟨/lua⟩

Part IX

The **tagpdf-roles** module
Tags, roles and namesspace code
Part of the tagpdf package

```
add-new-tag␣(setup-key)
tag␣(rolemap-key)
namespace␣(rolemap-key)
role␣(rolemap-key)
role-namespace␣(rolemap-key)
```

The `add-new-tag` key can be used in `\tagpdfsetup` to declare and rolemap new tags. It takes as value a key-value list or a simple `new-tag/old-tag`.

The key-value list knows the following keys:

tag This is the name of the new tag as it should then be used in `\tagstructbegin`.

namespace This is the namespace of the new tag. The value should be a shorthand of a namespace. The allowed values are currently `pdf`, `pdf2`, `mathml`,`latex`, `latex-book` and `user`. The default value (and recommended value for a new tag) is `user`. The public name of the user namespace is `tag/NS/user`. This can be used to reference the namespace e.g. in attributes.

role This is the tag the tag should be mapped too. In a PDF 1.7 or earlier this is normally a tag from the `pdf` set, in PDF 2.0 from the `pdf`, `pdf2` and `mathml` set. It can also be a user tag. The tag must be declared before, as the code retrieves the class of the new tag from it. The PDF format allows mapping to be done transitively. But tagpdf can't/won't check such unusual role mapping.

role-namespace If the role is a known tag the default value is the default namespace of this tag. With this key a specific namespace can be forced.

Namespaces are mostly a PDF 2.0 property, but it doesn't harm to set them also in a PDF 1.7 or earlier.

`\tag_check_child:nn`*TF* `\tag_check_child:nn{`⟨*tag*⟩`}{`⟨*namespace*⟩`} {`⟨*true code*⟩`} {`⟨*false code*⟩`}`

This checks if the tag ⟨*tag*⟩ from the name space ⟨*namespace*⟩ can be used at the current position. In tagpdf-base it is always true.

```
1  ⟨@@=tag⟩
2  ⟨*header⟩
3  \ProvidesExplPackage {tagpdf-roles-code} {2023-03-07} {0.98e}
4   {part of tagpdf - code related to roles and structure names}
5  ⟨/header⟩
```

1   Code related to roles and structure names

1.1 Variables

Tags are used in structures (`\tagstructbegin`) and mc-chunks (`\tagmcbegin`).

They have a name (a string), in lua a number (for the lua attribute), and in PDF 2.0 belong to one or more name spaces, with one being the default name space.

Tags of structures are classified, e.g. as grouping, inline or block level structure (and a few special classes like lists and tables), and must follow containments rules depending on their classification (for example a inline structure can not contain a block level structure). New tags inherit their classification from their rolemapping to the standard namespaces (`pdf` and/or `pdf2`). We store this classification as it will probably be needed for tests but currently the data is not much used. The classification for math (and the containment rules) is unclear currently and so not set.

The attribute number is only relevant in lua and only for the MC chunks (so tags with the same name from different names spaces can have the same number), and so only stored if luatex is detected.

Due to the namespaces the storing and processing of tags and there data are different in various places for PDF 2.0 and PDF <2.0, which makes things a bit difficult and leads to some duplications. Perhaps at some time there should be a clear split.

This are the main variables used by the code:

`\g__tag_role_tags_NS_prop` This is the core list of tag names. It uses tags as keys and the shorthand (e.g. pdf2, or mathml) of the default name space as value.

   In pdf 2.0 the value is needed in the structure dictionaries.

`\g__tag_role_tags_class_prop` This contains for each tag a classification type. It is used in pdf <2.0.

`\g__tag_role_NS_prop` This contains the names spaces. The values are the object references. They are used in pdf 2.0.

`\g__tag_role_rolemap_prop` This contains for each tag the role to a standard tag. It is used in pdf<2.0 for tag checking and to fill at the end the RoleMap dictionary.

`g_@@_role/RoleMap_dict` This dictionary contains the standard rolemaps. It is relevant only for pdf <2.0.

`\g__tag_role_NS_<ns>_prop` This prop contains the tags of a name space and their role. The props are also use for remapping. As value they contain two brace groups: tag and namespace. In pdf <2.0 the namespace is empty.

`\g__tag_role_NS_<ns>_class_prop` This prop contains the tags of a name space and their type. The value is only needed for pdf 2.0.

`\g__tag_role_index_prop` This prop contains the standard tags (pdf in pdf<2.0, pdf,pdf2 + mathml in pdf 2.0) as keys, the values are a two-digit number. These numbers are used to get the containment rule of two tags from the intarray.

`\l__tag_role_debug_prop` This property is used to pass some info around for info messages or debugging.

`\g__tag_role_tags_NS_prop` This is the core list of tag names. It uses tags as keys and the shorthand (e.g. pdf2, or mathml) of the default name space as value. We store the default name space also in pdf <2.0, even if not needed: it doesn't harm and simplifies the code. There is no need to access this from lua, so we use the standard prop commands.

```
7 \prop_new:N    \g__tag_role_tags_NS_prop
```

(*End definition for* `\g__tag_role_tags_NS_prop`.)

`\g__tag_role_tags_class_prop` With pdf 2.0 we store the class in the NS dependant props. With pdf <2.0 we store for now the type(s) of a tag in a common prop. Tags that are rolemapped should get the type from the target.

```
8 \prop_new:N    \g__tag_role_tags_class_prop
```

(*End definition for* `\g__tag_role_tags_class_prop`.)

`\g__tag_role_NS_prop` This holds the list of supported name spaces. The keys are the name tagpdf will use, the values the object reference. The urls identifier are stored in related dict object.

mathml http://www.w3.org/1998/Math/MathML

pdf2 http://iso.org/pdf2/ssn

pdf http://iso.org/pdf/ssn (default)

user `\c__tag_role_userNS_id_str` (random id, for user tags)

latex https://www.latex-project.org/ns/dflt/2022

latex-book https://www.latex-project.org/ns/book/2022

latex-inline https://www.latex-project.org/ns/inline/2022

More namespaces are possible and their objects references and their rolemaps must be collected so that an array can be written to the StructTreeRoot at the end (see tagpdf-tree). We use a prop to store the object reference as it will be needed rather often.

```
9 \prop_new:N \g__tag_role_NS_prop
```

(*End definition for* `\g__tag_role_NS_prop`.)

`\g__tag_role_index_prop` This prop contains the standard tags (pdf in pdf<2.0, pdf,pdf2 + mathml in pdf 2.0) as keys, the values are a two-digit number. These numbers are used to get the containment rule of two tags from the intarray.

```
10 \prop_new:N \g__tag_role_index_prop
```

(*End definition for* `\g__tag_role_index_prop`.)

`\l__tag_role_debug_prop` This variable is used to pass more infos to debug messages.

```
11 \prop_new:N \l__tag_role_debug_prop
```

(*End definition for* `\l__tag_role_debug_prop`.)

We need also a bunch of temporary variables.

`\l__tag_role_tag_tmpa_tl`
`\l__tag_role_tag_namespace_tmpa_tl`
`\l__tag_role_role_tmpa_tl`
`\l__tag_role_role_namespace_tmpa_tl`
`\l__tag_role_tmpa_seq`

```
12 \tl_new:N \l__tag_role_tag_tmpa_tl
13 \tl_new:N \l__tag_role_tag_namespace_tmpa_tl
14 \tl_new:N \l__tag_role_role_tmpa_tl
15 \tl_new:N \l__tag_role_role_namespace_tmpa_tl
16 \seq_new:N\l__tag_role_tmpa_seq
```

(*End definition for* `\l__tag_role_tag_tmpa_tl` *and others.*)

1.2 Namespaces

The following commands setups a name space. With pdf version <2.0 this is only a prop with the rolemap. With pdf 2.0 a dictionary must be set up. Such a name space dictionaries can contain an optional /Schema and /RoleMapNS entry. We only reserve the objects but delay the writing to the finish code, where we can test if the keys and the name spaces are actually needed. This commands setups objects for the name space and its rolemap. It also initialize a dict to collect the rolemaps if needed, and a property with the tags of the name space and their rolemapping for loops. It is unclear if a reference to a schema file will be ever needed, but it doesn't harm . . . .

g__tag_role/RoleMap_dict   This is the object which contains the normal RoleMap. It is probably not needed in pdf
\g__tag_role_rolemap_prop   2.0 but currently kept.

```
17  \pdfdict_new:n {g__tag_role/RoleMap_dict}
18  \prop_new:N \g__tag_role_rolemap_prop
```

(*End definition for* `g__tag_role/RoleMap_dict` *and* `\g__tag_role_rolemap_prop`.)

__tag_role_NS_new:nnn   __tag_role_NS_new:nnn{⟨shorthand⟩}{⟨URI-ID⟩}Schema

__tag_role_NS_new:nnn

```
19  \pdf_version_compare:NnTF < {2.0}
20  {
21    \cs_new_protected:Npn \__tag_role_NS_new:nnn #1 #2 #3
22    {
23      \prop_new:c { g__tag_role_NS_#1_prop }
24      \prop_new:c { g__tag_role_NS_#1_class_prop }
25      \prop_gput:Nnx \g__tag_role_NS_prop {#1}{}
26    }
27  }
28  {
29    \cs_new_protected:Npn \__tag_role_NS_new:nnn #1 #2 #3
30    {
31      \prop_new:c { g__tag_role_NS_#1_prop }
32      \prop_new:c { g__tag_role_NS_#1_class_prop }
33      \pdf_object_new:n {tag/NS/#1}
34      \pdfdict_new:n     {g__tag_role/Namespace_#1_dict}
35      \pdf_object_new:n {__tag/RoleMapNS/#1}
36      \pdfdict_new:n     {g__tag_role/RoleMapNS_#1_dict}
37      \pdfdict_gput:nnn
38        {g__tag_role/Namespace_#1_dict}
39        {Type}
40        {/Namespace}
41      \pdf_string_from_unicode:nnN{utf8/string}{#2}\l__tag_tmpa_str
42      \tl_if_empty:NF \l__tag_tmpa_str
43      {
44        \pdfdict_gput:nnx
45          {g__tag_role/Namespace_#1_dict}
46          {NS}
47          {\l__tag_tmpa_str}
48      }
49      %RoleMapNS is added in tree
50      \tl_if_empty:nF  {#3}
```

137

```
51        {
52          \pdfdict_gput:nnx{g__tag_role/Namespace_#1_dict}
53          {Schema}{#3}
54        }
55      \prop_gput:Nnx \g__tag_role_NS_prop {#1}{\pdf_object_ref:n{tag/NS/#1}~}
56    }
57  }
```

(*End definition for* `\__tag_role_NS_new:nnn`.)

We need an id for the user space. For the tests it should be possible to set it to a fix value. So we use random numbers which can be fixed by setting a seed. We fake a sort of GUID but do not try to be really exact as it doesn't matter ...

`\c__tag_role_userNS_id_str`

```
58  \str_const:Nx \c__tag_role_userNS_id_str
59    { data:,
60      \int_to_Hex:n{\int_rand:n {65535}}
61      \int_to_Hex:n{\int_rand:n {65535}}
62      -
63      \int_to_Hex:n{\int_rand:n {65535}}
64      -
65      \int_to_Hex:n{\int_rand:n {65535}}
66      -
67      \int_to_Hex:n{\int_rand:n {65535}}
68      -
69      \int_to_Hex:n{\int_rand:n {16777215}}
70      \int_to_Hex:n{\int_rand:n {16777215}}
71    }
```

(*End definition for* `\c__tag_role_userNS_id_str`.)

Now we setup the standard names spaces. The mathml space is currently only loaded for pdf 2.0.

```
72  \__tag_role_NS_new:nnn {pdf}    {http://iso.org/pdf/ssn}{}
73  \__tag_role_NS_new:nnn {pdf2}   {http://iso.org/pdf2/ssn}{}
74  \pdf_version_compare:NnF < {2.0}
75    {
76      \__tag_role_NS_new:nnn {mathml}{http://www.w3.org/1998/Math/MathML}{}
77    }
78  \__tag_role_NS_new:nnn {latex} {https://www.latex-project.org/ns/dflt/2022}{}
79  \__tag_role_NS_new:nnn {latex-book} {https://www.latex-project.org/ns/book/2022}{}
80  \__tag_role_NS_new:nnn {latex-inline} {https://www.latex-project.org/ns/inline/2022}{}
81  \exp_args:Nnx
82    \__tag_role_NS_new:nnn {user}{\c__tag_role_userNS_id_str}{}
```

1.3 Adding a new tag

Both when reading the files and when setting up a tag manually we have to store data in various places.

`\__tag_role_alloctag:nnn` This command allocates a new tag without role mapping. In the lua backend it will also record the attribute value.

```
83  \pdf_version_compare:NnTF < {2.0}
84    {
```

```
85      \sys_if_engine_luatex:TF
86        {
87          \cs_new_protected:Npn \__tag_role_alloctag:nnn #1 #2 #3 %#1 tagname, ns, type
88            {
89              \lua_now:e { ltx.__tag.func.alloctag ('#1') }
90              \prop_gput:Nnn \g__tag_role_tags_NS_prop    {#1}{#2}
91              \prop_gput:cnn {g__tag_role_NS_#2_prop}  {#1}{{}{}}
92              \prop_gput:Nnn \g__tag_role_tags_class_prop {#1}{#3}
93              \prop_gput:cnn {g__tag_role_NS_#2_class_prop}  {#1}{--UNUSED--}
94            }
95        }
96        {
97          \cs_new_protected:Npn \__tag_role_alloctag:nnn #1 #2 #3
98            {
99              \prop_gput:Nnn \g__tag_role_tags_NS_prop    {#1}{#2}
100             \prop_gput:cnn {g__tag_role_NS_#2_prop}  {#1}{{}{}}
101             \prop_gput:Nnn \g__tag_role_tags_class_prop {#1}{#3}
102             \prop_gput:cnn {g__tag_role_NS_#2_class_prop}  {#1}{--UNUSED--}
103           }
104       }
105   }
106   {
107    \sys_if_engine_luatex:TF
108      {
109        \cs_new_protected:Npn \__tag_role_alloctag:nnn #1 #2 #3 %#1 tagname, ns, type
110          {
111            \lua_now:e { ltx.__tag.func.alloctag ('#1') }
112            \prop_gput:Nnn \g__tag_role_tags_NS_prop    {#1}{#2}
113            \prop_gput:cnn {g__tag_role_NS_#2_prop}  {#1}{{}{}}
114            \prop_gput:Nnn \g__tag_role_tags_class_prop {#1}{--UNUSED--}
115            \prop_gput:cnn {g__tag_role_NS_#2_class_prop}  {#1}{#3}
116          }
117      }
118      {
119        \cs_new_protected:Npn \__tag_role_alloctag:nnn #1 #2 #3
120          {
121            \prop_gput:Nnn \g__tag_role_tags_NS_prop    {#1}{#2}
122            \prop_gput:cnn {g__tag_role_NS_#2_prop}  {#1}{{}{}}
123            \prop_gput:Nnn \g__tag_role_tags_class_prop {#1}{--UNUSED--}
124            \prop_gput:cnn {g__tag_role_NS_#2_class_prop}  {#1}{#3}
125          }
126      }
127   }
128 \cs_generate_variant:Nn  \__tag_role_alloctag:nnn {nnV}
```

(*End definition for* `\__tag_role_alloctag:nnn`.)

1.3.1   pdf 1.7 and earlier

`\__tag_role_add_tag:nn`   The pdf 1.7 version has only two arguments: new and rolemap name. The role must be an existing tag and should not be empty. We allow to change the role of an existing tag: as the rolemap is written at the end not confusion can happen.

```
129 \cs_new_protected:Nn \__tag_role_add_tag:nn % (new) name, reference to old
130   {
```

checks and messages

```
131       \__tag_check_add_tag_role:nn {#1}{#2}
132       \prop_if_in:NnF \g__tag_role_tags_NS_prop {#1}
133         {
134           \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
135             {
136               \msg_info:nnn { tag }{new-tag}{#1}
137             }
138         }
```

now the addition

```
139       \prop_get:NnN \g__tag_role_tags_class_prop {#2}\l__tag_tmpa_tl
140       \quark_if_no_value:NT \l__tag_tmpa_tl
141         {
142           \tl_set:Nn\l__tag_tmpa_tl{--UNKNOWN--}
143         }
144       \__tag_role_alloctag:nnV {#1}{user}\l__tag_tmpa_tl
```

We resolve rolemapping recursively so that all targets are stored as standard tags.

```
145       \tl_if_empty:nF { #2 }
146         {
147           \prop_get:NnN \g__tag_role_rolemap_prop {#2}\l__tag_tmpa_tl
148           \quark_if_no_value:NTF \l__tag_tmpa_tl
149             {
150               \prop_gput:Nnx \g__tag_role_rolemap_prop {#1}{\tl_to_str:n{#2}}
151             }
152             {
153               \prop_gput:NnV \g__tag_role_rolemap_prop {#1}\l__tag_tmpa_tl
154             }
155         }
156   }
157 \cs_generate_variant:Nn \__tag_role_add_tag:nn {VV,ne}
```

(*End definition for* `\__tag_role_add_tag:nn`.)

For the parent-child test we must be able to get the role. We use the same number of arguments as for the 2.0 command. If there is no role, we assume a standard tag.

`\__tag_role_get:nnN`

```
158 \pdf_version_compare:NnT < {2.0}
159   {
160     \cs_new:Npn \__tag_role_get:nnN #1#2#3
161       {
162         \prop_get:NnNF \g__tag_role_rolemap_prop {#1}#3
163           {
164             \tl_set:Nn #3 {#1}
165           }
166       }
167     \cs_generate_variant:Nn \__tag_role_get:nnN {VVN}
168   }
169
```

(*End definition for* `\__tag_role_get:nnN`.)

1.3.2 The pdf 2.0 version

__tag_role_add_tag:nnnn   The pdf 2.0 version takes four arguments: tag/namespace/role/namespace

```
170 \cs_new_protected:Nn \__tag_role_add_tag:nnnn %tag/namespace/role/namespace
171   {
172     \__tag_check_add_tag_role:nnn {#1/#2}{#3}{#4}
173     \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
174       {
175         \msg_info:nnn { tag }{new-tag}{#1}
176       }
177     \prop_get:cnN { g__tag_role_NS_#4_class_prop } {#3}\l__tag_tmpa_tl
178     \quark_if_no_value:NT \l__tag_tmpa_tl
179       {
180         \tl_set:Nn\l__tag_tmpa_tl{--UNKNOWN--}
181       }
182     \__tag_role_alloctag:nnV {#1}{#2}\l__tag_tmpa_tl
```

Do not remap standard tags. TODO add warning?

```
183     \tl_if_in:nnF {-pdf-pdf2-mathml-}{-#2-}
184       {
185         \pdfdict_gput:nnx {g__tag_role/RoleMapNS_#2_dict}{#1}
186           {
187             [
188               \pdf_name_from_unicode_e:n{#3}
189               \c_space_tl
190               \pdf_object_ref:n {tag/NS/#4}
191             ]
192           }
193       }
```

We resolve rolemapping recursively so that all targets are stored as standard tags for the tests.

```
194     \tl_if_empty:nF { #2 }
195       {
196         \prop_get:cnN { g__tag_role_NS_#4_prop } {#3}\l__tag_tmpa_tl
197         \quark_if_no_value:NTF \l__tag_tmpa_tl
198           {
199             \prop_gput:cnx { g__tag_role_NS_#2_prop } {#1}
200               {{\tl_to_str:n{#3}}{\tl_to_str:n{#4}}}
201           }
202           {
203             \prop_gput:cno { g__tag_role_NS_#2_prop } {#1}{\l__tag_tmpa_tl}
204           }
205       }
206   }
207 \cs_generate_variant:Nn \__tag_role_add_tag:nnnn {VVVV}
```

(*End definition for* __tag_role_add_tag:nnnn.)

For the parent-child test we must be able to get the role. We use the same number of arguments as for the <2.0 command (and assume that we don't need a name space)

__tag_role_get:nnN

```
208 \pdf_version_compare:NnF < {2.0}
209   {
210     \cs_new:Npn \__tag_role_get:nnN #1#2#3
```

```
211        {
212          \prop_get:cnNTF {g__tag_role_NS_#2_prop} {#1}\l__tag_get_tmpc_tl
213            {
214              \tl_set:Nx #3 {\exp_last_unbraced:NV\use_i:nn  \l__tag_get_tmpc_tl}
215            }
216            {
217              \tl_set:Nn #3 {#1}
218            }
219        }
220      \cs_generate_variant:Nn \__tag_role_get:nnN {VVN}
221    }
```

(*End definition for* __tag_role_get:nnN.)

1.4   Helper command to read the data from files

In this section we setup the helper command to read namespace files.

__tag_role_read_namespace_line:nw   This command will process a line in the name space file. The first argument is the name of the name space. The definition differ for pdf 2.0. as we have proper name spaces there. With pdf<2.0 not special name spaces shouldn't update the default role or add to the rolemap again. We use a boolean here.

```
222  \bool_new:N\l__tag_role_update_bool
223  \bool_set_true:N \l__tag_role_update_bool
224  \pdf_version_compare:NnTF < {2.0}
225    {
226      \cs_new_protected:Npn \__tag_role_read_namespace_line:nw #1#2,#3,#4,#5,#6\q_stop %
227      % #1 NS, #2 tag, #3 rolemap, #4 NS rolemap #5 type
228        {
229          \tl_if_empty:nF { #2 }
230            {
231              \bool_if:NTF \l__tag_role_update_bool
232                {
233                  \tl_if_empty:nTF {#5}
234                    {
235                      \prop_get:NnN \g__tag_role_tags_class_prop  {#3}\l__tag_tmpa_tl
236                      \quark_if_no_value:NT \l__tag_tmpa_tl
237                        {
238                          \tl_set:Nn\l__tag_tmpa_tl{--UNKNOWN--}
239                        }
240                    }
241                    {
242                      \tl_set:Nn \l__tag_tmpa_tl {#5}
243                    }
244                  \__tag_role_alloctag:nnV {#2}{#1}\l__tag_tmpa_tl
245                  \tl_if_eq:nnF {#2}{#3}
246                    {
247                      \__tag_role_add_tag:nn {#2}{#3}
248                    }
249                  \prop_gput:cnn {g__tag_role_NS_#1_prop}  {#2}{{#3}{}}
250                }
251                {
252                  \prop_gput:cnn {g__tag_role_NS_#1_prop}  {#2}{{#3}{}}
253                  \prop_gput:cnn {g__tag_role_NS_#1_class_prop}  {#2}{--UNUSED--}
```

142

```
254        }
255      }
256    }
257  }
258  {
259    \cs_new_protected:Npn \__tag_role_read_namespace_line:nw #1#2,#3,#4,#5,#6\q_stop %
260    % #1 NS, #2 tag, #3 rolemap, #4 NS rolemap #5 type
261    {
262      \tl_if_empty:nF {#2}
263        {
264          \tl_if_empty:nTF {#5}
265            {
266              \prop_get:cnN { g__tag_role_NS_#4_class_prop } {#3}\l__tag_tmpa_tl
267              \quark_if_no_value:NT \l__tag_tmpa_tl
268                {
269                  \tl_set:Nn\l__tag_tmpa_tl{--UNKNOWN--}
270                }
271            }
272            {
273              \tl_set:Nn \l__tag_tmpa_tl {#5}
274            }
275          \__tag_role_alloctag:nnV {#2}{#1}\l__tag_tmpa_tl
276          \bool_lazy_and:nnT
277            { ! \tl_if_empty_p:n {#3} }{! \str_if_eq_p:nn {#1}{pdf2}}
278            {
279              \__tag_role_add_tag:nnnn {#2}{#1}{#3}{#4}
280            }
281          \prop_gput:cnn {g__tag_role_NS_#1_prop}  {#2}{{#3}{#4}}
282        }
283    }
284  }
```

(*End definition for* `\__tag_role_read_namespace_line:nw`.)

`\__tag_role_read_namespace:n` This command reads the namespace file.

```
285  \cs_new_protected:Npn \__tag_role_read_namespace:n #1 %name of namespace
286    {
287      \prop_if_exist:cF {g__tag_role_NS_#1_prop}
288        { \msg_warning:nnn {tag}{namespace-unknown}{#1} }
289      \file_if_exist:nTF { tagpdf-ns-#1.def}
290        {
291          \ior_open:Nn \g_tmpa_ior {tagpdf-ns-#1.def}
292          \msg_info:nnn {tag}{read-namespace}{#1}
293          \ior_map_inline:Nn \g_tmpa_ior
294            {
295              \__tag_role_read_namespace_line:nw {#1} ##1,,,,\q_stop
296            }
297          \ior_close:N\g_tmpa_ior
298        }
299        {
300          \msg_warning:nnn{tag}{namespace-missing}{#1}
301        }
302    }
303
```

143

(*End definition for* `\__tag_role_read_namespace:n`.)

1.5 Reading the default data

The order is important as we want pdf2 and latex as default.

```
304 \__tag_role_read_namespace:n {pdf}
305 \__tag_role_read_namespace:n {pdf2}
306 \pdf_version_compare:NnF < {2.0}
307   {\__tag_role_read_namespace:n {mathml}}
308 \bool_set_false:N\l__tag_role_update_bool
309 \__tag_role_read_namespace:n {latex-inline}
310 \__tag_role_read_namespace:n {latex-book}
311 \bool_set_true:N\l__tag_role_update_bool
312 \__tag_role_read_namespace:n {latex}
313 \__tag_role_read_namespace:n {pdf}
314 \__tag_role_read_namespace:n {pdf2}
```

But is the class provides a `\chapter` command then we switch

```
315 \pdf_version_compare:NnTF < {2.0}
316   {
317     \hook_gput_code:nnn {begindocument}{tagpdf}
318       {
319         \cs_if_exist:NT \chapter
320           {
321             \prop_map_inline:cn{g__tag_role_NS_latex-book_prop}
322               {
323                 \__tag_role_add_tag:ne {#1}{\use_i:nn #2\c_empty_tl\c_empty_tl}
324               }
325           }
326       }
327   }
328   {
329     \hook_gput_code:nnn {begindocument}{tagpdf}
330       {
331         \cs_if_exist:NT \chapter
332           {
333             \prop_map_inline:cn{g__tag_role_NS_latex-book_prop}
334               {
335                 \prop_gput:Nnn \g__tag_role_tags_NS_prop    { #1 }{ latex-book }
336               }
337           }
338       }
339   }
```

1.6 Parent-child rules

PDF define various rules about which tag can be a child of another tag. The following code implements the matrix to allow to use it in tests.

`\g_tag_role_parent_child_intarray` This intarray will store the rule as a number. For parent nm and child ij (n,m,i,j digits) the rule is at position nmij. As we have around 56 tags, we need roughly a size 6000.

```
340 \intarray_new:Nn \g__tag_role_parent_child_intarray {6000}
```

(*End definition for* `\g__tag_role_parent_child_intarray`.)

`\c__tag_role_rules_prop`
`\c__tag_role_rules_num_prop` These two properties map the rule strings to numbers and back. There are in tagpdf-data.dtx near the csv files for easier maintenance.

(*End definition for* `\c__tag_role_rules_prop` *and* `\c__tag_role_rules_num_prop`.)

`\__tag_store_parent_child_rule:nnn` The helper command is used to store the rule. It assumes that parent and child are given as 2-digit number!

```
341 \cs_new_protected:Npn \__tag_store_parent_child_rule:nnn #1 #2 #3 % num parent, num child, #3
342   {
343     \intarray_gset:Nnn \g__tag_role_parent_child_intarray
344       { #1#2 }{0\prop_item:Nn\c__tag_role_rules_prop{#3}}
345   }
```

(*End definition for* `\__tag_store_parent_child_rule:nnn`.)

1.6.1 Reading in the csv-files

This counter will be used to identify the first (non-comment) line

```
346 \int_zero:N  \l__tag_tmpa_int
```

Open the file depending on the PDF version

```
347 \pdf_version_compare:NnTF < {2.0}
348   {
349     \ior_open:Nn \g_tmpa_ior {tagpdf-parent-child.csv}
350   }
351   {
352     \ior_open:Nn \g_tmpa_ior {tagpdf-parent-child-2.csv}
353   }
```

Now the main loop over the file

```
354 \ior_map_inline:Nn \g_tmpa_ior
355   {
```

ignore lines containing only comments

```
356     \tl_if_empty:nF{#1}
357       {
```

count the lines ...

```
358         \int_incr:N\l__tag_tmpa_int
```

put the line into a seq. Attention! empty cells are dropped.

```
359         \seq_set_from_clist:Nn\l__tag_tmpa_seq { #1 }
360         \int_compare:nNnTF {\l__tag_tmpa_int}=1
```

This handles the header line. It gives the tags 2-digit numbers

```
361           {
362             \seq_map_indexed_inline:Nn \l__tag_tmpa_seq
363               {
364                 \prop_gput:Nnx\g__tag_role_index_prop
365                   {##2}
366                   {\int_compare:nNnT{##1}<{10}{0}##1}
367               }
368           }
```

now the data lines.

```
369           {
370             \seq_set_from_clist:Nn\l__tag_tmpa_seq { #1 }
```

145

get the name of the child tag from the first column

```
371                \seq_pop_left:NN\l__tag_tmpa_seq\l__tag_tmpa_tl
```

get the number of the child, and store it in `\l__tag_tmpb_tl`

```
372                \prop_get:NVN \g__tag_role_index_prop \l__tag_tmpa_tl \l__tag_tmpb_tl
```

remove column 2+3

```
373                \seq_pop_left:NN\l__tag_tmpa_seq\l__tag_tmpa_tl
374                \seq_pop_left:NN\l__tag_tmpa_seq\l__tag_tmpa_tl
```

Now map over the rest. The index `##1` gives us the number of the parent, `##2` is the data.

```
375                \seq_map_indexed_inline:Nn \l__tag_tmpa_seq
376                  {
377                    \exp_args:Nnx
378                    \__tag_store_parent_child_rule:nnn {##1}{\l__tag_tmpb_tl}{ ##2 }
379                  }
380            }
381        }
382    }
```

close the read handle.

```
383 \ior_close:N\g_tmpa_ior
```

The Root, Hn and mathml tags are special and need to be added explicitly

```
384 \prop_get:NnN\g__tag_role_index_prop{StructTreeRoot}\l__tag_tmpa_tl
385 \prop_gput:Nnx\g__tag_role_index_prop{Root}{\l__tag_tmpa_tl}
386 \prop_get:NnN\g__tag_role_index_prop{Hn}\l__tag_tmpa_tl
387 \pdf_version_compare:NnTF < {2.0}
388    {
389      \int_step_inline:nn{6}
390        {
391          \prop_gput:Nnx\g__tag_role_index_prop{H#1}{\l__tag_tmpa_tl}
392        }
393    }
394    {
395      \int_step_inline:nn{10}
396        {
397          \prop_gput:Nnx\g__tag_role_index_prop{H#1}{\l__tag_tmpa_tl}
398        }
```

all mathml tags are currently handled identically

```
399      \prop_get:NnN\g__tag_role_index_prop {mathml}\l__tag_tmpa_tl
400      \prop_get:NnN\g__tag_role_index_prop {math}\l__tag_tmpb_tl
401      \prop_map_inline:Nn \g__tag_role_NS_mathml_prop
402        {
403          \prop_gput:NnV\g__tag_role_index_prop{#1}\l__tag_tmpa_tl
404        }
405      \prop_gput:NnV\g__tag_role_index_prop{math}\l__tag_tmpb_tl
406    }
```

1.6.2 Retrieving the parent-child rule

`\l__tag_role_real_parent_tl` Part, Div and NonStruct have no own rules, instead the parent(s) have to be inspected. To store this real parent we use this tlvar

```
407 \tl_new:N \l__tag_role_real_parent_tl
```

(*End definition for* `\l__tag_role_real_parent_tl`.)

`\__tag_role_get_parent_child_rule:nnN` This command retrieves the rule (as a number) and stores it in the tl-var. TODO check temporary variables. Check if the tl-var should be fix. The arguments should be standard tags for which a rule exist and role mapping should have already be done.

```
408 \tl_new:N \l__tag_parent_child_check_tl
409 \cs_new_protected:Npn \__tag_role_get_parent_child_rule:nnN #1 #2 #3
410   % #1 parent (string) #2 child (string) #3 tl for state
411   {
412     \tl_set:Nn \l__tag_role_real_parent_tl {#1}

413     \prop_get:NnN \g__tag_role_index_prop{#1}\l__tag_tmpa_tl
414     \prop_get:NnN \g__tag_role_index_prop{#2}\l__tag_tmpb_tl
415     \bool_lazy_and:nnTF
416       { ! \quark_if_no_value_p:N \l__tag_tmpa_tl }
417       { ! \quark_if_no_value_p:N \l__tag_tmpb_tl }
418       {
```

Get the rule from the intarray

```
419         \tl_set:Nx#3
420           {
421             \intarray_item:Nn
422             \g__tag_role_parent_child_intarray
423             {\l__tag_tmpa_tl\l__tag_tmpb_tl}
424           }
```

If the state is we have to check the parents from the stack and use the first which is not Part, Div or NonStruct

```
425         \int_compare:nNnT
426           {#3} = {\prop_item:Nn\c__tag_role_rules_prop{}}
427           {
428             \seq_set_eq:NN   \l__tag_role_tmpa_seq \g__tag_struct_tag_stack_seq
```

we must take the current child from the stack if is already there, depending on location the check is called, this could also remove the parent, but that is ok too.

```
429             \seq_pop_left:NN \l__tag_role_tmpa_seq\l__tag_get_tmpc_tl
430             \seq_map_inline:Nn\l__tag_role_tmpa_seq
431               {
432                 \tl_set:Nx\l__tag_tmpa_tl { \use_ii:nn ##1 }
433                 \exp_args:Nne
434                 \str_if_in:nnF {-Part-Div-NonStruct-}{-\l__tag_tmpa_tl-}
435                   {
436                     \tl_set:Nn\l__tag_role_real_parent_tl {##1}
437                     \int_zero:N\l__tag_tmpa_int
438                     \exp_args:NV
439                     \__tag_role_get_parent_child_rule:nnN \l__tag_tmpa_tl{#2}#3
440                     \int_set:Nn\l__tag_tmpa_int{1}
441                     \seq_map_break:
442                   }
```

```
443                    }
444                }
```
This is the message, this can perhaps go into debug mode.
```
445            \group_begin:
446            \int_compare:nNnT {\l__tag_tmpa_int*\l__tag_loglevel_int} > { 0 }
447              {
448                \prop_get:NVNF\c__tag_role_rules_num_prop #3 \l__tag_tmpa_tl
449                  {
450                    \tl_set:Nn \l__tag_tmpa_tl {unknown}
451                  }
452                \tl_set:Nn \l__tag_tmpb_tl {#1}
453                \msg_note:nnxxx
454                  { tag }
455                  { role-parent-child }
456                  { #1
457                    \tl_if_eq:NNTF\l__tag_tmpb_tl\l__tag_role_real_parent_tl
458                      {
459                        \bool_lazy_and:nnT
460                          {
461                            \prop_if_in_p:Nn \l__tag_role_debug_prop {parent}
462                          }
463                          {
464                            !\str_if_eq_p:ee {#1}{\prop_item:Nn\l__tag_role_debug_prop {parent}}
465                          }
466                          {
467                            \c_space_tl (from~\prop_item:Nn\l__tag_role_debug_prop {parent})
468                          }
469                      }
470                      {
471                        \c_space_tl(inherited~from~\l__tag_role_real_parent_tl)
472                      }
473                  }
474                  {
475                     #2
476                    \bool_lazy_and:nnT
477                      {
478                        \prop_if_in_p:Nn \l__tag_role_debug_prop {child}
479                      }
480                      {
481                        !\str_if_eq_p:ee {#2}{\prop_item:Nn\l__tag_role_debug_prop {child}}
482                      }
483                      {
484                        \c_space_tl (from~\prop_item:Nn\l__tag_role_debug_prop {child})
485                      }
486                  }
487                  { '#3~(\l__tag_tmpa_tl)' }
488              }
489            \group_end:
490          }
491          {
492            \tl_set:Nn#3 {0}
493            \msg_warning:nnxxx
494              { tag }
495              {role-parent-child}
```

```
496          { #1 }
497          { #2 }
498          { unknown! }
499       }
500   }
501 \cs_generate_variant:Nn\__tag_role_get_parent_child_rule:nnN {VVN}
```

(*End definition for* `\__tag_role_get_parent_child_rule:nnN`.)

`__tag_check_parent_child:nnnnN`   This is the main command. It has to retrieve the standard tags for a comparison. In pdf 2.0 the name spaces of the tags are relevant, so we have arguments for them, but in pdf <2.0 they are ignored and can be left empty.

```
502 \pdf_version_compare:NnTF < {2.0}
503   {
504    \cs_new_protected:Npn \__tag_check_parent_child:nnnnN #1 #2 #3 #4 #5
505       {
```

for debugging messages we store the arguments.

```
506          \prop_put:Nnn \l__tag_role_debug_prop {parent} {#1}
507          \prop_put:Nnn \l__tag_role_debug_prop {child}  {#3}
```

get the standard tags through rolemapping if needed at first the parent

```
508          \prop_get:NnNTF \g__tag_role_index_prop {#1}\l__tag_tmpa_tl
509            {
510              \tl_set:Nn \l__tag_tmpa_tl {#1}
511            }
512            {
513              \prop_get:NnNF \g__tag_role_rolemap_prop {#1}\l__tag_tmpa_tl
514                {
515                  \tl_set:Nn \l__tag_tmpa_tl {\q_no_value}
516                }
517            }
```

now the child

```
518          \prop_get:NnNTF \g__tag_role_index_prop {#3}\l__tag_tmpb_tl
519            {
520              \tl_set:Nn \l__tag_tmpb_tl {#3}
521            }
522            {
523              \prop_get:NnNF \g__tag_role_rolemap_prop {#3}\l__tag_tmpb_tl
524                {
525                  \tl_set:Nn \l__tag_tmpb_tl {\q_no_value}
526                }
527            }
```

if we got tags for parent and child we call the checking command

```
528          \bool_lazy_and:nnTF
529          { ! \quark_if_no_value_p:N \l__tag_tmpa_tl }
530          { ! \quark_if_no_value_p:N \l__tag_tmpb_tl }
531          {
532            \__tag_role_get_parent_child_rule:VVN \l__tag_tmpa_tl \l__tag_tmpb_tl #5
533          }
534          {
535            \tl_set:Nn #5 {0}
536            \msg_warning:nnxxx
537             { tag }
```

149

```
538          {role-parent-child}
539          { #1 }
540          { #3 }
541          { unknown! }
542        }
543      }
544    \cs_new_protected:Npn \__tag_check_parent_child:nnN #1#2#3
545      {
546        \__tag_check_parent_child:nnnnN {#1}{}{#2}{}#3
547      }
548  }
```

and now the pdf 2.0 version The version with three arguments retrieves the default names
space and then calls the full command. Not sure if this will ever be needed but we leave
it for now.

```
549  {
550    \cs_new_protected:Npn \__tag_check_parent_child:nnN #1 #2 #3
551      {
552        \prop_get:NnN\g__tag_role_tags_NS_prop {#1}\l__tag_role_tag_namespace_tmpa_tl
553        \prop_get:NnN\g__tag_role_tags_NS_prop {#2}\l__tag_role_tag_namespace_tmpb_tl
554        \str_if_eq:nnT{#2}{MC}{\tl_clear:N \l__tag_role_tag_namespace_tmpb_tl}
555        \bool_lazy_and:nnTF
556          { ! \quark_if_no_value_p:N \l__tag_role_tag_namespace_tmpa_tl }
557          { ! \quark_if_no_value_p:N \l__tag_role_tag_namespace_tmpb_tl }
558          {
559            \__tag_check_parent_child:nVnVN
560              {#1}\l__tag_role_tag_namespace_tmpa_tl
561              {#2}\l__tag_role_tag_namespace_tmpb_tl
562              #3
563          }
564          {
565            \tl_set:Nn #3 {0}
566            \msg_warning:nnxxx
567            { tag }
568            {role-parent-child}
569            { #1 }
570            { #2 }
571            { unknown! }
572          }
573      }
```

and now the real command.

```
574    \cs_new_protected:Npn \__tag_check_parent_child:nnnnN #1 #2 #3 #4 #5 %tag,NS,tag,NS, tl va
575      {
576        \prop_put:Nnn \l__tag_role_debug_prop {parent} {#1/#2}
577        \prop_put:Nnn \l__tag_role_debug_prop {child}  {#3/#4}
```

If the namespace is empty, we assume a standard tag, otherwise we retrieve the rolemap-
ping from the namespace

```
578        \tl_if_empty:nTF  {#2}
579          {
580            \tl_set:Nn \l__tag_tmpa_tl {#1}
581          }
582          {
583            \prop_get:cnNTF
```

150

```
584              { g__tag_role_NS_#2_prop }
585              {#1}
586              \l__tag_tmpa_tl
587              {
588                \tl_set:Nx \l__tag_tmpa_tl {\tl_head:N\l__tag_tmpa_tl}
589                \tl_if_empty:NT\l__tag_tmpa_tl
590                  {
591                     \tl_set:Nn \l__tag_tmpa_tl {#1}
592                  }
593              }
594              {
595                \tl_set:Nn \l__tag_tmpa_tl {\q_no_value}
596              }
597          }
```

and the same for the child If the namespace is empty, we assume a standard tag, otherwise we retrieve the rolemapping from the namespace

```
598          \tl_if_empty:nTF  {#4}
599            {
600              \tl_set:Nn \l__tag_tmpb_tl {#3}
601            }
602            {
603              \prop_get:cnNTF
604                { g__tag_role_NS_#4_prop }
605                {#3}
606                \l__tag_tmpb_tl
607                {
608                   \tl_set:Nx \l__tag_tmpb_tl { \tl_head:N\l__tag_tmpb_tl }
609                   \tl_if_empty:NT\l__tag_tmpb_tl
610                     {
611                        \tl_set:Nn \l__tag_tmpb_tl {#3}
612                     }
613                }
614                {
615                   \tl_set:Nn \l__tag_tmpb_tl {\q_no_value}
616                }
617            }
```

and now get the relation

```
618          \bool_lazy_and:nnTF
619            { ! \quark_if_no_value_p:N \l__tag_tmpa_tl }
620            { ! \quark_if_no_value_p:N \l__tag_tmpb_tl }
621            {
622              \__tag_role_get_parent_child_rule:VVN \l__tag_tmpa_tl \l__tag_tmpb_tl #5
623            }
624            {
625              \tl_set:Nn #5 {0}
626              \msg_warning:nnxxx
627              { tag }
628              {role-parent-child}
629              { #1 }
630              { #3 }
631              { unknown! }
632            }
633      }
```

```
634        }
635  \cs_generate_variant:Nn\__tag_check_parent_child:nnN {VVN}
636  \cs_generate_variant:Nn\__tag_check_parent_child:nnnnN {VVVVN,nVnVN,VVnnN}
637  ⟨/package⟩
```

(*End definition for* `__tag_check_parent_child:nnnnN`.)

`\tag_check_child:nnTF`

```
638  ⟨base⟩\prg_new_protected_conditional:Npnn \tag_check_child:nn #1 #2 {T,F,TF}{\prg_return_true:}
639  ⟨*package⟩
640  \prg_set_protected_conditional:Npnn \tag_check_child:nn #1 #2 {T,F,TF}
641    {
642      \seq_get:NN\g__tag_struct_stack_seq\l__tag_tmpa_tl
643      \__tag_struct_get_tag_info:eNN
644        {\l__tag_tmpa_tl}
645        \l__tag_get_parent_tmpa_tl
646        \l__tag_get_parent_tmpb_tl
647      \__tag_check_parent_child:VVnnN
648        \l__tag_get_parent_tmpa_tl
649        \l__tag_get_parent_tmpb_tl
650        {#1}{#2}
651        \l__tag_parent_child_check_tl
652      \int_compare:nNnTF {  \l__tag_parent_child_check_tl } < {0}
653        {\prg_return_false:}
654        {\prg_return_true:}
655    }
```

(*End definition for* `\tag_check_child:nnTF`. *This function is documented on page 134.*)

1.7 Remapping of tags

In some context it can be necessary to remap or replace the tags. That means instead of tag=H1 or tag=section one wants the effect of tag=Span. Or instead of tag=P one wants tag=Code.

The following command provide some general interface for this. The core idea is that before a tag is set it is fed through a function that can change it. We want to be able to chain such functions, so all of them manipulate the same variables.

`\l__tag_role_remap_tag_tl`
`\l__tag_role_remap_NS_tl`

```
656  \tl_new:N \l__tag_role_remap_tag_tl
657  \tl_new:N \l__tag_role_remap_NS_tl
```

(*End definition for* `\l__tag_role_remap_tag_tl` *and* `\l__tag_role_remap_NS_tl`.)

`\__tag_role_remap:` This function is used in the structure and the mc code before using a tag. By default it does nothing with the tl vars. Perhaps this should be a hook?

```
658  \cs_new_protected:Npn \__tag_role_remap: {  }
```

(*End definition for* `\__tag_role_remap:`.)

`\__tag_role_remap_id:` This is copy in case we have to restore the main command.

```
659  \cs_set_eq:NN \__tag_role_remap_id: \__tag_role_remap:
```

(*End definition for* `\__tag_role_remap_id:`.)

152

`\__tag_role_remap_inline:` The mapping is meant to "degrade" tags, e.g. if used inside some complex object. The pdf<2.0 code maps the tag to the new role, the pdf 2.0 code only switch the NS.

```
660 \pdf_version_compare:NnTF < {2.0}
661   {
662     \cs_new_protected:Npn \__tag_role_remap_inline:
663       {
664         \prop_get:cVNT { g__tag_role_NS_latex-inline_prop }\l__tag_role_remap_tag_tl\l__tag_t
665           {
666             \tl_set:Nx\l__tag_role_remap_tag_tl
667               {
668                 \exp_last_unbraced:NV\use_i:nn \l__tag_tmpa_tl
669               }
670             \tl_set:Nx\l__tag_role_remap_NS_tl
671               {
672                 \exp_last_unbraced:NV\use_ii:nn \l__tag_tmpa_tl
673               }
674           }
675         \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
676           {
677             \msg_note:nnx { tag } { role-remapping }{ \l__tag_role_remap_tag_tl }
678           }
679       }
680   }
681   {
682     \cs_new_protected:Npn \__tag_role_remap_inline:
683       {
684         \prop_get:cVNT { g__tag_role_NS_latex-inline_prop }\l__tag_role_remap_tag_tl\l__tag_t
685           {
686             \tl_set:Nn\l__tag_role_remap_NS_tl {latex-inline}
687           }
688         \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
689           {
690             \msg_note:nnx { tag } { role-remapping }{ \l__tag_role_remap_tag_tl/latex-
   inline }
691           }
692       }
693   }
```

(*End definition for* `\__tag_role_remap_inline:`.)

1.8 Key-val user interface

The user interface uses the key `add-new-tag`, which takes either a keyval list as argument, or a tag/role.

tag␣(rolemap-key)
tag-namespace␣(rolemap-key)
role␣(rolemap-key)
role-namespace␣(rolemap-key)
add-new-tag␣(setup-key)

```
694 \keys_define:nn { __tag / tag-role }
695   {
696     ,tag .tl_set:N = \l__tag_role_tag_tmpa_tl
697     ,tag-namespace .tl_set:N = \l__tag_role_tag_namespace_tmpa_tl
698     ,role .tl_set:N = \l__tag_role_role_tmpa_tl
699     ,role-namespace .tl_set:N = \l__tag_role_role_namespace_tmpa_tl
700   }
701
```

153

```
702  \keys_define:nn { __tag / setup }
703    {
704      add-new-tag .code:n =
705        {
706          \keys_set_known:nnnN
707            {__tag/tag-role}
708            {
709              tag-namespace=user,
710              role-namespace=, %so that we can test for it.
711             #1
712            }{__tag/tag-role}\l_tmpa_tl
713          \tl_if_empty:NF \l_tmpa_tl
714            {
715              \exp_args:NNno \seq_set_split:Nnn \l_tmpa_seq { / } {\l_tmpa_tl/}
716              \tl_set:Nx \l__tag_role_tag_tmpa_tl  { \seq_item:Nn \l_tmpa_seq {1} }
717              \tl_set:Nx \l__tag_role_role_tmpa_tl { \seq_item:Nn \l_tmpa_seq {2} }
718            }
719          \tl_if_empty:NT \l__tag_role_role_namespace_tmpa_tl
720            {
721              \prop_get:NVNTF
722                \g__tag_role_tags_NS_prop
723                \l__tag_role_role_tmpa_tl
724                \l__tag_role_role_namespace_tmpa_tl
725                {
726                    \prop_if_in:NVF\g__tag_role_NS_prop \l__tag_role_role_namespace_tmpa_tl
727                     {
728                        \tl_set:Nn \l__tag_role_role_namespace_tmpa_tl {user}
729                     }
730                }
731                {
732                    \tl_set:Nn \l__tag_role_role_namespace_tmpa_tl {user}
733                }
734            }
735          \pdf_version_compare:NnTF < {2.0}
736            {
737             %TODO add check for emptyness?
738              \__tag_role_add_tag:VV
739                  \l__tag_role_tag_tmpa_tl
740                  \l__tag_role_role_tmpa_tl
741            }
742            {
743              \__tag_role_add_tag:VVVV
744                \l__tag_role_tag_tmpa_tl
745                \l__tag_role_tag_namespace_tmpa_tl
746                \l__tag_role_role_tmpa_tl
747                \l__tag_role_role_namespace_tmpa_tl
748            }
749        }
750    }
751  ⟨/package⟩
```

(*End definition for* tag (rolemap-key) *and others. These functions are documented on page 134.*)

Part X

The **tagpdf-space** module
Code related to real space chars
Part of the tagpdf package

This key allows to activate/deactivate the real space chars if the engine supports it. The allowed values are `true`, `on`, `false`, `off`.

This key works only with luatex and shows with small red bars where spaces have been inserted. This is only for debugging and is not completly reliable (and change affect other literals and tagging), so it should be used with care.

```
1  ⟨@@=tag⟩
2  ⟨∗header⟩
3  \ProvidesExplPackage {tagpdf-space-code} {2023-03-07} {0.98e}
4    {part of tagpdf - code related to real space chars}
5  ⟨/header⟩
```

1   Code for interword spaces

The code is engine/backend dependant. Basically only pdftex and luatex support real space chars. Most of the code for luatex which uses attributes is in the lua code, here are only the keys.

```
6  ⟨∗package⟩
7  \keys_define:nn { __tag / setup }
8    {
9      interwordspace .choices:nn = { true, on }
10       { \msg_warning:nnx {tag}{sys-no-interwordspace}{\c_sys_engine_str}  },
11     interwordspace .choices:nn = { false, off }
12       { \msg_warning:nnx {tag}{sys-no-interwordspace}{\c_sys_engine_str}  },
13     interwordspace .default:n = true,
14     show-spaces .bool_set:N = \l__tag_showspaces_bool
15   }
16 \sys_if_engine_pdftex:T
17   {
18     \sys_if_output_pdf:TF
19       {
20         \pdfglyphtounicode{space}{0020}
21         \keys_define:nn { __tag / setup }
22           {
23             interwordspace .choices:nn = { true, on }  { \pdfinterwordspaceon },
24             interwordspace .choices:nn = { false, off }{ \pdfinterwordspaceon },
25             interwordspace .default:n = true,
```

```
26          show-spaces .bool_set:N = \l__tag_showspaces_bool
27        }
28      }
29      {
30        \keys_define:nn { __tag / setup }
31          {
32            interwordspace .choices:nn = { true, on, false, off }
33              { \msg_warning:nnn {tag}{sys-no-interwordspace}{dvi}  },
34            interwordspace .default:n = true,
35            show-spaces .bool_set:N = \l__tag_showspaces_bool
36          }
37      }
38  }
39
40
41 \sys_if_engine_luatex:T
42  {
43    \keys_define:nn { __tag / setup }
44      {
45        interwordspace .choices:nn =
46                              { true, on }
47                              {
48                                \bool_gset_true:N \g__tag_active_space_bool
49                                \lua_now:e{ltx.__tag.func.markspaceon()}
50                              },
51        interwordspace .choices:nn =
52                              { false, off }
53                              {
54                                \bool_gset_false:N \g__tag_active_space_bool
55                                \lua_now:e{ltx.__tag.func.markspaceoff()}
56                              },
57        interwordspace .default:n = true,
58        show-spaces     .choice:,
59        show-spaces  / true   .code:n =
60                              {\lua_now:e{ltx.__tag.trace.showspaces=true}},
61        show-spaces  / false .code:n =
62                              {\lua_now:e{ltx.__tag.trace.showspaces=nil}},
63        show-spaces .default:n = true
64      }
65  }
```

(*End definition for* interwordspace (setup-key) *and* show-spaces (setup-key)*. These functions are documented on page* 155*.*)

__tag_fakespace: For luatex we need a command for the fake space as equivalent of the pdftex primitive.

```
66 \sys_if_engine_luatex:T
67  {
68    \cs_new_protected:Nn \__tag_fakespace:
69      {
70        \group_begin:
71        \lua_now:e{ltx.__tag.func.fakespace()}
72        \skip_horizontal:n{\c_zero_skip}
73        \group_end:
74      }
```

```
75    }
76  ⟨/package⟩
```

(*End definition for* `\__tag_fakespace:`.)

Index

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

161

163

164

165

167