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

Ulrike Fischer[†]

Released 2024-02-02

Contents

---

[*]This file describes v0.98u, last revised 2024-02-02.

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

`\tag_stop:` `\tag_start:` `\tagstop` `\tagstart`	We need commands to stop tagging in some places. They switches three local booleans and also stop the counting of paragraphs. If they are nested an inner `\tag_start:` will not restart tagging.

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

The commands with argument allow to give a label. This is only used in debugging messages to allow to follow the nesting.

`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} {2024-02-02} {0.98u}
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} {2024-02-02} {0.98u}
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} {2024-02-02} {0.98u}
38   {part of tagpdf - provide base, no-op versions of the user commands }
39 ⟨/base⟩
```

3 Package options

There are only two documented options to switch for luatex between generic and luamode, TODO try to get rid of them. The option `disabledelayedshipout` is only temporary to be able to debug problem with the new shipout keyword if needed.

```
40 ⟨*package⟩
41 \bool_new:N\g__tag_mode_lua_bool
42 \bool_new:N\g__tag_delayed_shipout_bool
43 \bool_lazy_and:nnT
44   { \bool_if_exist_p:N \l__pdfmanagement_delayed_shipout_bool }
45   { \l__pdfmanagement_delayed_shipout_bool }
46   {
47     \bool_gset_true:N\g__tag_delayed_shipout_bool
48   }
49 \DeclareOption {luamode}  { \sys_if_engine_luatex:T { \bool_gset_true:N \g__tag_mode_lua_bool
50 \DeclareOption {genericmode}{ \bool_gset_false:N\g__tag_mode_lua_bool }
51 \DeclareOption {disabledelayedshipout}{ \bool_gset_false:N\g__tag_delayed_shipout_bool }
52 \ExecuteOptions{luamode}
53 \ProcessOptions
```

4 Packages

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

```
54 \RequirePackage{tagpdf-base}
55 ⟨/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:

```
56 ⟨*base⟩
57 \AddToHook{begindocument}
58  {
59   \str_case:VnF \c_sys_backend_str
60    {
61     { luatex  } { \cs_new_protected:Npn \__tag_whatsits: {} }
62     { dvisvgm } { \cs_new_protected:Npn \__tag_whatsits: {} }
63    }
64    {
65     \cs_new_protected:Npn \__tag_whatsits: {\tex_special:D {} }
66    }
67  }
68 ⟨/base⟩
```

4.1 a LastPage label

See also issue #2 in Accessible-xref

`\__tag_lastpagelabel:`

```
69 ⟨*package⟩
70 \cs_new_protected:Npn \__tag_lastpagelabel:
71  {
72    \legacy_if:nT { @filesw }
73      {
```

```
74        \exp_args:NNne \exp_args:NNe\iow_now:Nn \@auxout
75          {
76            \token_to_str:N \new@label@record
77              {@tag@LastPage}
78              {
79                {abspage} { \int_use:N \g_shipout_readonly_int}
80                {tagmcabs}{ \int_use:N \c@g__tag_MCID_abs_int }
81                {tagstruct}{\int_use:N \c@g__tag_struct_abs_int }
82              }
83          }
84      }
85  }
86
87  \AddToHook{enddocument/afterlastpage}
88    {\__tag_lastpagelabel:}
```

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

5 Variables

\l__tag_tmpa_tl A few temporary variables
\l__tag_tmpb_tl
\l__tag_get_tmpc_tl
_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

```
89  \tl_new:N     \l__tag_tmpa_tl
90  \tl_new:N     \l__tag_tmpb_tl
91  \tl_new:N     \l__tag_get_tmpc_tl
92  \tl_new:N     \l__tag_get_parent_tmpa_tl
93  \tl_new:N     \l__tag_get_parent_tmpb_tl
94  \str_new:N    \l__tag_tmpa_str
95  \prop_new:N   \l__tag_tmpa_prop
96  \seq_new:N    \l__tag_tmpa_seq
97  \seq_new:N    \l__tag_tmpb_seq
98  \clist_new:N  \l__tag_tmpa_clist
99  \int_new:N    \l__tag_tmpa_int
100 \box_new:N    \l__tag_tmpa_box
101 \box_new:N    \l__tag_tmpb_box
```

(*End of 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_property_mc_clist
\c_tag_property_struct_clist

```
102 \clist_const:Nn \c__tag_property_mc_clist     {tagabspage,tagmcabs,tagmcid}
103 \clist_const:Nn \c__tag_property_struct_clist {tagstruct,tagstructobj}
```

(*End of definition for* `\c__tag_property_mc_clist` *and* `\c__tag_property_struct_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.

```
104 \int_new:N   \l__tag_loglevel_int
```

(*End of 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 controls 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.

```
105 \bool_new:N \g__tag_active_space_bool
106 \bool_new:N \g__tag_active_mc_bool
107 \bool_new:N \g__tag_active_tree_bool
108 \bool_new:N \g__tag_active_struct_bool
109 \bool_new:N \g__tag_active_struct_dest_bool
110 \bool_gset_true:N \g__tag_active_struct_dest_bool
```

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

`\l__tag_active_mc_bool`
`\l__tag_active_struct_bool`
`\l__tag_active_socket_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.

```
111 \bool_new:N \l__tag_active_mc_bool
112 \bool_set_true:N \l__tag_active_mc_bool
113 \bool_new:N \l__tag_active_struct_bool
114 \bool_set_true:N \l__tag_active_struct_bool
115 \bool_new:N \l__tag_active_socket_bool
```

(*End of definition for* `\l__tag_active_mc_bool`, `\l__tag_active_struct_bool`, *and* `\l__tag_active_-socket_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.

```
116 \bool_new:N \g__tag_tagunmarked_bool
```

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

6 Variants of l3 commands

```
117 \prg_generate_conditional_variant:Nnn \pdf_object_if_exist:n {e}{T,F,TF}
118 \cs_generate_variant:Nn \pdf_object_ref:n {e}
119 \cs_generate_variant:Nn \pdfannot_dict_put:nnn {nne}
120 \cs_generate_variant:Nn \pdffile_embed_stream:nnn {nee,oee}
121 \cs_generate_variant:Nn \prop_gput:Nnn {Nee,Nen}
122 \cs_generate_variant:Nn \prop_put:Nnn  {Nee}
123 \cs_generate_variant:Nn \prop_item:Nn {No,Ne}
124 \cs_generate_variant:Nn \seq_set_split:Nnn{Nne}
125 \cs_generate_variant:Nn \str_set_convert:Nnnn {Nonn, Noon, Nnon }
126 \cs_generate_variant:Nn \clist_map_inline:nn {on}
```

10

7 Label and Reference commands

To ease transition to properties we setup internal definition. They can be replaced by the property definitions once that is released.

<code>__tag_property_new:nnnn</code>
<code>__tag_property_gset:nnnn</code>
<code>__tag_property_ref:nnn</code>

At first a command to define new properties

```
127    \cs_new_eq:NN \__tag_property_new:nnnn \property_new:nnnn
```

For the non-shipout code we need also the option to reset property

```
128    \cs_new_eq:NN \__tag_property_gset:nnnn \property_gset:nnnn
```

The command to reference while giving a local default.

```
129    \cs_new_eq:NN \__tag_property_ref:nnn \property_ref:nnn
130    \cs_new_eq:NN \__tag_property_ref:nn \property_ref:nn
```

The command to record

```
131    \cs_new_protected:Npn \__tag_property_record:nn #1#2
132      {
133        \@bsphack
134        \property_record:nn{#1}{#2}
135        \@esphack
136      }
137
```

And a few variants

```
138  \cs_generate_variant:Nn \__tag_property_ref:nnn {enn}
139  \cs_generate_variant:Nn \__tag_property_ref:nn {en}
140  \cs_generate_variant:Nn \__tag_property_record:nn {en,eV}
```

(*End of definition for* `\__tag_property_new:nnnn` *,* `\__tag_property_gset:nnnn` *, and* `\__tag_property_-ref:nnn` *.*)

<code>__tag_property_ref_lastpage:nn</code>

A command to retrieve the lastpage label, this will be adapted when there is a proper, kernel lastpage label.

```
141  \cs_new:Npn \__tag_property_ref_lastpage:nn #1 #2
142    {
143      \__tag_property_ref:nnn {@tag@LastPage}{#1}{#2}
144    }
```

(*End of definition for* `\__tag_property_ref_lastpage:nn` *.*)

8 Setup label attributes

<code>tagstruct</code>
<code>tagstructobj</code>
<code>tagabspage</code>
<code>tagmcabs</code>
<code>tagmcid</code>

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`.

```
145  \__tag_property_new:nnnn
146    { tagstruct } { now }
147    {0} { \int_use:N \c@g__tag_struct_abs_int }
148  \__tag_property_new:nnnn  { tagstructobj } { now }  {}
149    {
```

```
150     \pdf_object_if_exist:eT {__tag/struct/\int_use:N \c@g__tag_struct_abs_int}
151       {
152         \pdf_object_ref:e{__tag/struct/\int_use:N \c@g__tag_struct_abs_int}
153       }
154   }
155 \__tag_property_new:nnnn
156   { tagabspage } { shipout }
157   {0} { \int_use:N \g_shipout_readonly_int }
158 \__tag_property_new:nnnn  { tagmcabs } { now }
159   {0} { \int_use:N \c@g__tag_MCID_abs_int }
160
161 \flag_new:n { __tag/mcid }
162 \__tag_property_new:nnnn  {tagmcid } { shipout }
163   {0} { \flag_height:n { __tag/mcid } }
164
```

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

9   Commands to fill seq and prop

With most engines these are simply copies of the expl3 commands, but luatex will overwrite 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`

```
165 \cs_set_eq:NN \__tag_prop_new:N          \prop_new:N
166 \cs_set_eq:NN \__tag_seq_new:N           \seq_new:N
167 \cs_set_eq:NN \__tag_prop_gput:Nnn       \prop_gput:Nnn
168 \cs_set_eq:NN \__tag_seq_gput_right:Nn \seq_gput_right:Nn
169 \cs_set_eq:NN \__tag_seq_item:cn         \seq_item:cn
170 \cs_set_eq:NN \__tag_prop_item:cn        \prop_item:cn
171 \cs_set_eq:NN \__tag_seq_show:N          \seq_show:N
172 \cs_set_eq:NN \__tag_prop_show:N         \prop_show:N
173 % cnx temporary needed for latex-lab-graphic code
174 \cs_generate_variant:Nn \__tag_prop_gput:Nnn     { Nen , Nee, Nne , cnn, cen, cne, cno, cnx}
175 \cs_generate_variant:Nn \__tag_seq_gput_right:Nn { Ne  , No, cn, ce }
176 \cs_generate_variant:Nn \__tag_prop_new:N  { c }
177 \cs_generate_variant:Nn \__tag_seq_new:N   { c }
178 \cs_generate_variant:Nn \__tag_seq_show:N  { c }
179 \cs_generate_variant:Nn \__tag_prop_show:N  { c }
180 ⟨/package⟩
```

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

10   General tagging commands

`\tag_stop:`
`\tag_start:`
`\tag_stop:n`
`\tag_start:n`

We need commands to stop tagging in some places. They switch local booleans and also stop the counting of paragraphs. The commands keep track of the nesting with a local counter. Tagging only is only restarted at the outer level, if the current level is 1. The commands with argument allow to give a label. This is only used in debugging messages to allow to follow the nesting.

When stop/start pairs are nested we do not want the inner start command to restart tagging. To control this we use a local int: The stop command will increase it. The starting will decrease it and only restart tagging, if it is zero. This will replace the label version.

\l__tag_tag_stop_int

```
181 ⟨∗package | debug⟩
182 ⟨package⟩\int_new:N \l__tag_tag_stop_int

183 \cs_set_protected:Npn \tag_stop:
184   {
185 ⟨debug⟩      \msg_note:nnx {tag / debug }{tag-stop}{ \int_use:N \l__tag_tag_stop_int }
186     \int_incr:N \l__tag_tag_stop_int
187     \bool_set_false:N \l__tag_active_struct_bool
188     \bool_set_false:N \l__tag_active_mc_bool
189     \bool_set_false:N \l__tag_active_socket_bool
190     \__tag_stop_para_ints:
191   }
192 \cs_set_protected:Npn \tag_start:
193   {
194     \int_if_zero:nF { \l__tag_tag_stop_int } { \int_decr:N \l__tag_tag_stop_int }
195     \int_if_zero:nT { \l__tag_tag_stop_int }
196       {
197         \bool_set_true:N \l__tag_active_struct_bool
198         \bool_set_true:N \l__tag_active_mc_bool
199         \bool_set_true:N \l__tag_active_socket_bool
200         \__tag_start_para_ints:
201       }
202 ⟨debug⟩      \msg_note:nnx {tag / debug }{tag-start}{ \int_use:N \l__tag_tag_stop_int }
203   }
204 \cs_set_eq:NN\tagstop\tag_stop:
205 \cs_set_eq:NN\tagstart\tag_start:
206 \cs_set_protected:Npn \tag_stop:n #1
207   {
208 ⟨debug⟩      \msg_note:nnxx {tag / debug }{tag-stop}{ \int_use:N \l__tag_tag_stop_int }{#1}
209     \int_incr:N \l__tag_tag_stop_int
210     \bool_set_false:N \l__tag_active_struct_bool
211     \bool_set_false:N \l__tag_active_mc_bool
212     \bool_set_false:N \l__tag_active_socket_bool
213     \__tag_stop_para_ints:
214   }
215 \cs_set_protected:Npn \tag_start:n #1
216   {
217     \int_if_zero:nF { \l__tag_tag_stop_int } { \int_decr:N \l__tag_tag_stop_int }
218     \int_if_zero:nT { \l__tag_tag_stop_int }
219       {
220         \bool_set_true:N \l__tag_active_struct_bool
221         \bool_set_true:N \l__tag_active_mc_bool
222         \bool_set_true:N \l__tag_active_socket_bool
223         \__tag_start_para_ints:
224       }
225 ⟨debug⟩      \msg_note:nnxx {tag / debug }{tag-start}{ \int_use:N \l__tag_tag_stop_int }{#1}
226   }
227 ⟨/package | debug⟩
228 ⟨∗base⟩
229 \cs_new_protected:Npn \tag_stop:{}
```

```
230 \cs_new_protected:Npn \tag_start:{}
231 \cs_new_protected:Npn \tagstop{}
232 \cs_new_protected:Npn \tagstart{}
233 \cs_new_protected:Npn \tag_stop:n  #1 {}
234 \cs_new_protected:Npn \tag_start:n #1 {}
235 ⟨/base⟩
```

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

11 Keys for tagpdfsetup

TODO: the log-levels must be sorted

<div style="text-align: right"><code>activate-space␣(setup-key)</code>
<code>activate-mc␣(setup-key)</code>
<code>activate-tree␣(setup-key)</code>
<code>activate-struct␣(setup-key)</code>
<code>activate-all␣(setup-key)</code>
<code>no-struct-dest␣(setup-key)</code></div>

Keys to (globally) activate tagging. `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. `no-struct-dest` allows to suppress structure destinations.

```
236 ⟨*package⟩
237 \keys_define:nn { __tag / setup }
238   {
239     activate-space   .bool_gset:N = \g__tag_active_space_bool,
240     activate-mc      .bool_gset:N = \g__tag_active_mc_bool,
241     activate-tree    .bool_gset:N = \g__tag_active_tree_bool,
242     activate-struct  .bool_gset:N = \g__tag_active_struct_bool,
243     activate-all     .meta:n =
244       {activate-mc={#1},activate-tree={#1},activate-struct={#1}},
245     activate-all   .default:n = true,
246     no-struct-dest .bool_gset_inverse:N = \g__tag_active_struct_dest_bool,
247
```

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

<div style="text-align: right"><code>log␣(setup-key)</code></div>

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

```
248     log                .choice:,
249     log / none         .code:n = {\int_set:Nn \l__tag_loglevel_int { 0 }},
250     log / v            .code:n =
251       {
252         \int_set:Nn \l__tag_loglevel_int { 1 }
253         \cs_set_protected:Nn \__tag_check_typeout_v:n { \iow_term:e {##1} }
254       },
255     log / vv           .code:n = {\int_set:Nn \l__tag_loglevel_int { 2 }},
256     log / vvv          .code:n = {\int_set:Nn \l__tag_loglevel_int { 3 }},
257     log / all          .code:n = {\int_set:Nn \l__tag_loglevel_int { 10 }},
```

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

<div style="text-align: right"><code>tagunmarked␣(setup-key)</code></div>

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

```
258     tagunmarked        .bool_gset:N = \g__tag_tagunmarked_bool,
259     tagunmarked        .initial:n = true,
```

14

(End of 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.

```
260     tabsorder        .choice:,
261     tabsorder / row        .code:n =
262       \pdfmanagement_add:nnn { Page } {Tabs}{/R},
263     tabsorder / column     .code:n =
264       \pdfmanagement_add:nnn { Page } {Tabs}{/C},
265     tabsorder / structure .code:n =
266       \pdfmanagement_add:nnn { Page } {Tabs}{/S},
267     tabsorder / none       .code:n =
268       \pdfmanagement_remove:nn {Page} {Tabs},
269     tabsorder        .initial:n = structure,
270     uncompress       .code:n = { \pdf_uncompress:  },
271   }
```

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

12 loading of engine/more dependent code

```
272 \sys_if_engine_luatex:T
273   {
274     \file_input:n {tagpdf-luatex.def}
275   }
276 ⟨/package⟩
277 ⟨∗mcloading⟩
278 \bool_if:NTF \g__tag_mode_lua_bool
279   {
280    \RequirePackage {tagpdf-mc-code-lua}
281   }
282   {
283    \RequirePackage {tagpdf-mc-code-generic} %
284   }
285 ⟨/mcloading⟩
286 ⟨∗debug⟩
287 \bool_if:NTF \g__tag_mode_lua_bool
288   {
289    \RequirePackage {tagpdf-debug-lua}
290   }
291   {
292    \RequirePackage {tagpdf-debug-generic} %
293   }
294 ⟨/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`.

`\tag_if_box_tagged_p:N` ⋆
`\tag_if_box_tagged:N`*TF* ⋆

`\tag_if_box_tagged:N{`⟨*box*⟩`}`

This tests if a box contains tagging commands. It relies currently on that the code that saved the box correctly set the command `\l_tag_box_\int_use:N␣#1_tl` to a positive value. The LaTeX commands will do that automatically at some time but it is in the responsability of the user to ensure that when using low-level code. If the internal command doesn't exist the box is assumed to be untagged.

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	
\ShowTaggingdebug/structures = num	log+termn	debug mode only

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
`\@@_struct_write_obj:n`	`struct-orphan`	warning
`\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	

message	log-level	remark
`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	
`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-unknown`
`struct-no-objnum`
`struct-orphan`
`struct-faulty-nesting`
`struct-missing-tag`
`struct-used-twice`
`struct-label-unknown`
`struct-show-closing`
	Various messages related to structure. Check the definition in the code for their meaning and the arguments they take.

`tree-struct-still-open`	Message issued at the end of the compilation if there are (beside Root) other open structures on the stack.
`show-struct` `show-kids`	These two messages are used in debug mode to show the current structures in the log and terminal.
`attr-unknown`	Message if an attribute i sunknown.

`role-missing`
`role-unknown`
`role-unknown-tag`
`role-tag`
`new-tag`
`role-parent-child`
`role-remapping`
	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} {2024-02-02} {0.98u}
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 of 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 of 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 of 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 of 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 of definition for* `mc-not-open`*. This function is documented on page 18.*)

mc-pushed Informational messages about mc-pushing.
mc-popped
```
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 of 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 of 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 of definition for* struct-unknown. *This function is documented on page 19.*)

struct-no-objnum   Should not happen . . .

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

(*End of definition for* struct-no-objnum. *This function is documented on page 19.*)

struct-orphan   This indicates that there is a structure which has kids but no parent. This can happen if a structure is stashed but then not used.

```
25 \msg_new:nnn { tag } {struct-orphan}
26    {
27      Structure~#1~has~#2~kids~but~no~parent.\\
28      It~is~turned~into~an~artifact.\\
29      Did~you~stashed~a~structure~and~then~didn't~use~it?
30    }
31
```

(*End of definition for* struct-orphan. *This function is documented on page 19.*)

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

```
32 \msg_new:nnn { tag }
33    {struct-faulty-nesting}
34    { there~is~no~open~structure~on~the~stack }
```

(*End of definition for* struct-faulty-nesting. *This function is documented on page 19.*)

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

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

(*End of definition for* struct-missing-tag. *This function is documented on page 19.*)

struct-used-twice

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

(*End of definition for* struct-used-twice. *This function is documented on page 19.*)

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

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

(*End of definition for* struct-label-unknown. *This function is documented on page 19.*)

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

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

(*End of definition for* struct-show-closing. *This function is documented on page 19.*)

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

```
42 \msg_new:nnn { tag } {tree-struct-still-open}
43   {
44     There~are~still~open~structures~on~the~stack!\\
45     The~stack~contains~\seq_use:Nn\g__tag_struct_tag_stack_seq{,}.\\
46     The~structures~are~automatically~closed,\\
47     but~their~nesting~can~be~wrong.
48   }
49 ⟨/package⟩
```

(*End of definition for* tree-struct-still-open. *This function is documented on page 19.*)

The following messages are only needed in debug mode.

show-struct This two messages are used to show the current structures in the log and terminal.
show-kids

```
50 ⟨*debug⟩
51 \msg_new:nnn { tag/debug } { show-struct }
52   {
53     ========================\\
54     The~structure~#1~
55     \tl_if_empty:nTF {#2}
56       { is~empty \\>~ . }
57       { contains: #2  }
58     \\
59   }
60 \msg_new:nnn { tag/debug } { show-kids }
61   {
62     The~structure~has~the~following~kids:
63     \tl_if_empty:nTF {#2}
64       { \\>~ NONE }
65       { #2  }
66     \\
67     ========================
68   }
69 ⟨/debug⟩
```

(*End of definition for* show-struct *and* show-kids. *These functions are documented on page 19.*)

3.3   Attributes

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

attr-unknown

```
70 ⟨*package⟩
71 \msg_new:nnn { tag } {attr-unknown}  { attribute~#1~is~unknown}
```

(*End of definition for* attr-unknown. *This function is documented on page 19.*)

3.4   Roles

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

```
72 \msg_new:nnn { tag } {role-missing}    { tag~#1~has~no~role~assigned  }
73 \msg_new:nnn { tag } {role-unknown}    { role~#1~is~not~known  }
74 \msg_new:nnn { tag } {role-unknown-tag} { tag~#1~is~not~known  }
```

(*End of definition for* `role-missing`*,* `role-unknown`*, and* `role-unknown-tag`*. These functions are documented on page* *19.*)

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

```
75 \msg_new:nnn { tag } {role-parent-child}
76   { Parent-Child~'#1'~-->~'#2'.\\Relation~is~#3~\msg_line_context:}
```

(*End of definition for* `role-parent-child`*. This function is documented on page* *19.*)

role-remapping | This is info and warning message about role-remapping

```
77 \msg_new:nnn { tag } {role-remapping}
78   { remapping~tag~to~#1 }
```

(*End of definition for* `role-remapping`*. This function is documented on page* *19.*)

role-tag
new-tag | Info messages.

```
79 \msg_new:nnn { tag } {role-tag}          { mapping~tag~#1~to~role~#2  }
80 \msg_new:nnn { tag } {new-tag}           { adding~new~tag~#1 }
81 \msg_new:nnn { tag } {read-namespace}    { reading~namespace~definitions~tagpdf-ns-
   #1.def }
82 \msg_new:nnn { tag } {namespace-missing}{ namespace~definitions~tagpdf-ns-#1.def~not~found }
83 \msg_new:nnn { tag } {namespace-unknown}{ namespace~#1~is~not~declared }
```

(*End of definition for* `role-tag` *and* `new-tag`*. These functions are documented on page* *19.*)

3.5 Miscellaneous

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

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

(*End of 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.

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

(*End of 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.

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

(*End of 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.

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

(*End of 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`.

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

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

5 User conditionals

`\tag_if_active_p:`
`\tag_if_active:TF`
This tests if 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.

```
94 ⟨*base⟩
95 \prg_new_conditional:Npnn \tag_if_active: { p , T , TF, F }
96   { \prg_return_false: }
97 ⟨/base⟩
98 ⟨*package⟩
99 \prg_set_conditional:Npnn \tag_if_active: { p , T , TF, F }
100   {
101      \bool_lazy_all:nTF
102        {
103          {\g__tag_active_struct_bool}
104          {\g__tag_active_mc_bool}
105          {\g__tag_active_tree_bool}
106          {\l__tag_active_struct_bool}
107          {\l__tag_active_mc_bool}
108        }
109        {
110          \prg_return_true:
111        }
112        {
113          \prg_return_false:
114        }
115   }
116 ⟨/package⟩
```

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

`\tag_if_box_tagged_p:N`
`\tag_if_box_tagged:NTF`
This tests if a box contains tagging commands. It relies on that the code that saved the box correctly set `\l_tag_box_<box␣number>_tl` to a positive value. The LaTeX commands will do that automatically at some time but it is in the responsability of the user to ensure that when using low-level code. If the internal command doesn't exist the box is assumed to be untagged.

```
117 ⟨*base⟩
118 \prg_new_conditional:Npnn \tag_if_box_tagged:N #1 {p,T,F,TF}
119     {
120        \tl_if_exist:cTF {l_tag_box_\int_use:N #1_tl}
121        {
122           \int_compare:nNnTF {0\tl_use:c{l_tag_box_\int_use:N #1_tl}}>{0}
123           { \prg_return_true:  }
124           { \prg_return_false: }
125        }
```

24

```
126           {
127              \prg_return_false:
128              % warning??
129           }
130        }
131 ⟨/base⟩
```

(*End of definition for* `\tag_if_box_tagged:NTF`. *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

`\__tag_check_if_active_mc:TF`
`\__tag_check_if_active_struct:TF`

This checks if mc are active.

```
132 ⟨*package⟩
133 \prg_new_conditional:Npnn \__tag_check_if_active_mc: {T,F,TF}
134    {
135      \bool_lazy_and:nnTF { \g__tag_active_mc_bool } { \l__tag_active_mc_bool }
136        {
137           \prg_return_true:
138        }
139        {
140           \prg_return_false:
141        }
142    }
143 \prg_new_conditional:Npnn \__tag_check_if_active_struct: {T,F,TF}
144    {
145      \bool_lazy_and:nnTF { \g__tag_active_struct_bool } { \l__tag_active_struct_bool }
146        {
147           \prg_return_true:
148        }
149        {
150           \prg_return_false:
151        }
152    }
```

(*End of 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.

```
153 \cs_new_protected:Npn \__tag_check_structure_has_tag:n #1 %#1 struct num
154    {
155      \prop_if_in:cnF { g__tag_struct_#1_prop }
156        {S}
157        {
158           \msg_error:nn { tag } {struct-missing-tag}
159        }
160    }
```

25

(End of 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.

```
161 \cs_new_protected:Npn \__tag_check_structure_tag:N #1
162   {
163     \prop_if_in:NoF \g__tag_role_tags_NS_prop {#1}
164       {
165         \msg_warning:nne { tag } {role-unknown-tag} {#1}
166       }
167   }
```

(End of 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.

```
168 \cs_new_protected:Npn \__tag_check_info_closing_struct:n #1 %#1 struct num
169   {
170     \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
171       {
172         \msg_info:nnn { tag } {struct-show-closing} {#1}
173       }
174   }
175
176 \cs_generate_variant:Nn \__tag_check_info_closing_struct:n {o,e}
```

(End of 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.

```
177 \cs_new_protected:Npn \__tag_check_no_open_struct:
178   {
179     \msg_error:nn { tag } {struct-faulty-nesting}
180   }
```

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

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

```
181 \cs_new_protected:Npn \__tag_check_struct_used:n #1 %#1 label
182   {
183     \prop_get:cnNT
184       {g__tag_struct_\__tag_property_ref:enn{tagpdfstruct-#1}{tagstruct}{unknown}_prop}
185       {P}
186       \l__tag_tmpa_tl
187       {
188         \msg_warning:nnn { tag } {struct-used-twice} {#1}
189       }
190   }
```

(End of 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.

```
191 \cs_new_protected:Npn \__tag_check_add_tag_role:nn #1 #2 %#1 tag, #2 role
192   {
193     \tl_if_empty:nTF {#2}
194       {
195         \msg_error:nnn { tag } {role-missing} {#1}
196       }
197       {
198         \prop_get:NnNTF \g__tag_role_tags_NS_prop {#2} \l_tmpa_tl
199           {
200             \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
201               {
202                 \msg_info:nnnn { tag } {role-tag} {#1} {#2}
203               }
204           }
205           {
206             \msg_error:nnn { tag } {role-unknown} {#2}
207           }
208       }
209   }
```

Similar with a namespace

```
210 \cs_new_protected:Npn \__tag_check_add_tag_role:nnn #1 #2 #3 %#1 tag/NS, #2 role #3 namespace
211   {
212     \tl_if_empty:nTF {#2}
213       {
214         \msg_error:nnn { tag } {role-missing} {#1}
215       }
216       {
217         \prop_get:cnNTF { g__tag_role_NS_#3_prop } {#2} \l_tmpa_tl
218           {
219             \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
220               {
221                 \msg_info:nnnn { tag } {role-tag} {#1} {#2/#3}
222               }
223           }
224           {
225             \msg_error:nnn { tag } {role-unknown} {#2/#3}
226           }
227       }
228   }
```

(*End of 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).

```
229 \cs_new_protected:Npn \__tag_check_mc_if_nested:
230   {
231     \__tag_mc_if_in:T
232       {
```

```
233          \msg_warning:nne { tag } {mc-nested} { \__tag_get_mc_abs_cnt: }
234        }
235    }
236
237  \cs_new_protected:Npn \__tag_check_mc_if_open:
238    {
239      \__tag_mc_if_in:F
240        {
241          \msg_warning:nne { tag } {mc-not-open} { \__tag_get_mc_abs_cnt: }
242        }
243    }
```

(*End of 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.

```
244  \cs_new_protected:Npn \__tag_check_mc_pushed_popped:nn #1 #2
245    {
246      \int_compare:nNnT
247        { \l__tag_loglevel_int } ={ 2 }
248        { \msg_info:nne {tag}{mc-#1}{#2} }
249      \int_compare:nNnT
250        { \l__tag_loglevel_int } > { 2 }
251        {
252          \msg_info:nne {tag}{mc-#1}{#2}
253          \seq_log:N \g__tag_mc_stack_seq
254        }
255    }
```

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

`\__tag_check_mc_tag:N` This checks if the mc has a (known) tag.

```
256  \cs_new_protected:Npn \__tag_check_mc_tag:N #1  %#1 is var with a tag name in it
257    {
258      \tl_if_empty:NT #1
259        {
260          \msg_error:nne { tag } {mc-tag-missing} { \__tag_get_mc_abs_cnt: }
261        }
262      \prop_if_in:NoF \g__tag_role_tags_NS_prop {#1}
263        {
264          \msg_warning:nne { tag } {role-unknown-tag} {#1}
265        }
266    }
```

(*End of 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??

28

```
267 \cs_new_protected:Npn \__tag_check_init_mc_used:
268   {
269     \intarray_new:Nn \g__tag_check_mc_used_intarray { 65536 }
270     \cs_gset_eq:NN \__tag_check_init_mc_used: \prg_do_nothing:
271   }
```

(*End of 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.

```
272 \cs_new_protected:Npn \__tag_check_mc_used:n #1 %#1 mcid abscnt
273   {
274     \int_compare:nNnT {\l__tag_loglevel_int} > { 2 }
275       {
276         \__tag_check_init_mc_used:
277         \intarray_gset:Nnn \g__tag_check_mc_used_intarray
278           {#1}
279           { \intarray_item:Nn \g__tag_check_mc_used_intarray {#1} + 1 }
280         \int_compare:nNnT
281           {
282             \intarray_item:Nn \g__tag_check_mc_used_intarray {#1}
283           }
284           >
285           { 1 }
286           {
287             \msg_warning:nnn { tag } {mc-used-twice} {#1}
288           }
289       }
290   }
```

(*End of definition for* __tag_check_mc_used:n*.)

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

```
291 \cs_new_protected:Npn \__tag_check_show_MCID_by_page:
292   {
293     \tl_set:Ne \l__tag_tmpa_tl
294       {
295         \__tag_property_ref_lastpage:nn
296           {abspage}
297           {-1}
298       }
299     \int_step_inline:nnnn {1}{1}
300       {
301         \l__tag_tmpa_tl
302       }
303       {
304         \seq_clear:N \l_tmpa_seq
305         \int_step_inline:nnnn
306           {1}
307           {1}
308           {
309             \__tag_property_ref_lastpage:nn
310               {tagmcabs}
311               {-1}
312           }
313           {
```

```
314            \int_compare:nT
315              {
316                \__tag_property_ref:enn
317                  {mcid-####1}
318                  {tagabspage}
319                  {-1}
320                =
321                ##1
322              }
323              {
324                \seq_gput_right:Ne \l_tmpa_seq
325                  {
326                    Page##1-####1-
327                    \__tag_property_ref:enn
328                      {mcid-####1}
329                      {tagmcid}
330                      {-1}
331                  }
332              }
333            }
334          \seq_show:N \l_tmpa_seq
335        }
336    }
```

(*End of 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.

```
337  \prg_new_conditional:Npnn \__tag_check_if_mc_in_galley: { T,F,TF }
338    {
339      \tl_if_eq:NNTF \l__tag_mc_firstmarks_seq \l__tag_mc_botmarks_seq
340      { \prg_return_false: }
341      { \prg_return_true: }
342    }
```

(*End of 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.

```
343  \prg_new_conditional:Npnn \__tag_check_if_mc_tmb_missing: { T,F,TF }
344    {
345      \bool_if:nTF
```

```
346     {
347       \str_if_eq_p:ee {\seq_item:Nn \l__tag_mc_firstmarks_seq {1}}{e-}
348       ||
349       \str_if_eq_p:ee {\seq_item:Nn \l__tag_mc_firstmarks_seq {1}}{b+}
350     }
351     { \prg_return_true: }
352     { \prg_return_false: }
353   }
```

(*End of definition for* `\__tag_check_if_mc_tmb_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.

```
354 \prg_new_conditional:Npnn \__tag_check_if_mc_tme_missing: { T,F,TF }
355   {
356     \str_if_eq:eeTF {\seq_item:Nn \l__tag_mc_botmarks_seq {1}}{b+}
357     { \prg_return_true: }
358     { \prg_return_false: }
359   }
```

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

```
360 ⟨/package⟩
```

```
361 ⟨∗debug⟩
```

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

```
362 \msg_new:nnn { tag / debug } {mc-begin} { MC~begin~#1~with~options:~\tl_to_str:n{#2}~[\msg_lin
363 \msg_new:nnn { tag / debug } {mc-end}   { MC~end~#1~[\msg_line_context:] }
364
365 \cs_new_protected:Npn \__tag_debug_mc_begin_insert:n #1
366   {
367     \int_compare:nNnT { \l__tag_loglevel_int } > {0}
368       {
369         \msg_note:nnnn { tag / debug } {mc-begin} {inserted} { #1 }
370       }
371   }
372 \cs_new_protected:Npn \__tag_debug_mc_begin_ignore:n #1
373   {
374     \int_compare:nNnT { \l__tag_loglevel_int } > {0}
375       {
376         \msg_note:nnnn { tag / debug } {mc-begin } {ignored} { #1 }
377       }
378   }
379 \cs_new_protected:Npn \__tag_debug_mc_end_insert:
380   {
381     \int_compare:nNnT { \l__tag_loglevel_int } > {0}
382       {
383         \msg_note:nnn { tag / debug } {mc-end} {inserted}
384       }
385   }
386 \cs_new_protected:Npn \__tag_debug_mc_end_ignore:
387   {
388     \int_compare:nNnT { \l__tag_loglevel_int } > {0}
```

```
389        {
390            \msg_note:nnn { tag / debug } {mc-end } {ignored}
391        }
392    }
```
And now something for the structures
```
393  \msg_new:nnn { tag / debug } {struct-begin}
394    {
395      Struct~\tag_get:n{struct_num}~begin~#1~with~options:~\tl_to_str:n{#2}~\\[\msg_line_context
396    }
397  \msg_new:nnn { tag / debug } {struct-end}
398    {
399      Struct~end~#1~[\msg_line_context:]
400    }
401  \msg_new:nnn { tag / debug } {struct-end-wrong}
402    {
403      Struct~end~'#1'~doesn't~fit~start~'#2'~[\msg_line_context:]
404    }
405
406  \cs_new_protected:Npn \__tag_debug_struct_begin_insert:n #1
407    {
408      \int_compare:nNnT { \l__tag_loglevel_int } > {0}
409        {
410            \msg_note:nnnn { tag / debug } {struct-begin} {inserted} { #1 }
411            \seq_log:N \g__tag_struct_tag_stack_seq
412        }
413    }
414  \cs_new_protected:Npn \__tag_debug_struct_begin_ignore:n #1
415    {
416      \int_compare:nNnT { \l__tag_loglevel_int } > {0}
417        {
418            \msg_note:nnnn { tag / debug } {struct-begin } {ignored} { #1 }
419        }
420    }
421  \cs_new_protected:Npn \__tag_debug_struct_end_insert:
422    {
423      \int_compare:nNnT { \l__tag_loglevel_int } > {0}
424        {
425            \msg_note:nnn { tag / debug } {struct-end} {inserted}
426            \seq_log:N \g__tag_struct_tag_stack_seq
427        }
428    }
429  \cs_new_protected:Npn \__tag_debug_struct_end_ignore:
430    {
431      \int_compare:nNnT { \l__tag_loglevel_int } > {0}
432        {
433            \msg_note:nnn { tag / debug } {struct-end } {ignored}
434        }
435    }
436  \cs_new_protected:Npn \__tag_debug_struct_end_check:n #1
437    {
438      \int_compare:nNnT { \l__tag_loglevel_int } > {0}
439        {
440          \seq_get:NNT \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
441            {
```

```
442        \str_if_eq:eeF
443          {#1}
444          {\exp_last_unbraced:NV\use_i:nn \l__tag_tmpa_tl}
445          {
446            \msg_warning:nnee { tag/debug }{ struct-end-wrong }
447              {#1}
448              {\exp_last_unbraced:NV\use_i:nn \l__tag_tmpa_tl}
449          }
450      }
451    }
452  }
```

This tracks tag stop and start. The tag-stop message should go before the int is increased. The tag-start message after the int is decreased.

```
453 \msg_new:nnn { tag / debug } {tag-stop}
454   {
455     \int_if_zero:nTF
456       {#1}
457       {Tagging~stopped}
458       {Tagging~(not)~stopped~(already~inactive)}\\
459     level:~#1~==>~\int_eval:n{#1+1}\tl_if_empty:nF{#2}{,~label:~#2}~[\msg_line_context:]
460   }
461 \msg_new:nnn { tag / debug } {tag-start}
462   {
463     \int_if_zero:nTF
464       {#1}
465       {Tagging~restarted}
466       {Tagging~(not)~restarted}\\
467     level:~\int_eval:n{#1+1}~==>~#1\tl_if_empty:nF{#2}{,~label:~#2}~[\msg_line_context:]
468   }
```

469 ⟨/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 variety 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.

debug/structures␣(show-key)   debug/structures = ⟨structure number⟩

This key is available only if the tagpdf-debug package is loaded and shows all structures starting with the one with the number given by the key.

5 Extension commands

The following commands and code parts are not core commands 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}%
```

36

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 Socket support

\tag_socket_use:n   `\tag_socket_use:n {⟨socket name⟩}`
\tag_socket_use:nn `\tag_socket_use:nn {⟨socket name⟩} {⟨socket argument⟩}`
\UseTaggingSocket   `\UseTaggingSocket {⟨socket name⟩}`
                    `\UseTaggingSocket {⟨socket name⟩} {⟨socket argument⟩}`

The next LaTeX will use special sockets for the tagging.

These sockets will use names starting with `tagsupport/`. Usually, these sockets have exactly two plugs defined: `noop` (when no tagging is requested or tagging is not wanted for some reason) and a second plug that enables the tagging. There may be more, e.g., tagging with special debugging, etc., but right now it is usually just on or off.

Given that we sometimes have to suspend tagging, it would be fairly inefficient to put different plugs into these sockets whenever that happens. We therefore offer `\UseTaggingSocket` which is like `\UseSocket` except that the socket name is specified without `tagsupport/`, i.e.,

$$\texttt{\textbackslash UseTaggingSocket\{foo\}} \rightarrow \texttt{\textbackslash UseSocket\{tagsupport/foo\}}$$

Beside being slightly shorter, the big advantage is that this way we can change `\UseTaggingSocket` to do nothing by switching a boolean instead of changing the plugs of the tagging support sockets back and forth.

It is possible to use the tagging support sockets with `\UseSocket` directly, but in this case the socket remains active if e.g. `\SuspendTagging` is in force. There may be reasons for doing that but in general we expect to always use `\UseTaggingSocket`.

The L3 programming layer versions `\tag_socket_use:n` and `\tag_socket_use:nn` are slightly more efficient than `\UseTaggingSocket` because they do not have to determine how many arguments the socket takes when disabling it.

7 User commands and extensions of document commands

```
1 ⟨@@=tag⟩
2 ⟨*header⟩
3 \ProvidesExplPackage {tagpdf-user} {2024-02-02} {0.98u}
4   {tagpdf - user commands}
5 ⟨/header⟩
```

8 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 of definition for* `\tagpdfsetup`. *This function is documented on page 34.*)

\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 of definition for* `\tag_tool:n` *and* `\tagtool`. *These functions are documented on page 34.*)

9 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 {  }
```

38

```
30    {
31      \tag_mc_end:
32    }
33
34  \NewDocumentCommand \tagmcuse { m }
35    {
36      \tag_mc_use:n {#1}
37    }
38  ⟨/base⟩
```

(*End of definition for* \tagmcbegin *,* \tagmcend *, and* \tagmcuse*. These functions are documented on page 34.*)

\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 of definition for* \tagmcifinTF*. This function is documented on page 34.*)

10   Commands for the structure

\tagstructbegin   These are structure related user commands. There are direct wrapper around the expl3
\tagstructend   variants.
\tagstructuse
```
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 of definition for* \tagstructbegin *,* \tagstructend *, and* \tagstructuse*. These functions are documented on page 35.*)

11 Socket support

Until we can be sure that the kernel defines the commands we provide them before redefining them:

```
61 ⟨∗base⟩
62 \providecommand\tag_socket_use:n[1]{}
63 \providecommand\tag_socket_use:nn[2]{}
64 \providecommand\UseTaggingSocket[1]{}
65 ⟨/base⟩
```

`\tag_socket_use:n`
`\tag_socket_use:nn`
`\UseTaggingSocket`

```
66 ⟨∗package⟩
67 \cs_set_protected:Npn \tag_socket_use:n #1
68   {
69     \bool_if:NT \l__tag_active_socket_bool
70       { \UseSocket {tagsupport/#1} }
71   }
72 \cs_set_protected:Npn \tag_socket_use:nn #1#2
73   {
74     \bool_if:NT \l__tag_active_socket_bool
75       { \UseSocket {tagsupport/#1} {#2} }
76   }
77 \cs_set_protected:Npn \UseTaggingSocket #1
78   {
79     \bool_if:NTF \l__tag_active_socket_bool
80       { \UseSocket{tagsupport/#1} }
81       {
82         \int_case:nnF
83           { \int_use:c { c__socket_tagsupport/#1_args_int } }
84           {
85             0 \prg_do_nothing:
86             1 \use_none:n
87             2 \use_none:nn
```

We do not expect tagging sockets with more than one or two arguments, so for now we only provide those.

```
88           }
89           \ERRORusetaggingsocket
90       }
91   }
92 ⟨/package⟩
```

(*End of definition for* `\tag_socket_use:n`*,* `\tag_socket_use:nn`*, and* `\UseTaggingSocket`*. These functions are documented on page 37.*)

12 Debugging

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

```
93 ⟨∗package⟩
94 \NewDocumentCommand\ShowTagging { m }
95   {
```

```
96      \keys_set:nn { __tag / show }{ #1}
97
98    }
```

(*End of definition for* \ShowTagging*. This function is documented on page 35.*)

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.

```
99  \keys_define:nn { __tag / show }
100   {
101     mc-data .code:n =
102       {
103         \sys_if_engine_luatex:T
104           {
105             \lua_now:e{ltx.__tag.trace.show_all_mc_data(#1,\__tag_get_mc_abs_cnt:,0)}
106           }
107       }
108     ,mc-data .default:n = 1
109   }
110
```

(*End of definition for* mc-data (show-key)*. This function is documented on page 35.*)

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

```
111  \keys_define:nn { __tag / show }
112   { mc-current .code:n =
113       {
114         \bool_if:NTF \g__tag_mode_lua_bool
115           {
116             \sys_if_engine_luatex:T
117               {
118                 \int_compare:nNnTF
119                   { -2147483647 }
120                   =
121                   {
122                     \lua_now:e
123                       {
124                         tex.print
125                           (tex.getattribute
126                             (luatexbase.attributes.g__tag_mc_cnt_attr))
127                       }
128                   }
129                   {
130                     \lua_now:e
131                       {
132                         ltx.__tag.trace.log
133                           (
134                             "mc-current:~no~MC~open,~current~abscnt
135                              =\__tag_get_mc_abs_cnt:"
136                             ,0
137                           )
138                         texio.write_nl("")
139                       }
```

41

```
140                              }
141                              {
142                                \lua_now:e
143                                  {
144                                    ltx.__tag.trace.log
145                                      (
146                                        "mc-current:~abscnt=\__tag_get_mc_abs_cnt:=="
147                                        ..
148                                        tex.getattribute(luatexbase.attributes.g__tag_mc_cnt_attr)
149                                        ..
150                                        "~=>tag="
151                                        ..
152                                        tostring
153                                          (ltx.__tag.func.get_tag_from
154                                            (tex.getattribute
155                                              (luatexbase.attributes.g__tag_mc_type_attr)))
156                                        ..
157                                        "="
158                                        ..
159                                        tex.getattribute
160                                         (luatexbase.attributes.g__tag_mc_type_attr)
161                                        ,0
162                                      )
163                                    texio.write_nl("")
164                                  }
165                              }
166                          }
167                      }
168                      {
169                       \msg_note:nn{ tag }{ mc-current }
170                      }
171                  }
172    }
```

(End of definition for `mc-current` *(show-key). This function is documented on page 35.)*

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).

```
173  \keys_define:nn { __tag / show }
174    {
175      mc-marks .choice: ,
176      mc-marks / show .code:n =
177        {
178          \__tag_mc_get_marks:
179          \__tag_check_if_mc_in_galley:TF
180            {
181             \iow_term:n {Marks~from~this~page:~}
182            }
183            {
184              \iow_term:n {Marks~from~a~previous~page:~}
185            }
186          \seq_show:N \l__tag_mc_firstmarks_seq
187          \seq_show:N \l__tag_mc_botmarks_seq
188          \__tag_check_if_mc_tmb_missing:T
```

```
189        {
190            \iow_term:n {BDC~missing~on~this~page!}
191        }
192        \__tag_check_if_mc_tme_missing:T
193        {
194            \iow_term:n {EMC~missing~on~this~page!}
195        }
196      },
197    mc-marks / use .code:n =
198      {
199        \__tag_mc_get_marks:
200        \__tag_check_if_mc_in_galley:TF
201        { Marks~from~this~page:~}
202        { Marks~from~a~previous~page:~}
203        \seq_use:Nn \l__tag_mc_firstmarks_seq {,~}\quad
204        \seq_use:Nn \l__tag_mc_botmarks_seq {,~}\quad
205        \__tag_check_if_mc_tmb_missing:T
206        {
207          BDC~missing~
208        }
209        \__tag_check_if_mc_tme_missing:T
210        {
211          EMC~missing
212        }
213      },
214    mc-marks .default:n = show
215    }
```

(End of definition for `mc-marks` *(show-key). This function is documented on page 35.)*

struct-stack␣(show-key)

```
216 \keys_define:nn { __tag / show }
217   {
218      struct-stack .choice:
219     ,struct-stack / log .code:n = \seq_log:N \g__tag_struct_tag_stack_seq
220     ,struct-stack / show .code:n = \seq_show:N \g__tag_struct_tag_stack_seq
221     ,struct-stack .default:n = show
222   }
223 ⟨/package⟩
```

(End of definition for `struct-stack` *(show-key). This function is documented on page 35.)*

debug/structures␣(show-key) The following key is available only if the tagpdf-debug package is loaded and shows all structures starting with the one with the number given by the key.

```
224 ⟨*debug⟩
225 \keys_define:nn { __tag / show }
226   {
227      ,debug/structures .code:n =
228        {
229          \int_step_inline:nnn{#1}{\c@g__tag_struct_abs_int}
230            {
231              \msg_term:nneeee
232                    { tag/debug } { show-struct }
233                    { ##1 }
```

43

```
234                         {
235                           \prop_map_function:cN
236                             {g__tag_struct_debug_##1_prop}
237                             \msg_show_item_unbraced:nn
238                         }
239                         { } { }
240                   \msg_term:nneeee
241                         { tag/debug } { show-kids }
242                         {  ##1 }
243                         {
244                           \seq_map_function:cN
245                             {g__tag_struct_debug_kids_##1_seq}
246                             \msg_show_item_unbraced:n
247                         }
248                         { } { }
249               }
250         }
251       ,debug/structures .default:n = 0
252   }
253 ⟨/debug⟩
```

(*End of definition for* `debug/structures` (`show-key`)*. This function is documented on page 35.*)

13   Commands to extend document commands

The following commands and code parts are not core commands of tagpdf. They 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.

```
254 ⟨*package⟩
```

13.1   Document structure

<div style="margin-left:auto; text-align:right;">
\g__tag_root_default_tl
activate␣(setup-key)
activate-socket␣(setup-key)
</div>

```
255 \tl_new:N\g__tag_root_default_tl
256 \tl_gset:Nn\g__tag_root_default_tl {Document}
257
258 \hook_gput_code:nnn{begindocument}{tagpdf}{\tagstructbegin{tag=\g__tag_root_default_tl}}
259 \hook_gput_code:nnn{tagpdf/finish/before}{tagpdf}{\tagstructend}
260
261 \keys_define:nn { __tag / setup}
262  {
263    activate-socket .bool_set:N  = \l__tag_active_socket_bool,
264    activate   .code:n =
265     {
266       \keys_set:nn { __tag / setup }
267         { activate-mc,activate-tree,activate-struct,activate-socket }
268       \tl_gset:Nn\g__tag_root_default_tl {#1}
269     },
270    activate .default:n = Document
271  }
272
```

(End of definition for `\g__tag_root_default_tl`, `activate (setup-key)`, *and* `activate-socket (setup-key)`. *These functions are documented on page [34](#).)*

13.2 Structure destinations

Since TeXlive 2022 pdftex and luatex offer support for structure destinations and the pdfmanagement has backend support for. We activate them if structures are actually created. Structure destinations are actually PDF 2.0 only but they don't harm in older PDF and can improve html export.

```
273 \AddToHook{begindocument/before}
274   {
275     \bool_lazy_and:nnT
276       { \g__tag_active_struct_dest_bool }
277       { \g__tag_active_struct_bool }
278       {
279         \tl_set:Nn \l_pdf_current_structure_destination_tl
280           { __tag/struct/\g__tag_struct_stack_current_tl }
281         \pdf_activate_structure_destination:
282       }
283   }
```

13.3 Fake space

`\pdffakespace`   We need a luatex variant for `\pdffakespace`. This should probably go into the kernel at some time. We also provide a no-op version for dvi mode

```
284 \sys_if_engine_luatex:T
285   {
286     \NewDocumentCommand\pdffakespace { }
287       {
288         \__tag_fakespace:
289       }
290   }
291 \providecommand\pdffakespace{}
```

(End of definition for `\pdffakespace`. *This function is documented on page [36](#).)*

13.4 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`   At first some variables.
`\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`
`\g__tag_para_main_struct_tl`
`\l__tag_para_tag_default_tl`
`\l__tag_para_tag_tl`
`\l__tag_para_main_tag_tl`
`\l__tag_para_attr_class_tl`
`\l__tag_para_main_attr_class_tl`

```
292 ⟨/package⟩
293 ⟨base⟩\bool_new:N \l__tag_para_flattened_bool
294 ⟨base⟩\bool_new:N \l__tag_para_bool
295 ⟨*package⟩
296 \int_new:N  \g__tag_para_begin_int
297 \int_new:N  \g__tag_para_end_int
298 \int_new:N  \g__tag_para_main_begin_int
299 \int_new:N  \g__tag_para_main_end_int
```

this will hold the structure number of the current text-unit.

```
300 \tl_new:N    \g__tag_para_main_struct_tl
301 \tl_new:N    \l__tag_para_tag_default_tl
302 \tl_set:Nn  \l__tag_para_tag_default_tl { text }
303 \tl_new:N    \l__tag_para_tag_tl
304 \tl_set:Nn  \l__tag_para_tag_tl { \l__tag_para_tag_default_tl }
305 \tl_new:N    \l__tag_para_main_tag_tl
306 \tl_set:Nn  \l__tag_para_main_tag_tl {text-unit}
```

this is perhaps already defined by the block code

```
307 \tl_if_exist:NF \l__tag_para_attr_class_tl
308  {\tl_new:N \l__tag_para_attr_class_tl }
309 \tl_new:N \l__tag_para_main_attr_class_tl
```

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

__tag_gincr_para_main_begin_int:
__tag_gincr_para_main_end_int:
__tag_gincr_para_begin_int:
__tag_gincr_para_end_int:

The global para counter should be set through commands so that \tag_stop: can stop them.

```
310 \cs_new_protected:Npn \__tag_gincr_para_main_begin_int:
311  {
312    \int_gincr:N \g__tag_para_main_begin_int
313  }
314 \cs_new_protected:Npn \__tag_gincr_para_begin_int:
315  {
316    \int_gincr:N \g__tag_para_begin_int
317  }
318 \cs_new_protected:Npn \__tag_gincr_para_main_end_int:
319  {
320    \int_gincr:N \g__tag_para_main_end_int
321  }
322 \cs_new_protected:Npn \__tag_gincr_para_end_int:
323  {
324    \int_gincr:N \g__tag_para_end_int
325  }
```

(*End of definition for* __tag_gincr_para_main_begin_int: *and others.*)

__tag_start_para_ints:
__tag_stop_para_ints:

```
326 \cs_new_protected:Npn \__tag_start_para_ints:
327  {
328    \cs_set_protected:Npn \__tag_gincr_para_main_begin_int:
329      {
330        \int_gincr:N \g__tag_para_main_begin_int
331      }
332    \cs_set_protected:Npn \__tag_gincr_para_begin_int:
333      {
334        \int_gincr:N \g__tag_para_begin_int
335      }
336    \cs_set_protected:Npn \__tag_gincr_para_main_end_int:
337      {
338        \int_gincr:N \g__tag_para_main_end_int
339      }
340    \cs_set_protected:Npn \__tag_gincr_para_end_int:
341      {
```

```
342        \int_gincr:N \g__tag_para_end_int
343      }
344  }
345  \cs_new_protected:Npn \__tag_stop_para_ints:
346  {
347    \cs_set_eq:NN \__tag_gincr_para_main_begin_int:\prg_do_nothing:
348    \cs_set_eq:NN \__tag_gincr_para_begin_int: \prg_do_nothing:
349    \cs_set_eq:NN \__tag_gincr_para_main_end_int: \prg_do_nothing:
350    \cs_set_eq:NN \__tag_gincr_para_end_int: \prg_do_nothing:
351  }
```

(*End of definition for* __tag_start_para_ints: *and* __tag_stop_para_ints:.)

We want to be able to inspect the current para main structure, so we need a command to store its structure number

__tag_para_main_store_struct:

```
352  \cs_new:Npn \__tag_para_main_store_struct:
353    {
354      \tl_gset:Ne \g__tag_para_main_struct_tl {\int_use:N \c@g__tag_struct_abs_int }
355    }
```

(*End of definition for* __tag_para_main_store_struct:.)

TEMPORARLY FIX (2023-11-17). Until latex-lab is updated we must adapt a sec command:

```
356  \AddToHook{package/latex-lab-testphase-sec/after}
357    {
358      \cs_set_protected:Npn \@kernel@tag@hangfrom #1
359        {
360        \tagstructbegin{tag=\l__tag_para_tag_tl}
361        \__tag_gincr_para_begin_int:
362        \tagstructbegin{tag=Lbl}
363        \setbox\@tempboxa
364          \hbox
365            {
366              \bool_lazy_and:nnT
367                {\tag_if_active_p:}
368                {\g__tag_mode_lua_bool}
369                {\tagmcbegin{tag=Lbl}}
370              {#1}
371            }
372      \tag_stop:n{hangfrom}
373      \hangindent \wd\@tempboxa\noindent
374      \tag_start:n{hangfrom}
375      \tagmcbegin{}\box\@tempboxa\tagmcend\tagstructend\tagmcbegin{}
376      }
377  }
```

and one temporary adaptions for the block module:

```
378  \AddToHook{package/latex-lab-testphase-block/after}
379  {
380    \tl_if_exist:NT \l_tag_para_attr_class_tl
381      {
382        \tl_set:Nn \l__tag_para_attr_class_tl { \l_tag_para_attr_class_tl }
383      }
384  }
```

47

<div style="text-align: right">

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

</div>

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`

```
386 \keys_define:nn { __tag / setup }
387   {
388     paratagging      .bool_set:N = \l__tag_para_bool,
389     paratagging-show .bool_set:N = \l__tag_para_show_bool,
390     paratag          .tl_set:N   = \l__tag_para_tag_tl
391   }
392 \keys_define:nn { tag / tool}
393   {
394     paratag .tl_set:N = \l__tag_para_tag_tl,
395     unittag .tl_set:N = \l__tag_para_main_tag_tl,
396     para-flattened .bool_set:N = \l__tag_para_flattened_bool
397   }
```

(*End of definition for* `paratagging (setup-key)` *and others. These functions are documented on page 36.*)

Helper command for debugging:

```
398 \cs_new_protected:Npn \__tag_check_para_begin_show:nn #1 #2
399 %#1 color, #2 prefix
400   {
401     \bool_if:NT \l__tag_para_show_bool
402       {
403         \tag_mc_begin:n{artifact}
404         \llap{\color_select:n{#1}\tiny#2\int_use:N\g__tag_para_begin_int\ }
405         \tag_mc_end:
406       }
407   }
408
409 \cs_new_protected:Npn \__tag_check_para_end_show:nn #1 #2
410 %#1  color, #2 prefix
411   {
412     \bool_if:NT \l__tag_para_show_bool
413       {
414         \tag_mc_begin:n{artifact}
415         \rlap{\color_select:n{#1}\tiny\ #2\int_use:N\g__tag_para_end_int}
416         \tag_mc_end:
417       }
418   }
```

The para/begin and para/end code. We have two variants here: a simpler one, which must be used if the block code is not used (and so probably will disappear at some time) and a more sophisticated one that must be used if the block code is used. It is possible that we will need more variants, so we setup a socket so that the code can be easily switched.

```
419 \socket_new:nn       {tagsupport/para/begin}{0}
420 \socket_new:nn       {tagsupport/para/end}{0}
421
422 \socket_new_plug:nnn{tagsupport/para/begin}{plain}
423   {
```

```
424    \bool_if:NT \l__tag_para_bool
425      {
426        \bool_if:NF \l__tag_para_flattened_bool
427          {
428            \__tag_gincr_para_main_begin_int:
429            \tag_struct_begin:n
430              {
431                tag=\l__tag_para_main_tag_tl,
432              }
433            \__tag_para_main_store_struct:
434          }
435        \__tag_gincr_para_begin_int:
436        \tag_struct_begin:n {tag=\l__tag_para_tag_tl}
437        \__tag_check_para_begin_show:nn {green}{}
438        \tag_mc_begin:n {}
439      }
440  }
441  \socket_new_plug:nnn{tagsupport/para/begin}{block}
442  {
443    \bool_if:NT \l__tag_para_bool
444      {
445        \legacy_if:nF { @inlabel }
446          {
447            \__tag_check_typeout_v:n
448              {==>~ @endpe = \legacy_if:nTF { @endpe }{true}{false} \on@line }
449            \legacy_if:nF { @endpe }
450              {
451                \bool_if:NF \l__tag_para_flattened_bool
452                  {
453                    \__tag_gincr_para_main_begin_int:
454                    \tag_struct_begin:n
455                     {
456                      tag=\l__tag_para_main_tag_tl,
457                      attribute-class=\l__tag_para_main_attr_class_tl,
458                     }
459                    \__tag_para_main_store_struct:
460                  }
461              }
462            \__tag_gincr_para_begin_int:
463            \__tag_check_typeout_v:n {==>~increment~ P \on@line }
464            \tag_struct_begin:n
465              {
466                tag=\l__tag_para_tag_tl
467                ,attribute-class=\l__tag_para_attr_class_tl
468              }
469            \__tag_check_para_begin_show:nn {green}{\PARALABEL}
470            \tag_mc_begin:n {}
471          }
472      }
473  }
```

there was no real difference between the original and in the block variant, only a debug message. We therefore define only a plain variant.

```
474  \socket_new_plug:nnn{tagsupport/para/end}{plain}
```

```
475    {
476      \bool_if:NT \l__tag_para_bool
477        {
478          \__tag_gincr_para_end_int:
479          \__tag_check_typeout_v:n {==>~increment~ /P \on@line }
480          \tag_mc_end:
481          \__tag_check_para_end_show:nn {red}{}
482          \tag_struct_end:
483          \bool_if:NF \l__tag_para_flattened_bool
484            {
485              \__tag_gincr_para_main_end_int:
486              \tag_struct_end:
487            }
488        }
489    }
```

By default we assign the plain plug:

```
490  \socket_assign_plug:nn { tagsupport/para/begin}{plain}
491  \socket_assign_plug:nn { tagsupport/para/end}{plain}
```

And use the sockets in the hooks. Once tagging sockets exist, this can be adapted.

```
492  \AddToHook{para/begin}{ \socket_use:n { tagsupport/para/begin }
493    }
494  \AddToHook{para/end} { \socket_use:n { tagsupport/para/end } }
```

If the block code is loaded we must ensure that it doesn't overwrite the hook again. And we must reassign the para/begin plug. This can go once the block code no longer tries to adapt the hooks.

```
495  \AddToHook{package/latex-lab-testphase-block/after}
496  {
497    \RemoveFromHook{para/begin}[tagpdf]
498    \RemoveFromHook{para/end}[latex-lab-testphase-block]
499    \AddToHook{para/begin}[tagpdf]
500      {
501        \socket_use:n { tagsupport/para/begin }
502      }
503    \AddToHook{para/end}[tagpdf]
504      {
505        \socket_use:n { tagsupport/para/end }
506      }
507    \socket_assign_plug:nn { tagsupport/para/begin}{block}
508  }
509
```

We check the para count at the end. If tagging is not active it is not a error, but we issue a warning as it perhaps indicates that the testphase code didn't guard everything correctly.

```
510  \AddToHook{enddocument/info}
511  {
512    \tag_if_active:F
513      {
514        \msg_redirect_name:nnn { tag } { para-hook-count-wrong } { warning }
515      }
516    \int_compare:nNnF {\g__tag_para_main_begin_int}={\g__tag_para_main_end_int}
517        {
```

```
518        \msg_error:nneee
519          {tag}
520          {para-hook-count-wrong}
521          {\int_use:N\g__tag_para_main_begin_int}
522          {\int_use:N\g__tag_para_main_end_int}
523          {text-unit}
524      }
525    \int_compare:nNnF {\g__tag_para_begin_int}={\g__tag_para_end_int}
526      {
527        \msg_error:nneee
528          {tag}
529          {para-hook-count-wrong}
530          {\int_use:N\g__tag_para_begin_int}
531          {\int_use:N\g__tag_para_end_int}
532          {text}
533      }
534  }
```

We need at least the new-or-1 code. In generic mode we also must insert the code to finish the MC-chunks

```
535 \@ifpackageloaded{footmisc}
536   {\PackageWarning{tagpdf}{tagpdf~has~been~loaded~too~late!}} %
537   {\RequirePackage{latex-lab-testphase-new-or-1}}
538
539 \AddToHook{begindocument/before}
540  {
541    \providecommand\@kernel@tagsupport@@makecol{}
542    \providecommand\@kernel@before@cclv{}
543    \bool_if:NF \g__tag_mode_lua_bool
544      {
545        \cs_if_exist:NT \@kernel@before@footins
546          {
547            \tl_put_right:Nn \@kernel@before@footins
548              { \__tag_add_missing_mcs_to_stream:Nn \footins {footnote} }
549            \tl_put_right:Nn \@kernel@before@cclv
550              {
551                \__tag_check_typeout_v:n {====>~In~\token_to_str:N \@makecol\c_space_tl\the\c@p
552                \__tag_add_missing_mcs_to_stream:Nn \@cclv {main}
553              }
554            \tl_put_right:Nn \@kernel@tagsupport@@makecol
555              {
556                \__tag_check_typeout_v:n {====>~In~\token_to_str:N \@makecol\c_space_tl\the\c@p
557                \__tag_add_missing_mcs_to_stream:Nn \@outputbox {main}
558              }
559            \tl_put_right:Nn \@mult@ptagging@hook
560              {
561                \__tag_check_typeout_v:n {====>~In~\string\page@sofar}
562                \process@cols\mult@firstbox
563                  {
564                    \__tag_add_missing_mcs_to_stream:Nn \count@ {multicol}
565                  }
566                \__tag_add_missing_mcs_to_stream:Nn \mult@rightbox {multicol}
567              }
568          }
```

```
569        }
570      }
571  ⟨/package⟩
```

\tagpdfparaOn
\tagpdfparaOff This two command switch para mode on and off. \tagpdfsetup could be used too but is longer. An alternative is \tag_tool:n{para=false}

```
572  ⟨base⟩\newcommand\tagpdfparaOn {}
573  ⟨base⟩\newcommand\tagpdfparaOff{}
574  ⟨*package⟩
575  \renewcommand\tagpdfparaOn {\bool_set_true:N \l__tag_para_bool}
576  \renewcommand\tagpdfparaOff{\bool_set_false:N \l__tag_para_bool}
577  \keys_define:nn { tag / tool }
578    {
579      para .bool_set:N = \l__tag_para_bool,
580      para-flattened .bool_set:N = \l__tag_para_flattened_bool,
581    }
```

(*End of definition for* \tagpdfparaOn *and* \tagpdfparaOff*. These functions are documented on page 36.*)

\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}%
```

```
582  \NewDocumentCommand\tagpdfsuppressmarks{m}
583    {{\use:c{__tag_mc_disable_marks:} #1}}
```

(*End of definition for* \tagpdfsuppressmarks*. This function is documented on page 36.*)

13.5 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.

```
584  \cs_new_protected:Npn\__tag_hook_kernel_before_head:{}
585  \cs_new_protected:Npn\__tag_hook_kernel_after_head:{}
586  \cs_new_protected:Npn\__tag_hook_kernel_before_foot:{}
587  \cs_new_protected:Npn\__tag_hook_kernel_after_foot:{}
588
589  \AddToHook{begindocument}
590   {
591    \cs_if_exist:NT \@kernel@before@head
592     {
593        \tl_put_right:Nn \@kernel@before@head {\__tag_hook_kernel_before_head:}
594        \tl_put_left:Nn  \@kernel@after@head  {\__tag_hook_kernel_after_head:}
```

```
595     \tl_put_right:Nn \@kernel@before@foot {\__tag_hook_kernel_before_foot:}
596     \tl_put_left:Nn  \@kernel@after@foot  {\__tag_hook_kernel_after_foot:}
597   }
598 }
599
600 \bool_new:N \g__tag_saved_in_mc_bool
601 \cs_new_protected:Npn \__tag_exclude_headfoot_begin:
602 {
603     \bool_set_false:N  \l__tag_para_bool
604     \bool_if:NTF \g__tag_mode_lua_bool
605       {
606        \tag_mc_end_push:
607       }
608       {
609         \bool_gset_eq:NN   \g__tag_saved_in_mc_bool \g__tag_in_mc_bool
610         \bool_gset_false:N \g__tag_in_mc_bool
611       }
612     \tag_mc_begin:n {artifact}
613     \tag_stop:n{headfoot}
614 }
615 \cs_new_protected:Npn \__tag_exclude_headfoot_end:
616 {
617     \tag_start:n{headfoot}
618     \tag_mc_end:
619     \bool_if:NTF \g__tag_mode_lua_bool
620       {
621        \tag_mc_begin_pop:n{}
622       }
623       {
624         \bool_gset_eq:NN \g__tag_in_mc_bool\g__tag_saved_in_mc_bool
625       }
626 }
```

This version allows to use an Artifact structure

```
627 \__tag_attr_new_entry:nn {__tag/attr/pagination}{/O/Artifact/Type/Pagination}
628 \cs_new_protected:Npn \__tag_exclude_struct_headfoot_begin:n #1
629 {
630     \bool_set_false:N  \l__tag_para_bool
631     \bool_if:NTF \g__tag_mode_lua_bool
632       {
633        \tag_mc_end_push:
634       }
635       {
636         \bool_gset_eq:NN   \g__tag_saved_in_mc_bool \g__tag_in_mc_bool
637         \bool_gset_false:N \g__tag_in_mc_bool
638       }
639     \tag_struct_begin:n{tag=Artifact,attribute-class=__tag/attr/#1}
640     \tag_mc_begin:n {artifact=#1}
641     \tag_stop:n{headfoot}
642 }
643
644 \cs_new_protected:Npn \__tag_exclude_struct_headfoot_end:
645 {
646     \tag_start:n{headfoot}
647     \tag_mc_end:
```

```
648     \tag_struct_end:
649     \bool_if:NTF \g__tag_mode_lua_bool
650       {
651        \tag_mc_begin_pop:n{}
652       }
653       {
654         \bool_gset_eq:NN \g__tag_in_mc_bool\g__tag_saved_in_mc_bool
655       }
656   }
```

And now the keys

```
657 \keys_define:nn { __tag / setup }
658   {
659     exclude-header-footer .choice:,
660     exclude-header-footer / true .code:n =
661       {
662         \cs_set_eq:NN \__tag_hook_kernel_before_head: \__tag_exclude_headfoot_begin:
663         \cs_set_eq:NN \__tag_hook_kernel_before_foot: \__tag_exclude_headfoot_begin:
664         \cs_set_eq:NN \__tag_hook_kernel_after_head:  \__tag_exclude_headfoot_end:
665         \cs_set_eq:NN \__tag_hook_kernel_after_foot:  \__tag_exclude_headfoot_end:
666       },
667     exclude-header-footer / pagination .code:n =
668       {
669         \cs_set:Nn \__tag_hook_kernel_before_head: { \__tag_exclude_struct_headfoot_begin:n {pa
670         \cs_set:Nn \__tag_hook_kernel_before_foot: { \__tag_exclude_struct_headfoot_begin:n {pa
671         \cs_set_eq:NN \__tag_hook_kernel_after_head:  \__tag_exclude_struct_headfoot_end:
672         \cs_set_eq:NN \__tag_hook_kernel_after_foot:  \__tag_exclude_struct_headfoot_end:
673       },
674     exclude-header-footer / false .code:n =
675       {
676         \cs_set_eq:NN \__tag_hook_kernel_before_head: \prg_do_nothing:
677         \cs_set_eq:NN \__tag_hook_kernel_before_foot: \prg_do_nothing:
678         \cs_set_eq:NN \__tag_hook_kernel_after_head:  \prg_do_nothing:
679         \cs_set_eq:NN \__tag_hook_kernel_after_foot:  \prg_do_nothing:
680       },
681     exclude-header-footer .default:n = true,
682     exclude-header-footer .initial:n = true
683   }
```

(*End of definition for* `exclude-header-footer` *(setup-key). This function is documented on page 37.*)

13.6   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.

```
684 \hook_gput_code:nnn
685   {pdfannot/link/URI/before}
686   {tagpdf}
687   {
688     \tag_mc_end_push:
689     \tag_struct_begin:n { tag=Link }
690     \tag_mc_begin:n { tag=Link }
```

54

```
691    \pdfannot_dict_put:nne
692      { link/URI }
693      { StructParent }
694      { \tag_struct_parent_int: }
695    }
696
697  \hook_gput_code:nnn
698    {pdfannot/link/URI/after}
699    {tagpdf}
700    {
701      \tag_struct_insert_annot:ee {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
702      \tag_mc_end:
703      \tag_struct_end:
704      \tag_mc_begin_pop:n{}
705    }
706
707  \hook_gput_code:nnn
708    {pdfannot/link/GoTo/before}
709    {tagpdf}
710    {
711      \tag_mc_end_push:
712      \tag_struct_begin:n{tag=Link}
713      \tag_mc_begin:n{tag=Link}
714      \pdfannot_dict_put:nne
715        { link/GoTo }
716        { StructParent }
717        { \tag_struct_parent_int: }
718    }
719
720  \hook_gput_code:nnn
721    {pdfannot/link/GoTo/after}
722    {tagpdf}
723    {
724      \tag_struct_insert_annot:ee {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
725      \tag_mc_end:
726      \tag_struct_end:
727      \tag_mc_begin_pop:n{}
728
729    }
730
731  % "alternative descriptions " for PAX3. How to get better text here??
732  \pdfannot_dict_put:nnn
733   { link/URI }
734   { Contents }
735   { (url) }
736
737  \pdfannot_dict_put:nnn
738   { link/GoTo }
739   { Contents }
740   { (ref) }
741
</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} {2024-02-02} {0.98u}
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 of 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 of 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:nne
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 of 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_property_ref_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:Ne \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:ne {dict}
64                { /Limits~[\__tag_struct_get_id:n{##1-\l__tag_tmpa_int+1}~\__tag_struct_get_id:n
65                  /Names~[\l__tag_tmpa_tl]
66                }
67              \tl_put_right:Ne\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:ne {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:Ne\l__tag_tmpb_tl {\pdf_object_ref_last:}
82        }
83      \pdf_object_unnamed_write:ne {dict}{/Kids~[\l__tag_tmpb_tl]}
84      \__tag_prop_gput:cne
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.

This writes out the root object.

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

(*End of definition for* __tag_tree_write_structtreeroot:.)

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

```
108 \cs_new_protected:Npn \__tag_tree_write_structelements:
109   {
110     \int_step_inline:nnnn {1}{1}{\c@g__tag_struct_abs_int}
111       {
112         \__tag_struct_write_obj:n { ##1 }
113       }
114   }
```

(*End of definition for* __tag_tree_write_structelements:.)

1.5   ParentTree

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

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

(*End of 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.

```
116 \newcounter   { g__tag_parenttree_obj_int }
117 \hook_gput_code:nnn{begindocument}{tagpdf}
118   {
119     \int_gset:Nn
120       \c@g__tag_parenttree_obj_int
121       { \__tag_property_ref_lastpage:nn{abspage}{100}  }
122   }
```

(*End of 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

```
123 \tl_new:N \g__tag_parenttree_objr_tl
```

(*End of 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.

```
124 \cs_new_protected:Npn \__tag_parenttree_add_objr:nn #1 #2 %#1 StructParent number, #2 objref
125   {
126     \tl_gput_right:Ne \g__tag_parenttree_objr_tl
127       {
128         #1 \c_space_tl #2 ^^J
129       }
130   }
```

(*End of definition for* `\__tag_parenttree_add_objr:nn`.)

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

```
131 \tl_new:N \l__tag_parenttree_content_tl
```

(*End of 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.

```
132 \cs_new_protected:Npn \__tag_tree_parenttree_rerun_msg: {}
133 \cs_new_protected:Npn \__tag_tree_fill_parenttree:
134   {
135     \int_step_inline:nnnn{1}{1}{\__tag_property_ref_lastpage:nn{abspage}{-1}} %not quite clea
136       { %page ##1
137         \prop_clear:N \l__tag_tmpa_prop
138         \int_step_inline:nnnn{1}{1}{\__tag_property_ref_lastpage:nn{tagmcabs}{-1}}
139           {
140             %mcid####1
141             \int_compare:nT
142               {\__tag_property_ref:enn{mcid-####1}{tagabspage}{-1}=##1} %mcid is on current pa
143               {% yes
144                 \prop_put:Nee
145                   \l__tag_tmpa_prop
146                   {\__tag_property_ref:enn{mcid-####1}{tagmcid}{-1}}
147                   {\prop_item:Nn \g__tag_mc_parenttree_prop {####1}}
148               }
149           }
150         \tl_put_right:Ne\l__tag_parenttree_content_tl
151           {
152             \int_eval:n {##1-1}\c_space_tl
153             [\c_space_tl %]
154           }
155         \int_step_inline:nnnn
156           {0}
157           {1}
158           { \prop_count:N \l__tag_tmpa_prop -1 }
159           {
160             \prop_get:NnNTF \l__tag_tmpa_prop {####1} \l__tag_tmpa_tl
161               {% page#1:mcid##1:\l__tag_tmpa_tl :content
162                 \tl_put_right:Ne \l__tag_parenttree_content_tl
163                   {
164                     \pdf_object_if_exist:eTF { __tag/struct/\l__tag_tmpa_tl }
165                       {
166                         \pdf_object_ref:e { __tag/struct/\l__tag_tmpa_tl }
167                       }
168                       {
169                         null
170                       }
171                     \c_space_tl
172                   }
173               }
174               {
175                 \cs_set_protected:Npn \__tag_tree_parenttree_rerun_msg:
176                   {
```

60

```
177                        \msg_warning:nn { tag } {tree-mcid-index-wrong}
178                    }
179                }
180            }
181        \tl_put_right:Nn
182            \l__tag_parenttree_content_tl
183            {%[
184            ]^^J
185            }
186        }
187    }
```

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

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

```
188 \cs_new_protected:Npn \__tag_tree_lua_fill_parenttree:
189    {
190        \tl_set:Nn \l__tag_parenttree_content_tl
191            {
192                \lua_now:e
193                    {
194                        ltx.__tag.func.output_parenttree
195                            (
196                                \int_use:N\g_shipout_readonly_int
197                            )
198                    }
199            }
200    }
```

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

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

```
201 \cs_new_protected:Npn \__tag_tree_write_parenttree:
202    {
203        \bool_if:NTF \g__tag_mode_lua_bool
204            {
205                \__tag_tree_lua_fill_parenttree:
206            }
207            {
208                \__tag_tree_fill_parenttree:
209            }
210        \__tag_tree_parenttree_rerun_msg:
211        \tl_put_right:NV \l__tag_parenttree_content_tl\g__tag_parenttree_objr_tl
212        \pdf_object_write:nne  { __tag/tree/parenttree }{dict}
213            {
214                /Nums\c_space_tl [\l__tag_parenttree_content_tl]
215            }
216    }
```

(*End of 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. Rolemap is also used in PDF 2.0 as a fallback.

```
217 \pdf_object_new:n { __tag/tree/rolemap }
```

(*End of 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.

```
218 \cs_new_protected:Npn \__tag_tree_write_rolemap:
219 {
220   \bool_if:NT \g__tag_role_add_mathml_bool
221     {
222       \prop_map_inline:Nn \g__tag_role_NS_mathml_prop
223         {
224           \prop_gput:Nnn \g__tag_role_rolemap_prop {##1}{Span}
225         }
226     }
227   \prop_map_inline:Nn\g__tag_role_rolemap_prop
228     {
229       \tl_if_eq:nnF {##1}{##2}
230         {
231           \pdfdict_gput:nne {g__tag_role/RoleMap_dict}
232           {##1}
233           {\pdf_name_from_unicode_e:n{##2}}
234         }
235     }
236   \pdf_object_write:nne  { __tag/tree/rolemap }{dict}
237     {
238       \pdfdict_use:n{g__tag_role/RoleMap_dict}
239     }
240 }
```

(*End of 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.

__tag_tree_write_classmap:

```
241 \cs_new_protected:Npn \__tag_tree_write_classmap:
242   {
243     \tl_clear:N \l__tag_tmpa_tl
244     \seq_gremove_duplicates:N \g__tag_attr_class_used_seq
245     \seq_set_map:NNn \l__tag_tmpa_seq \g__tag_attr_class_used_seq
246       {
247         ##1\c_space_tl
248         <<
249           \prop_item:Nn
250             \g__tag_attr_entries_prop
```

```
251                   {##1}
252               >>
253             }
254           \tl_set:Ne \l__tag_tmpa_tl
255             {
256               \seq_use:Nn
257                 \l__tag_tmpa_seq
258                 { \iow_newline: }
259             }
260           \tl_if_empty:NF
261             \l__tag_tmpa_tl
262             {
263               \pdf_object_new:n { __tag/tree/classmap }
264               \pdf_object_write:nne
265                 { __tag/tree/classmap }
266                 {dict}
267                 { \l__tag_tmpa_tl }
268               \__tag_prop_gput:cne
269                 { g__tag_struct_0_prop }
270                 { ClassMap }
271                 { \pdf_object_ref:n { __tag/tree/classmap }  }
272             }
273         }
```

(*End of 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.

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

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

`\__tag_tree_write_namespaces:`

```
275 \cs_new_protected:Npn \__tag_tree_write_namespaces:
276   {
277     \pdf_version_compare:NnF < {2.0}
278       {
279         \prop_map_inline:Nn \g__tag_role_NS_prop
280           {
281             \pdfdict_if_empty:nF {g__tag_role/RoleMapNS_##1_dict}
282               {
283                 \pdf_object_write:nne {__tag/RoleMapNS/##1}{dict}
284                   {
285                     \pdfdict_use:n {g__tag_role/RoleMapNS_##1_dict}
286                   }
287                 \pdfdict_gput:nne{g__tag_role/Namespace_##1_dict}
288                   {RoleMapNS}{\pdf_object_ref:n {__tag/RoleMapNS/##1}}
289               }
290             \pdf_object_write:nne{tag/NS/##1}{dict}
291               {
292                 \pdfdict_use:n {g__tag_role/Namespace_##1_dict}
```

```
293              }
294            }
295          \pdf_object_write:nne {__tag/tree/namespaces}{array}
296            {
297               \prop_map_tokens:Nn \g__tag_role_NS_prop{\use_ii:nn}
298            }
299        }
300      }
```

(*End of 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

__tag_finish_structure:

```
301 \hook_new:n {tagpdf/finish/before}
302 \cs_new_protected:Npn \__tag_finish_structure:
303   {
304      \bool_if:NT\g__tag_active_tree_bool
305        {
306           \hook_use:n {tagpdf/finish/before}
307           \__tag_tree_final_checks:
308           \__tag_tree_write_parenttree:
309           \__tag_tree_write_idtree:
310           \__tag_tree_write_rolemap:
311           \__tag_tree_write_classmap:
312           \__tag_tree_write_namespaces:
313           \__tag_tree_write_structelements: %this is rather slow!!
314           \__tag_tree_write_structtreeroot:
315        }
316   }
```

(*End of 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.

```
317 \hook_gput_code:nnn{begindocument}{tagpdf}
318   {
319      \bool_if:NT\g__tag_active_tree_bool
320        {
321           \hook_gput_code:nnn{shipout/before} { tagpdf/structparents }
322             {
323                \pdfmanagement_add:nne
324                  { Page }
325                  { StructParents }
326                  { \int_eval:n { \g_shipout_readonly_int} }
327             }
328        }
329   }
330 ⟨/package⟩
```

64

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.

`\tag_mc_reset_box:N` ⋆   `\tag_mc_reset_box:N {⟨box⟩}`

This resets in lua mode the mc attributes to the one currently in use. It does nothing in generic mode.

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. If it is empty, nothing will happen.

`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. If it is empty, nothing will happen.

`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} {2024-02-02} {0.98u}
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 ⟨*base⟩
8 \newcounter { g__tag_MCID_abs_int }
```

(*End of definition for* g__tag_MCID_abs_int.)

__tag_get_data_mc_counter: This command allows `\tag_get:n` to get the current state of the mc counter with the keyword `mc_counter`. By comparing the numbers it can be used to check the number of structure commands in a piece of code.

```
9 \cs_new:Npn \__tag_get_data_mc_counter:
10   {
11     \int_use:N \c@g__tag_MCID_abs_int
12   }
13 ⟨/base⟩
```

(*End of definition for* __tag_get_data_mc_counter:.)

__tag_get_mc_abs_cnt: A (expandable) function to get the current value of the cnt. TODO: duplicate of the previous one, this should be cleaned up.

```
14 ⟨*shared⟩
15 \cs_new:Npn \__tag_get_mc_abs_cnt: { \int_use:N \c@g__tag_MCID_abs_int }
```

(*End of definition for* __tag_get_mc_abs_cnt:.)

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

```
16 \bool_new:N \g__tag_in_mc_bool
```

(*End of 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

```
17 \__tag_prop_new:N \g__tag_mc_parenttree_prop
```

(*End of 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:

```
18 \seq_new:N \g__tag_mc_stack_seq
```

(*End of 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.

```
19 \tl_new:N \l__tag_mc_artifact_type_tl
```

(*End of definition for* `\l__tag_mc_artifact_type_tl`.)

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

```
20 \bool_new:N \l__tag_mc_key_stash_bool
21 \bool_new:N \l__tag_mc_artifact_bool
```

(*End of 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`

```
22 \tl_new:N \l__tag_mc_key_tag_tl
23 \tl_new:N \g__tag_mc_key_tag_tl
24 \tl_new:N \l__tag_mc_key_label_tl
25 \tl_new:N \l__tag_mc_key_properties_tl
```

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

3.2 Functions

`\__tag_mc_handle_mc_label:e` 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`. The reference command is based on l3ref.

```
26 \cs_new:Npn \__tag_mc_handle_mc_label:e #1
27   {
28     \__tag_property_record:en{tagpdf-#1}{tagabspage,tagmcabs}
29   }
```

(*End of definition for* `\__tag_mc_handle_mc_label:e`.)

`\__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

```
30 \cs_new_protected:Npn \__tag_mc_set_label_used:n #1 %#1 labelname
31   {
32     \tl_new:c { g__tag_mc_label_\tl_to_str:n{#1}_used_tl }
33   }
34 ⟨/shared⟩
```

(*End of 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?

```
35 ⟨base⟩\cs_new_protected:Npn \tag_mc_use:n #1 { \__tag_whatsits: }
36 ⟨∗shared⟩
37 \cs_set_protected:Npn \tag_mc_use:n #1 %#1: label name
38   {
39     \__tag_check_if_active_struct:T
40       {
41         \tl_set:Ne  \l__tag_tmpa_tl { \__tag_property_ref:nnn{tagpdf-#1}{tagmcabs}{} }
42         \tl_if_empty:NTF\l__tag_tmpa_tl
43           {
44             \msg_warning:nnn {tag} {mc-label-unknown} {#1}
45           }
46           {
47             \cs_if_free:cTF { g__tag_mc_label_\tl_to_str:n{#1}_used_tl }
48               {
49                 \__tag_mc_handle_stash:e { \l__tag_tmpa_tl }
50                 \__tag_mc_set_label_used:n {#1}
51               }
52               {
53                 \msg_warning:nnn {tag}{mc-used-twice}{#1}
54               }
55           }
56       }
57   }
58 ⟨/shared⟩
```

(*End of definition for* \tag_mc_use:n. *This function is documented on page 65.*)

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

```
59 ⟨base⟩\cs_new_protected:Npn \tag_mc_artifact_group_begin:n #1 {}
60 ⟨base⟩\cs_new_protected:Npn \tag_mc_artifact_group_end:{}
61 ⟨∗shared⟩
62 \cs_set_protected:Npn \tag_mc_artifact_group_begin:n #1
63   {
64   \tag_mc_end_push:
65   \tag_mc_begin:n {artifact=#1}
66   \group_begin:
67   \tag_stop:n{artifact-group}
68   }
69
70 \cs_set_protected:Npn \tag_mc_artifact_group_end:
71   {
72   \tag_start:n{artifact-group}
73   \group_end:
74   \tag_mc_end:
75   \tag_mc_begin_pop:n{}
76   }
77 ⟨/shared⟩
```

(End of definition for `\tag_mc_artifact_group_begin:n` *and* `\tag_mc_artifact_group_end:`. *These functions are documented on page 65.)*

`\tag_mc_reset_box:N` This allows to reset the mc-attributes in box. On base and generic mode it should do nothing.

```
78 ⟨base⟩\cs_new_protected:Npn \tag_mc_reset_box:N #1 {}
```

(End of definition for `\tag_mc_reset_box:N`. *This function is documented on page 66.)*

`\tag_mc_end_push:`
`\tag_mc_begin_pop:n`

```
79 ⟨base⟩\cs_new_protected:Npn \tag_mc_end_push: {}
80 ⟨base⟩\cs_new_protected:Npn \tag_mc_begin_pop:n #1 {}
81 ⟨∗shared⟩
82 \cs_set_protected:Npn \tag_mc_end_push:
83   {
84     \__tag_check_if_active_mc:T
85       {
86         \__tag_mc_if_in:TF
87           {
88             \seq_gpush:Ne \g__tag_mc_stack_seq { \tag_get:n {mc_tag} }
89             \__tag_check_mc_pushed_popped:nn
90               { pushed }
91               { \tag_get:n {mc_tag} }
92             \tag_mc_end:
93           }
94           {
95             \seq_gpush:Nn \g__tag_mc_stack_seq {-1}
96             \__tag_check_mc_pushed_popped:nn { pushed }{-1}
97           }
98       }
99   }
100
101 \cs_set_protected:Npn \tag_mc_begin_pop:n #1
102   {
103     \__tag_check_if_active_mc:T
104       {
105         \seq_gpop:NNTF \g__tag_mc_stack_seq \l__tag_tmpa_tl
106           {
107             \tl_if_eq:NnTF \l__tag_tmpa_tl {-1}
108               {
109                 \__tag_check_mc_pushed_popped:nn {popped}{-1}
110               }
111               {
112                 \__tag_check_mc_pushed_popped:nn {popped}{\l__tag_tmpa_tl}
113                 \tag_mc_begin:n {tag=\l__tag_tmpa_tl,#1}
114               }
115           }
116           {
117             \__tag_check_mc_pushed_popped:nn {popped}{empty~stack,~nothing}
118           }
119       }
120   }
```

(End of definition for `\tag_mc_end_push:` *and* `\tag_mc_begin_pop:n`. *These functions are documented on page 65.)*

3.3 Keys

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

stash␣(mc-key)
__artifact-bool
__artifact-type
the two internal artifact keys are use to define the public `artifact`. For now we add support for the subtypes Header and Footer. Watermark,PageNum, LineNum,Redaction,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.

```
121 \keys_define:nn { __tag / mc }
122   {
123     stash                    .bool_set:N    = \l__tag_mc_key_stash_bool,
124     __artifact-bool          .bool_set:N    = \l__tag_mc_artifact_bool,
125     __artifact-type          .choice:,
126     __artifact-type / pagination .code:n     =
127       {
128         \tl_set:Nn \l__tag_mc_artifact_type_tl { Pagination }
129       },
130     __artifact-type / pagination/header .code:n    =
131       {
132         \tl_set:Nn \l__tag_mc_artifact_type_tl { Pagination/Subtype/Header }
133       },
134     __artifact-type / pagination/footer .code:n    =
135       {
136         \tl_set:Nn \l__tag_mc_artifact_type_tl { Pagination/Subtype/Footer }
137       },
138     __artifact-type / layout      .code:n    =
139       {
140         \tl_set:Nn \l__tag_mc_artifact_type_tl { Layout }
141       },
142     __artifact-type / page        .code:n    =
143       {
144         \tl_set:Nn \l__tag_mc_artifact_type_tl { Page }
145       },
146     __artifact-type / background .code:n     =
147       {
148         \tl_set:Nn \l__tag_mc_artifact_type_tl { Background }
149       },
150     __artifact-type / notype      .code:n    =
151       {
152         \tl_set:Nn \l__tag_mc_artifact_type_tl {}
153       },
154     __artifact-type /         .code:n    =
155       {
156         \tl_set:Nn \l__tag_mc_artifact_type_tl {}
157       },
158   }
```

(*End of definition for* stash (mc-key)*,* __artifact-bool*, and* __artifact-type*. This function is documented on page 66.*)

```
159 ⟨/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} {2024-02-02} {0.98u}
4  {part of tagpdf - code related to marking chunks - generic mode}
5 ⟨/generic⟩
6 ⟨*debug⟩
7 \ProvidesExplPackage {tagpdf-debug-generic} {2024-02-02} {0.98u}
8  {part of tagpdf - debugging code related to marking chunks - generic mode}
9 ⟨/debug⟩
```

1.1 Variables

```
10 ⟨*generic⟩
```

\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.

```
11 \tl_new:N \l__tag_mc_ref_abspage_tl
```

(*End of definition for* \l__tag_mc_ref_abspage_tl.)

\l__tag_mc_tmpa_tl temporary variable

```
12 \tl_new:N \l__tag_mc_tmpa_tl
```

(*End of 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.

```
13 \newmarks  \g__tag_mc_marks
```

(*End of definition for* \g__tag_mc_marks.)

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

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

(*End of definition for* \g__tag_mc_main_marks_seq, \g__tag_mc_footnote_marks_seq, *and* \g__tag_-
mc_multicol_marks_seq.)

```
58
59  \cs_new_protected:Npn \__tag_mc_end_marks:
60    {
61      \tex_marks:D \g__tag_mc_marks
62        {
63          e-, %first of end pair
64          \int_use:N\c@g__tag_MCID_abs_int, %mc-num
65          \g__tag_struct_stack_current_tl,  %structure num
66        }
67      \tex_marks:D \g__tag_mc_marks
68        {
69          e+, %second of end pair
70          \int_use:N\c@g__tag_MCID_abs_int, %mc-num
71          \g__tag_struct_stack_current_tl,  %structure num
72        }
73    }
```

(*End of 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.

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

(*End of 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.

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

(*End of 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.

```
89  \cs_new_protected:Npn \__tag_mc_store:nnn #1 #2 #3 %#1 mc-prev, #2 mc-num #3 structure-
    num
90    {
91      %\prop_show:N \g__tag_struct_cont_mc_prop
92      \prop_get:NnNTF \g__tag_struct_cont_mc_prop {#1} \l__tag_tmpa_tl
93        {
```

```
94        \prop_gput:Nne \g__tag_struct_cont_mc_prop {#1}{ \l__tag_tmpa_tl \__tag_struct_mcid_d:
95      }
96      {
97        \prop_gput:Nne \g__tag_struct_cont_mc_prop {#1}{ \__tag_struct_mcid_dict:n {#2}}
98      }
99    \prop_gput:Nee \g__tag_mc_parenttree_prop
100      {#2}
101      {#3}
102  }
103  \cs_generate_variant:Nn \__tag_mc_store:nnn {eee}
```

(*End of definition for* __tag_mc_store:nnn.)

__tag_mc_insert_extra_tmb:n These two functions should be used in the output routine at the place where a mc-literal
__tag_mc_insert_extra_tme:n 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.

```
104  \cs_new_protected:Npn \__tag_mc_insert_extra_tmb:n #1 % #1 stream: e.g. main or footnote
105    {
106      \__tag_check_typeout_v:n {=>~ first~ \seq_use:Nn \l__tag_mc_firstmarks_seq {,~}}
107      \__tag_check_typeout_v:n {=>~ bot~ \seq_use:Nn \l__tag_mc_botmarks_seq {,~}}
108      \__tag_check_if_mc_tmb_missing:TF
109        {
110          \__tag_check_typeout_v:n {=>~ TMB~ ~ missing~ ---~ inserted}
111          %test if artifact
112          \int_compare:nNnTF { \seq_item:cn { g__tag_mc_#1_marks_seq } {3} } = {-
1}
113            {
114              \tl_set:Ne \l__tag_tmpa_tl { \seq_item:cn { g__tag_mc_#1_marks_seq } {4} }
115              \__tag_mc_handle_artifact:N \l__tag_tmpa_tl
116            }
117            {
118              \exp_args:Ne
119              \__tag_mc_bdc_mcid:n
120                {
121                  \seq_item:cn { g__tag_mc_#1_marks_seq } {4}
122                }
123              \str_if_eq:eeTF
124                {
125                  \seq_item:cn { g__tag_mc_#1_marks_seq } {5}
126                }
127                {}
128                {
129                  %store
130                  \__tag_mc_store:eee
131                    {
132                      \seq_item:cn { g__tag_mc_#1_marks_seq } {2}
133                    }
134                    { \int_eval:n{\c@g__tag_MCID_abs_int} }
135                    {
```

```
136                      \seq_item:cn { g__tag_mc_#1_marks_seq } {3}
137                    }
138                }
139                {
140                    %stashed -> warning!!
141                }
142            }
143        }
144        {
145            \__tag_check_typeout_v:n {=>~ TMB~ not~ missing}
146        }
147  }
148
149 \cs_new_protected:Npn \__tag_mc_insert_extra_tme:n #1 % #1 stream, eg. main or footnote
150  {
151    \__tag_check_if_mc_tme_missing:TF
152      {
153        \__tag_check_typeout_v:n {=>~ TME~ ~ missing~ --~ inserted}
154        \__tag_mc_emc:
155        \seq_gset_eq:cN
156          {  g__tag_mc_#1_marks_seq }
157          \l__tag_mc_botmarks_seq
158      }
159      {
160        \__tag_check_typeout_v:n {=>~ TME~ not~ missing}
161      }
162  }
```

(*End of 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.

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

76

```
171          \seq_log:c { g__tag_mc_#2_marks_seq}
172        }
```

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

```
173        \box_set_ht:Nn \l__tag_tmpa_box \c_zero_dim
174        \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.

```
175        \box_set_ht:Nn \l__tag_tmpb_box \c_zero_dim
176        \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.

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

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

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

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

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

(*End of 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.

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

First set up a temp box for trial splitting.

```
188        \vbadness\maxdimen
189        \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).

```
190        \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`

```
191        \exp_args:NNe
192        \seq_set_from_clist:Nn \l__tag_mc_firstmarks_seq
193          { \tex_splitfirstmarks:D \g__tag_mc_marks }
```

Some debugging info:

```
194 %      \iow_term:n { First~ mark~ from~ this~ box: }
195 %      \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!

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

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

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.

```
206        {
207          \__tag_check_typeout_v:n
208            {
209              Pick~ up~ new~ bot~ mark!
210            }
211          \exp_args:NNe
212          \seq_set_from_clist:Nn \l__tag_mc_botmarks_seq
213            { \tex_splitbotmarks:D   \g__tag_mc_marks }
214        }
```

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

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

(*End of 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 is a test if a mc is open or not. It depends simply on a global boolean: mc-chunks are added linearly so nesting should not be relevant.

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.

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

(*End of definition for* `\__tag_mc_if_in:TF` *and* `\tag_mc_if_in:TF`. *This function is documented on page* *65.*)

`\__tag_mc_bmc:n`
`\__tag_mc_emc:`
`\__tag_mc_bdc:nn`

These are the low-level commands. There are now equal to the pdfmanagement commands generic mode, but we use an indirection in case luamode need something else. change 04.08.2018: the commands do not check the validity of the arguments or try to escape them, this should be done before using them. change 2023-08-18: we are delaying the writing to the shipout.

```
229  % #1 tag, #2 properties
230  \cs_set_eq:NN \__tag_mc_bmc:n  \pdf_bmc:n
231  \cs_set_eq:NN \__tag_mc_emc:    \pdf_emc:
232  \cs_set_eq:NN \__tag_mc_bdc:nn \pdf_bdc:nn
233  \cs_set_eq:NN \__tag_mc_bdc_shipout:ee \pdf_bdc_shipout:ee
```

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

`\__tag_mc_bdc_mcid:nn`
`\__tag_mc_bdc_mcid:n`
`\__tag_mc_handle_mcid:nn`
`\__tag_mc_handle_mcid:VV`

This create a BDC mark with an `/MCID` key. Most of the work here is to get the current number value for the MCID: they must be numbered by page starting with 0 and then successively. The first argument is the tag, e.g. `P` or `Span`, the second is used to pass more properties. Starting with texlive 2023 this is much simpler and faster as we can use delay the numbering to the shipout. We also define a wrapper around the low-level command as luamode will need something different.

```
234  \bool_if:NTF\g__tag_delayed_shipout_bool
235    {
236      \hook_gput_code:nnn {shipout/before}{tagpdf}{ \flag_clear:n { __tag/mcid } }
237      \cs_set_protected:Npn \__tag_mc_bdc_mcid:nn #1 #2
238        {
239          \int_gincr:N \c@g__tag_MCID_abs_int
240          \__tag_property_record:eV
241            {
242              mcid-\int_use:N \c@g__tag_MCID_abs_int
243            }
244          \c__tag_property_mc_clist
245          \__tag_mc_bdc_shipout:ee
246            {#1}
247            {
248              /MCID~\flag_height:n { __tag/mcid }
249              \flag_raise:n { __tag/mcid }~  #2
250            }
251        }
252    }
```

if the engine is too old, we have to revert to earlier method.

```
253    {
254      \msg_new:nnn { tagpdf } { old-engine }
```

```
255      {
256        The~engine~or~the~PDF management~is~too~old~or\\
257        delayed~shipout~has~been~disabled.\\
258        Fast~numbering~of~MC-chunks~not~available.\\
259        More~compilations~will~be~needed~in~generic~mode.
260      }
261    \msg_warning:nn { tagpdf} { old-engine }
262    \__tag_prop_new:N \g__tag_MCID_byabspage_prop
263    \int_new:N \g__tag_MCID_tmp_bypage_int
264    \cs_generate_variant:Nn \__tag_mc_bdc:nn {ne}
```

revert the attribute:

```
265    \__tag_property_gset:nnnn {tagmcid } { now }
266        {0}  { \int_use:N \g__tag_MCID_tmp_bypage_int }
267    \cs_new_protected:Npn \__tag_mc_bdc_mcid:nn #1 #2
268      {
269        \int_gincr:N \c@g__tag_MCID_abs_int
270        \tl_set:Ne \l__tag_mc_ref_abspage_tl
271          {
272            \__tag_property_ref:enn %3 args
273              {
274                mcid-\int_use:N \c@g__tag_MCID_abs_int
275              }
276              { tagabspage }
277              {-1}
278          }
279        \prop_get:NoNTF
280          \g__tag_MCID_byabspage_prop
281          {
282            \l__tag_mc_ref_abspage_tl
283          }
284          \l__tag_mc_tmpa_tl
285          {
286            %key already present, use value for MCID and add 1 for the next
287            \int_gset:Nn \g__tag_MCID_tmp_bypage_int { \l__tag_mc_tmpa_tl }
288            \__tag_prop_gput:Nee
289              \g__tag_MCID_byabspage_prop
290              { \l__tag_mc_ref_abspage_tl }
291              { \int_eval:n {\l__tag_mc_tmpa_tl +1} }
292          }
293          {
294            %key not present, set MCID to 0 and insert 1
295            \int_gzero:N \g__tag_MCID_tmp_bypage_int
296            \__tag_prop_gput:Nee
297              \g__tag_MCID_byabspage_prop
298              { \l__tag_mc_ref_abspage_tl }
299              {1}
300          }
301        \__tag_property_record:eV
302          {
303            mcid-\int_use:N \c@g__tag_MCID_abs_int
304          }
305          \c__tag_property_mc_clist
306        \__tag_mc_bdc:ne
307          {#1}
```

```
308            { /MCID~\int_eval:n { \g__tag_MCID_tmp_bypage_int }~ \exp_not:n { #2 } }
309        }
310  }
311  \cs_new_protected:Npn \__tag_mc_bdc_mcid:n #1
312    {
313      \__tag_mc_bdc_mcid:nn {#1} {}
314    }
315
316  \cs_new_protected:Npn \__tag_mc_handle_mcid:nn #1 #2 %#1 tag, #2 properties
317    {
318      \__tag_mc_bdc_mcid:nn {#1} {#2}
319    }
320
321  \cs_generate_variant:Nn \__tag_mc_handle_mcid:nn {VV}
```

(*End of 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:e

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?

```
322  \cs_new_protected:Npn \__tag_mc_handle_stash:n #1 %1 mcidnum
323    {
324      \__tag_check_mc_used:n {#1}
325      \__tag_struct_kid_mc_gput_right:nn
326        { \g__tag_struct_stack_current_tl }
327        {#1}
328      \prop_gput:Nee \g__tag_mc_parenttree_prop
329        {#1}
330        { \g__tag_struct_stack_current_tl }
331    }
332  \cs_generate_variant:Nn \__tag_mc_handle_stash:n { e }
```

(*End of definition for* __tag_mc_handle_stash:n.)

__tag_mc_bmc_artifact:
__tag_mc_bmc_artifact:n
__tag_mc_handle_artifact:N

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

```
333  \cs_new_protected:Npn  \__tag_mc_bmc_artifact:
334    {
335      \__tag_mc_bmc:n {Artifact}
336    }
337  \cs_new_protected:Npn \__tag_mc_bmc_artifact:n #1
338    {
339      \__tag_mc_bdc:nn {Artifact}{/Type/#1}
340    }
341  \cs_new_protected:Npn \__tag_mc_handle_artifact:N #1
342    % #1 is a var containing the artifact type
343    {
344      \int_gincr:N \c@g__tag_MCID_abs_int
345      \tl_if_empty:NTF #1
346        { \__tag_mc_bmc_artifact: }
347        { \exp_args:NV\__tag_mc_bmc_artifact:n #1 }
348    }
```

81

(*End of 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.

```
349 \cs_new:Nn \__tag_get_data_mc_tag: { \g__tag_mc_key_tag_tl }
350 ⟨/generic⟩
```

(*End of 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.

```
351 ⟨base⟩\cs_new_protected:Npn \tag_mc_begin:n #1 { \__tag_whatsits: \int_gincr:N \c@g__tag_MCID_a
352 ⟨base⟩\cs_new_protected:Nn \tag_mc_end:{ \__tag_whatsits: }
353 ⟨*generic | debug⟩
354 ⟨*generic⟩
355 \cs_set_protected:Npn \tag_mc_begin:n #1 %#1 keyval
356   {
357     \__tag_check_if_active_mc:T
358       {
359 ⟨/generic⟩
360 ⟨*debug⟩
361 \cs_set_protected:Npn \tag_mc_begin:n #1 %#1 keyval
362   {
363     \__tag_check_if_active_mc:TF
364       {
365         \__tag_debug_mc_begin_insert:n { #1 }
366 ⟨/debug⟩
367       \group_begin: %hm
368       \__tag_check_mc_if_nested:
369       \bool_gset_true:N \g__tag_in_mc_bool
```

set default MC tags to structure:

```
370       \tl_set_eq:NN \l__tag_mc_key_tag_tl \g__tag_struct_tag_tl
371       \tl_gset_eq:NN\g__tag_mc_key_tag_tl \g__tag_struct_tag_tl
372       \keys_set:nn { __tag / mc } {#1}
373       \bool_if:NTF \l__tag_mc_artifact_bool
374         { %handle artifact
375           \__tag_mc_handle_artifact:N \l__tag_mc_artifact_type_tl
376           \exp_args:NV
377           \__tag_mc_artifact_begin_marks:n \l__tag_mc_artifact_type_tl
378         }
379         { %handle mcid type
380           \__tag_check_mc_tag:N  \l__tag_mc_key_tag_tl
381           \__tag_mc_handle_mcid:VV
382             \l__tag_mc_key_tag_tl
383             \l__tag_mc_key_properties_tl
384           \__tag_mc_begin_marks:oo{\l__tag_mc_key_tag_tl}{\l__tag_mc_key_label_tl}
385           \tl_if_empty:NF {\l__tag_mc_key_label_tl}
386             {
387               \exp_args:NV
388               \__tag_mc_handle_mc_label:e \l__tag_mc_key_label_tl
389             }
```

```
390              \bool_if:NF \l__tag_mc_key_stash_bool
391                {
392                  \exp_args:NV\__tag_struct_get_parentrole:nNN
393                      \g__tag_struct_stack_current_tl
394                      \l__tag_get_parent_tmpa_tl
395                      \l__tag_get_parent_tmpb_tl
396                  \__tag_check_parent_child:VVnnN
397                      \l__tag_get_parent_tmpa_tl
398                      \l__tag_get_parent_tmpb_tl
399                      {MC}{}
400                      \l__tag_parent_child_check_tl
401                  \int_compare:nNnT {\l__tag_parent_child_check_tl}<{0}
402                    {
403                      \prop_get:cnN
404                        { g__tag_struct_ \g__tag_struct_stack_current_tl _prop}
405                        {S}
406                        \l__tag_tmpa_tl
407                      \msg_warning:nneee
408                        { tag }
409                        {role-parent-child}
410                        { \l__tag_get_parent_tmpa_tl/\l__tag_get_parent_tmpb_tl  }
411                        { MC~(real content) }
412                        { not~allowed~
413                          (struct~\g__tag_struct_stack_current_tl,~\l__tag_tmpa_tl)
414                        }
415                    }
416                  \__tag_mc_handle_stash:e { \int_use:N \c@g__tag_MCID_abs_int }
417                }
418            }
419          \group_end:
420        }
421 ⟨∗debug⟩
422        {
423          \__tag_debug_mc_begin_ignore:n { #1 }
424        }
425 ⟨/debug⟩
426    }
427 ⟨∗generic⟩
428 \cs_set_protected:Nn \tag_mc_end:
429    {
430      \__tag_check_if_active_mc:T
431        {
432 ⟨/generic⟩
433 ⟨∗debug⟩
434 \cs_set_protected:Nn \tag_mc_end:
435    {
436      \__tag_check_if_active_mc:TF
437        {
438          \__tag_debug_mc_end_insert:
439 ⟨/debug⟩
440          \__tag_check_mc_if_open:
441          \bool_gset_false:N \g__tag_in_mc_bool
442          \tl_gset:Nn  \g__tag_mc_key_tag_tl { }
443          \__tag_mc_emc:
```

```
444            \__tag_mc_end_marks:
445        }
```
⟨∗debug⟩
```
447        {
448            \__tag_debug_mc_end_ignore:
449        }
```
⟨/debug⟩
```
451    }
```
⟨/generic | debug⟩

(*End of definition for* \tag_mc_begin:n *and* \tag_mc_end:. *These functions are documented on page 65.*)

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.

```
453 ⟨∗generic⟩
454 \keys_define:nn { __tag / mc }
455   {
456     tag .code:n = % the name (H,P,Span) etc
457       {
458         \tl_set:Ne    \l__tag_mc_key_tag_tl { #1 }
459         \tl_gset:Ne   \g__tag_mc_key_tag_tl { #1 }
460       },
461     raw   .code:n =
462       {
463         \tl_put_right:Ne \l__tag_mc_key_properties_tl { #1 }
464       },
465     alt .code:n      = % Alt property
466       {
467         \str_set_convert:Noon
468           \l__tag_tmpa_str
469           { #1 }
470           { default }
471           { utf16/hex }
472         \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
473         \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
474       },
475     alttext .meta:n = {alt=#1},
476     actualtext .code:n      = % ActualText property
477       {
478         \tl_if_empty:oF{#1}
479         {
480           \str_set_convert:Noon
481             \l__tag_tmpa_str
482             { #1 }
483             { default }
484             { utf16/hex }
485           \tl_put_right:Nn \l__tag_mc_key_properties_tl { /ActualText~< }
486           \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
487         }
```

84

```
488        },
489      label .tl_set:N        = \l__tag_mc_key_label_tl,
490      artifact .code:n       =
491        {
492          \exp_args:Nne
493            \keys_set:nn
494              { __tag / mc }
495              { __artifact-bool, __artifact-type=#1 }
496        },
497      artifact .default:n    = {notype}
498    }
499 ⟨/generic⟩
```

(*End of definition for* tag (mc-key) *and others. These functions are documented on page* 66.)

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} {2024-02-02} {0.98u}
4    {tagpdf - mc code only for the luamode }
5  ⟨/luamode⟩
6  ⟨*debug⟩
7  \ProvidesExplPackage {tagpdf-debug-lua} {2024-02-02} {0.98u}
8    {part of tagpdf - debugging code related to marking chunks - lua mode}
9  ⟨/debug⟩
```

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

```
10 ⟨*luamode⟩
11 \hook_gput_code:nnn{begindocument}{tagpdf/mc}
12   {
13     \bool_if:NT\g__tag_active_space_bool
14       {
15         \lua_now:e
16           {
17             if~luatexbase.callbacktypes.pre_shipout_filter~then~
18               luatexbase.add_to_callback("pre_shipout_filter", function(TAGBOX)~
19               ltx.__tag.func.space_chars_shipout(TAGBOX)~return~true~
20               end, "tagpdf")~
21               if~luatexbase.declare_callback_rule~then~
22                 luatexbase.declare_callback_rule("pre_shipout_filter", "luaotfload.dvi", "afte
23               end~
24             end
25           }
26         \lua_now:e
27           {
28             if~luatexbase.callbacktypes.pre_shipout_filter~then~
29             token.get_next()~
30             end
31           }\@secondoftwo\@gobble
32           {
33             \hook_gput_code:nnn{shipout/before}{tagpdf/lua}
34               {
35                 \lua_now:e
36                   { ltx.__tag.func.space_chars_shipout (tex.box["ShipoutBox"]) }
37               }
38           }
39       }
40     \bool_if:NT\g__tag_active_mc_bool
41       {
42         \lua_now:e
43           {
44             if~luatexbase.callbacktypes.pre_shipout_filter~then~
45               luatexbase.add_to_callback("pre_shipout_filter", function(TAGBOX)~
46               ltx.__tag.func.mark_shipout(TAGBOX)~return~true~
47               end, "tagpdf")~
48             end
49           }
50         \lua_now:e
51           {
52             if~luatexbase.callbacktypes.pre_shipout_filter~then~
53             token.get_next()~
54             end
55           }\@secondoftwo\@gobble
56           {
57             \hook_gput_code:nnn{shipout/before}{tagpdf/lua}
58               {
59                 \lua_now:e
60                   { ltx.__tag.func.mark_shipout (tex.box["ShipoutBox"]) }
61               }
```

```
62                    }
63                }
64            }
```

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.

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

(*End of 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.

```
66 \prg_new_conditional:Nnn \__tag_mc_if_in: {p,T,F,TF}
67    {
68      \int_compare:nNnTF
69        { -2147483647 }
70        =
71        {\lua_now:e
72          {
73            tex.print(\int_use:N \c_document_cctab,tex.getattribute(luatexbase.attributes.g__ta
74          }
75        }
76        { \prg_return_false:  }
77        { \prg_return_true: }
78    }
79
80 \prg_new_eq_conditional:NNn \tag_mc_if_in: \__tag_mc_if_in: {p,T,F,TF}
```

(*End of definition for* __tag_mc_if_in:TF *and* \tag_mc_if_in:TF. *This function is documented on page 65.*)

__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.

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

88

```
102          }
103      }
104
105  \cs_generate_variant:Nn\__tag_mc_lua_set_mc_type_attr:n { o }
106
107  \cs_new:Nn \__tag_mc_lua_unset_mc_type_attr:
108      {
109        \lua_now:e
110          {
111            tex.setattribute
112              (
113                "global",
114                luatexbase.attributes.g__tag_mc_type_attr,
115                -2147483647
116              )
117          }
118        \lua_now:e
119          {
120            tex.setattribute
121              (
122                "global",
123                luatexbase.attributes.g__tag_mc_cnt_attr,
124                -2147483647
125              )
126          }
127      }
128
```

(*End of 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

```
129  \cs_new:Nn \__tag_mc_insert_mcid_kids:n
130      {
131        \lua_now:e { ltx.__tag.func.mc_insert_kids (#1,0) }
132      }
133
134  \cs_new:Nn \__tag_mc_insert_mcid_single_kids:n
135      {
136        \lua_now:e {ltx.__tag.func.mc_insert_kids (#1,1) }
137      }
```

(*End of 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:e This is the lua variant for the command to put an mcid absolute number in the current structure.

```
138  ⟨/luamode⟩
139  ⟨*luamode | debug⟩
140  ⟨luamode⟩\cs_new_protected:Npn \__tag_mc_handle_stash:n #1 %1 mcidnum
141  ⟨debug⟩\cs_set_protected:Npn \__tag_mc_handle_stash:n #1 %1 mcidnum
142      {
143        \__tag_check_mc_used:n { #1 }
144        \seq_gput_right:cn % Don't fill a lua table due to the command in the item,
145                           % so use the kernel command
```

```
146        { g__tag_struct_kids_\g__tag_struct_stack_current_tl _seq }
147        {
148          \__tag_mc_insert_mcid_kids:n {#1}%
149        }
150 ⟨debug⟩     \seq_gput_right:cn % Don't fill a lua table due to the command in the item,
151 ⟨debug⟩                         % so use the kernel command
152 ⟨debug⟩        { g__tag_struct_debug_kids_\g__tag_struct_stack_current_tl _seq }
153 ⟨debug⟩        {
154 ⟨debug⟩          MC~#1%
155 ⟨debug⟩        }
156      \lua_now:e
157        {
158          ltx.__tag.func.store_struct_mcabs
159            (
160              \g__tag_struct_stack_current_tl,#1
161            )
162        }
163      \prop_gput:Nee
164        \g__tag_mc_parenttree_prop
165        { #1 }
166        { \g__tag_struct_stack_current_tl }
167    }
168 ⟨/luamode | debug⟩
169 ⟨*luamode⟩
170 \cs_generate_variant:Nn \__tag_mc_handle_stash:n { e }
```

(*End of 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.

```
171 \cs_set_protected:Nn \tag_mc_begin:n
172   {
173      \__tag_check_if_active_mc:T
174        {
175          \group_begin:
176          %\__tag_check_mc_if_nested:
177          \bool_gset_true:N \g__tag_in_mc_bool
178          \bool_set_false:N\l__tag_mc_artifact_bool
179          \tl_clear:N \l__tag_mc_key_properties_tl
180          \int_gincr:N \c@g__tag_MCID_abs_int
```

set the default tag to the structure:

```
181          \tl_set_eq:NN \l__tag_mc_key_tag_tl \g__tag_struct_tag_tl
182          \tl_gset_eq:NN\g__tag_mc_key_tag_tl \g__tag_struct_tag_tl
183          \lua_now:e
184            {
185              ltx.__tag.func.store_mc_data(\__tag_get_mc_abs_cnt:,"tag","\g__tag_struct_tag_tl")
186            }
187          \keys_set:nn { __tag / mc }{ label={}, #1 }
188          %check that a tag or artifact has been used
189          \__tag_check_mc_tag:N \l__tag_mc_key_tag_tl
190          %set the attributes:
191          \__tag_mc_lua_set_mc_type_attr:o  { \l__tag_mc_key_tag_tl }
192          \bool_if:NF \l__tag_mc_artifact_bool
193            { % store the absolute num name in a label:
```

90

```
194                        \tl_if_empty:NF {\l__tag_mc_key_label_tl}
195                          {
196                            \exp_args:NV
197                             \__tag_mc_handle_mc_label:e  \l__tag_mc_key_label_tl
198                          }
199                    % if not stashed record the absolute number
200                     \bool_if:NF \l__tag_mc_key_stash_bool
201                          {
202                            \exp_args:NV\__tag_struct_get_parentrole:nNN
203                             \g__tag_struct_stack_current_tl
204                             \l__tag_get_parent_tmpa_tl
205                             \l__tag_get_parent_tmpb_tl
206                           \__tag_check_parent_child:VVnnN
207                             \l__tag_get_parent_tmpa_tl
208                             \l__tag_get_parent_tmpb_tl
209                             {MC}{}
210                             \l__tag_parent_child_check_tl
211                           \int_compare:nNnT {\l__tag_parent_child_check_tl}<{0}
212                            {
213                              \prop_get:cnN
214                               { g__tag_struct_ \g__tag_struct_stack_current_tl _prop}
215                               {S}
216                               \l__tag_tmpa_tl
217                              \msg_warning:nneee
218                                { tag }
219                                {role-parent-child}
220                                { \l__tag_get_parent_tmpa_tl/\l__tag_get_parent_tmpb_tl  }
221                                { MC~(real content) }
222                                {
223                                  not~allowed~
224                                  (struct~\g__tag_struct_stack_current_tl,~\l__tag_tmpa_tl)
225                                }
226                            }
227                            \__tag_mc_handle_stash:e { \__tag_get_mc_abs_cnt: }
228                          }
229                      }
230               \group_end:
231          }
232    }
```

(*End of definition for* `\tag_mc_begin:n`*. This function is documented on page 65.*)

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

```
233  \cs_set_protected:Nn \tag_mc_end:
234    {
235      \__tag_check_if_active_mc:T
236        {
237          %\__tag_check_mc_if_open:
238          \bool_gset_false:N \g__tag_in_mc_bool
239          \bool_set_false:N\l__tag_mc_artifact_bool
240          \__tag_mc_lua_unset_mc_type_attr:
241          \tl_set:Nn  \l__tag_mc_key_tag_tl { }
242          \tl_gset:Nn \g__tag_mc_key_tag_tl { }
243        }
244    }
```

(End of definition for `\tag_mc_end:`*. This function is documented on page 65.)*

`\tag_mc_reset_box:N`   This allows to reset the mc-attributes in box. On base and generic mode it should do nothing.

```
245 \cs_set_protected:Npn \tag_mc_reset_box:N #1
246   {
247     \lua_now:e
248       {
249         local~type=tex.getattribute(luatexbase.attributes.g__tag_mc_type_attr)
250         local~mc=tex.getattribute(luatexbase.attributes.g__tag_mc_cnt_attr)
251         ltx.__tag.func.update_mc_attributes(tex.getbox(\int_use:N #1),mc,type)
252       }
253   }
```

(End of definition for `\tag_mc_reset_box:N`*. This function is documented on page 66.)*

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

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

(End of 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)

```
255 \keys_define:nn { __tag / mc }
256   {
257     tag .code:n = %
258       {
259         \tl_set:Ne   \l__tag_mc_key_tag_tl { #1 }
260         \tl_gset:Ne  \g__tag_mc_key_tag_tl { #1 }
261         \lua_now:e
262           {
263             ltx.__tag.func.store_mc_data(\__tag_get_mc_abs_cnt:,"tag","#1")
264           }
265       },
266     raw .code:n =
267       {
268         \tl_put_right:Ne \l__tag_mc_key_properties_tl { #1 }
269         \lua_now:e
270           {
271             ltx.__tag.func.store_mc_data(\__tag_get_mc_abs_cnt:,"raw","#1")
272           }
273       },
274     alt .code:n      = % Alt property
275       {
276         \tl_if_empty:oF{#1}
277           {
278             \str_set_convert:Noon
279               \l__tag_tmpa_str
280               { #1 }
281               { default }
282               { utf16/hex }
283             \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
```

```
284        \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
285        \lua_now:e
286          {
287            ltx.__tag.func.store_mc_data
288              (
289                \__tag_get_mc_abs_cnt:,"alt","/Alt~<\str_use:N \l__tag_tmpa_str>"
290              )
291          }
292        }
293     },
294   alttext .meta:n = {alt=#1},
295   actualtext .code:n      = % Alt property
296     {
297        \tl_if_empty:oF{#1}
298          {
299            \str_set_convert:Noon
300              \l__tag_tmpa_str
301              { #1 }
302              { default }
303              { utf16/hex }
304            \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
305            \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
306            \lua_now:e
307              {
308                ltx.__tag.func.store_mc_data
309                  (
310                    \__tag_get_mc_abs_cnt:,
311                    "actualtext",
312                    "/ActualText~<\str_use:N \l__tag_tmpa_str>"
313                  )
314              }
315          }
316     },
317   label .code:n =
318     {
319        \tl_set:Nn\l__tag_mc_key_label_tl { #1 }
320        \lua_now:e
321          {
322            ltx.__tag.func.store_mc_data
323              (
324                \__tag_get_mc_abs_cnt:,"label","#1"
325              )
326          }
327     },
328   __artifact-store .code:n =
329     {
330        \lua_now:e
331          {
332            ltx.__tag.func.store_mc_data
333              (
334                \__tag_get_mc_abs_cnt:,"artifact","#1"
335              )
336          }
337     },
```

```
338    artifact .code:n        =
339      {
340        \exp_args:Nne
341          \keys_set:nn
342            { __tag / mc}
343            { __artifact-bool, __artifact-type=#1, tag=Artifact }
344        \exp_args:Nne
345          \keys_set:nn
346            { __tag / mc }
347            { __artifact-store=\l__tag_mc_artifact_type_tl }
348      },
349    artifact .default:n    = { notype }
350  }
351
352 ⟨/luamode⟩
```

(End of definition for tag (mc-key) *and others. These functions are documented on page* 66*.)*

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_num:n`

`\tag_struct_use:n{⟨label⟩}`
`\tag_struct_use_num:n{⟨structure number⟩}`

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_num⟩}`. 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.

`\tag_struct_gput:nnn` `\tag_struct_gput:nnn{`⟨*structure number*⟩`}{`⟨*keyword*⟩`}{`⟨*value*⟩`}`

This is a command that allows to update the data of a structure. This often can't done simply by replacing the value, as we have to preserve and extend existing content. We use therefore dedicated functions adjusted to the key in question. 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` which updates the Ref key (an array)

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 `\property_ref:nn{tagpdfstruct-label}{tagstruct}` to retrieve it.

title␣(struct-key) This keys allows to set the dictionary entry `/Title` in the structure object. The value
title-o␣(struct-key) 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. If it is empty, nothing will happen.

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. If it is empty, nothing will happen.

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)
AFref␣(struct-key)
AFinline␣(struct-key)
AFinline-o␣(struct-key)
texsource
mathml

`AF = ⟨object name⟩`
`AFref = ⟨object reference⟩`
`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.

`texsource` is a special variant of `AF-inline-o` which embeds the file as `.tex` source with the `/AFrelationship` key set to `/Source`. It also sets the `/Desc` key to a (currently) fix text.

`mathml` is a special variant of `AF-inline-o` which embeds the file as `.xml` file with the `/AFrelationship` key set to `/Supplement`. It also sets the `/Desc` key to a (currently) fix text.

The argument of `AF` is an object name referring an embedded file as declared for example with `\pdf_object_new:n` or with the l3pdffile module. `AF` expands its argument (this allows e.g. to use some variable for automatic numbering) and can be used more than once, to associate more than one file.

The argument of `AFref` is an object reference to an embedded file or a variable expanding to such a object reference in the format as you would get e.g. from `\pdf_-object_ref_last:` or `\pdf_object_ref:n` (and which is different for the various engines!). The key allows to make use of anonymous objects. Like `AF` the `AFref` key expands its argument and can be used more than once, to associate more than one file. *It does not check if the reference is valid!*

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

<div align="center">

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

</div>

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

newattribute␣(setup-key) `newattribute = {⟨name⟩}{⟨Content⟩}`

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},
  }
```

root-AF␣(setup-key) `root-AF = ⟨object name⟩`

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} {2024-02-02} {0.98u}
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 of 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 of definition for* `\g__tag_struct_objR_seq`.)

`\c__tag_struct_null_tl` In lua mode we have to test if the kids a null

```
10 \tl_const:Nn\c__tag_struct_null_tl {null}
```

(*End of definition for* `\c__tag_struct_null_tl`.)

`\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.

```
11 \__tag_prop_new:N  \g__tag_struct_cont_mc_prop
```

(*End of 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.

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

(*End of 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.

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

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

`\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 of 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`, $N >= 1$ 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,
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 of 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
\l__tag_struct_roletag_tl
\g__tag_struct_roletag_NS_tl Use by the tag key to store the tag and the namespace. The role tag variables will hold locally rolemapping info needed for the parent-child checks

```
58  \tl_new:N \g__tag_struct_tag_tl
59  \tl_new:N \g__tag_struct_tag_NS_tl
60  \tl_new:N \l__tag_struct_roletag_tl
61  \tl_new:N \l__tag_struct_roletag_NS_tl
```

(*End of definition for* `\g__tag_struct_tag_tl` *and others.*)

`\l__tag_struct_key_label_tl` This will hold the label value.

```
62  \tl_new:N \l__tag_struct_key_label_tl
```

(*End of definition for* `\l__tag_struct_key_label_tl`.)

`\l__tag_struct_elem_stash_bool` This will keep track of the stash status

```
63  \bool_new:N \l__tag_struct_elem_stash_bool
```

(*End of 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.

```
64  ⟨/package⟩
65  ⟨base⟩\prop_new:N \g__tag_struct_dest_num_prop
66  ⟨*package⟩
```

(*End of 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.

```
67  \prop_new:N \g__tag_struct_ref_by_dest_prop
```

(*End of 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`

```
68  \cs_new:Npn \__tag_struct_output_prop_aux:nn #1 #2 %#1 num, #2 key
69    {
70      \prop_if_in:cnT
71        { g__tag_struct_#1_prop }
72        { #2 }
73        {
74          \c_space_tl/#2~ \prop_item:cn{ g__tag_struct_#1_prop } { #2 }
75        }
```

101

```
76      }
77
78   \cs_new_protected:Npn \__tag_new_output_prop_handler:n #1
79     {
80       \cs_new:cn { __tag_struct_output_prop_#1:n }
81         {
82           \__tag_struct_output_prop_aux:nn {#1}{##1}
83         }
84     }
85  ⟨/package⟩
```

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

__tag_struct_prop_gput:nnn   The structure props must be filled in various places. For this we use a common command which also takes care of the debug package:

```
86  ⟨*package | debug⟩
87  ⟨package⟩\cs_new_protected:Npn \__tag_struct_prop_gput:nnn #1 #2 #3
88  ⟨debug⟩\cs_set_protected:Npn \__tag_struct_prop_gput:nnn #1 #2 #3
89   {
90     \__tag_prop_gput:cnn
91         { g__tag_struct_#1_prop }{#2}{#3}
92  ⟨debug⟩\prop_gput:cnn { g__tag_struct_debug_#1_prop } {#2} {#3}
93   }
94  \cs_generate_variant:Nn \__tag_struct_prop_gput:nnn {nne,nee,nno}
95  ⟨/package | debug⟩
```

(*End of definition for* __tag_struct_prop_gput:nnn.)

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.

```
96  ⟨*package⟩
97  \tl_gset:Nn \g__tag_struct_stack_current_tl {0}
```

__tag_pdf_name_e:n

```
98  \cs_new:Npn \__tag_pdf_name_e:n #1{\pdf_name_from_unicode_e:n{#1}}
99  ⟨/package⟩
```

(*End of definition for* __tag_pdf_name_e:n.)

g__tag_struct_0_prop
g__tag_struct_kids_0_seq

```
100 ⟨*package⟩
101 \__tag_prop_new:c { g__tag_struct_0_prop }
102 \__tag_new_output_prop_handler:n {0}
103 \__tag_seq_new:c  { g__tag_struct_kids_0_seq }
104
105 \__tag_struct_prop_gput:nne
106   { 0 }
107   { Type }
108   { \pdf_name_from_unicode_e:n {StructTreeRoot} }
```

```
109
110  \__tag_struct_prop_gput:nne
111    { 0 }
112    { S }
113    { \pdf_name_from_unicode_e:n {StructTreeRoot} }

114
115  \__tag_struct_prop_gput:nne
116    { 0 }
117    { rolemap }
118    { {StructTreeRoot}{pdf} }

119
120  \__tag_struct_prop_gput:nne
121    { 0 }
122    { parentrole }
123    { {StructTreeRoot}{pdf} }

124
```

Namespaces are pdf 2.0. If the code moves into the kernel, the setting must be probably delayed.

```
125  \pdf_version_compare:NnF < {2.0}
126   {
127     \__tag_struct_prop_gput:nne
128     { 0 }
129     { Namespaces }
130     { \pdf_object_ref:n { __tag/tree/namespaces } }
131   }
132  ⟨/package⟩
```

In debug mode we have to copy the root manually as it is already setup:

```
133  ⟨debug⟩\prop_new:c { g__tag_struct_debug_0_prop }
134  ⟨debug⟩\seq_new:c  { g__tag_struct_debug_kids_0_seq }
135  ⟨debug⟩\prop_gset_eq:cc { g__tag_struct_debug_0_prop }{ g__tag_struct_0_prop }
136  ⟨debug⟩\prop_gremove:cn { g__tag_struct_debug_0_prop }{Namespaces}
```

(*End of 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

```
137  ⟨*package⟩
138  \cs_new:Npn \__tag_struct_get_id:n #1 %#1=struct num
139    {
140      (
141       ID.
142       \prg_replicate:nn
143       { \int_abs:n{\g__tag_tree_id_pad_int - \tl_count:e { \int_to_arabic:n { #1 } }} }
144       { 0 }
145       \int_to_arabic:n { #1 }
146      )
147    }
```

(*End of 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.

```
148 \pdf_version_compare:NnTF < {2.0}
149 {
150    \cs_new_protected:Npn \__tag_struct_set_tag_info:nnn #1 #2 #3
151      %#1 structure number, #2 tag, #3 NS
152      {
153         \__tag_struct_prop_gput:nne
154           { #1 }
155           { S }
156           { \pdf_name_from_unicode_e:n {#2}  } %
157      }
158 }
159 {
160    \cs_new_protected:Npn \__tag_struct_set_tag_info:nnn #1 #2 #3
161      {
162         \__tag_struct_prop_gput:nne
163           { #1 }
164           { S }
165           { \pdf_name_from_unicode_e:n {#2} } %
166         \prop_get:NnNT \g__tag_role_NS_prop {#3} \l__tag_get_tmpc_tl
167           {
168              \__tag_struct_prop_gput:nne
169                { #1 }
170                { NS }
171                { \l__tag_get_tmpc_tl } %
172           }
173      }
174 }
175 \cs_generate_variant:Nn \__tag_struct_set_tag_info:nnn {eVV}
```

(*End of definition for* __tag_struct_set_tag_info:nnn.)

__tag_struct_get_parentrole:nNN

We also need a way to get the tag info needed for parent child check from parent structures.

```
176 \cs_new_protected:Npn \__tag_struct_get_parentrole:nNN #1 #2 #3
177    %#1 struct num, #2 tlvar for tag , #3 tlvar for NS
178      {
179         \prop_get:cnNTF
180           { g__tag_struct_#1_prop }
181           { parentrole }
182           \l__tag_get_tmpc_tl
183           {
184              \tl_set:Ne #2{\exp_last_unbraced:NV\use_i:nn  \l__tag_get_tmpc_tl}
185              \tl_set:Ne #3{\exp_last_unbraced:NV\use_ii:nn \l__tag_get_tmpc_tl}
186           }
187           {
188              \tl_clear:N#2
189              \tl_clear:N#3
190           }
191      }
192 \cs_generate_variant:Nn\__tag_struct_get_parentrole:nNN {eNN}
```

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:ne`

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!

```
193 \cs_new:Npn \__tag_struct_mcid_dict:n #1 %#1 MCID absnum
194   {
195     <<
196      /Type \c_space_tl /MCR \c_space_tl
197      /Pg
198        \c_space_tl
199      \pdf_pageobject_ref:n { \__tag_property_ref:enn{mcid-#1}{tagabspage}{1} }
200       /MCID \c_space_tl \__tag_property_ref:enn{mcid-#1}{tagmcid}{1}
201     >>
202   }
203 ⟨/package⟩
204 ⟨*package | debug⟩
205 ⟨package⟩\cs_new_protected:Npn \__tag_struct_kid_mc_gput_right:nn #1 #2 %#1 structure num, #2 
206 ⟨debug⟩\cs_set_protected:Npn \__tag_struct_kid_mc_gput_right:nn #1 #2 %#1 structure num, #2 MC
207   {
208     \__tag_seq_gput_right:ce
209       { g__tag_struct_kids_#1_seq }
210       {
211         \__tag_struct_mcid_dict:n {#2}
212       }
213 ⟨debug⟩    \seq_gput_right:cn
214 ⟨debug⟩      { g__tag_struct_debug_kids_#1_seq }
215 ⟨debug⟩      {
216 ⟨debug⟩        MC~#2
217 ⟨debug⟩      }
218     \__tag_seq_gput_right:cn
219       { g__tag_struct_kids_#1_seq }
220       {
221         \prop_item:Nn \g__tag_struct_cont_mc_prop {#2}
222       }
223   }
224 ⟨package⟩\cs_generate_variant:Nn \__tag_struct_kid_mc_gput_right:nn {ne}
```

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

`\__tag_struct_kid_struct_gput_right:nn`
`\__tag_struct_kid_struct_gput_right:ee`

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

```
225 ⟨package⟩\cs_new_protected:Npn\__tag_struct_kid_struct_gput_right:nn #1 #2 %#1 num of parent st
```

```
226 ⟨debug⟩\cs_set_protected:Npn\__tag_struct_kid_struct_gput_right:nn #1 #2 %#1 num of parent str
227     {
228       \__tag_seq_gput_right:ce
229         { g__tag_struct_kids_#1_seq }
230         {
231           \pdf_object_ref:n { __tag/struct/#2 }
232         }
233 ⟨debug⟩      \seq_gput_right:cn
234 ⟨debug⟩        { g__tag_struct_debug_kids_#1_seq }
235 ⟨debug⟩        {
236 ⟨debug⟩          Struct~#2
237 ⟨debug⟩        }
238     }
239
240 ⟨package⟩\cs_generate_variant:Nn \__tag_struct_kid_struct_gput_right:nn {ee}
```

(*End of definition for* __tag_struct_kid_struct_gput_right:nn.)

__tag_struct_kid_OBJR_gput_right:nnn At last the command to add an OBJR object. This has to write an object first. The
__tag_struct_kid_OBJR_gput_right:eee 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

```
241 ⟨package⟩\cs_new_protected:Npn\__tag_struct_kid_OBJR_gput_right:nnn #1 #2 #3 %#1 num of parent
242 ⟨package⟩                                                                  %#2 obj reference
243 ⟨package⟩                                                                  %#3 page object reference
244 ⟨debug⟩\cs_set_protected:Npn\__tag_struct_kid_OBJR_gput_right:nnn #1 #2 #3
245     {
246       \pdf_object_unnamed_write:nn
247         { dict }
248         {
249           /Type/OBJR/Obj~#2/Pg~#3
250         }
251       \__tag_seq_gput_right:ce
252         { g__tag_struct_kids_#1_seq }
253         {
254           \pdf_object_ref_last:
255         }
256 ⟨debug⟩      \seq_gput_right:ce
257 ⟨debug⟩        { g__tag_struct_debug_kids_#1_seq }
258 ⟨debug⟩        {
259 ⟨debug⟩          OBJR~reference
260 ⟨debug⟩        }
261     }
262 ⟨/package | debug⟩
263 ⟨*package⟩
264 \cs_generate_variant:Nn\__tag_struct_kid_OBJR_gput_right:nnn { eee }
```

(*End of definition for* __tag_struct_kid_OBJR_gput_right:nnn.)

__tag_struct_exchange_kid_command:N In luamode it can happen that a single kid in a structure is split at a page break into
__tag_struct_exchange_kid_command:c 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.

```
265 \cs_new_protected:Npn\__tag_struct_exchange_kid_command:N #1 %#1 = seq var
266     {
```

```
267     \seq_gpop_left:NN #1 \l__tag_tmpa_tl
268     \regex_replace_once:nnN
269       { \c{\__tag_mc_insert_mcid_kids:n} }
270       { \c{\__tag_mc_insert_mcid_single_kids:n} }
271       \l__tag_tmpa_tl
272     \seq_gput_left:NV #1 \l__tag_tmpa_tl
273   }
274
275 \cs_generate_variant:Nn\__tag_struct_exchange_kid_command:N { c }
```

(*End of 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.

```
276 \cs_new_protected:Npn \__tag_struct_fill_kid_key:n #1 %#1 is the struct num
277   {
278     \bool_if:NF\g__tag_mode_lua_bool
279       {
280         \seq_clear:N \l__tag_tmpa_seq
281         \seq_map_inline:cn { g__tag_struct_kids_#1_seq }
282          { \seq_put_right:Ne \l__tag_tmpa_seq { ##1 } }
283         %\seq_show:c { g__tag_struct_kids_#1_seq }
284         %\seq_show:N \l__tag_tmpa_seq
285         \seq_remove_all:Nn \l__tag_tmpa_seq {}
286         %\seq_show:N \l__tag_tmpa_seq
287         \seq_gset_eq:cN { g__tag_struct_kids_#1_seq } \l__tag_tmpa_seq
288       }
289
290     \int_case:nnF
291       {
292         \seq_count:c
293           {
294             g__tag_struct_kids_#1_seq
295           }
296       }
297       {
298         { 0 }
299         { } %no kids, do nothing
300         { 1 } % 1 kid, insert
301         {
302           % in this case we need a special command in
303           % luamode to get the array right. See issue #13
304           \bool_if:NTF\g__tag_mode_lua_bool
305             {
306               \__tag_struct_exchange_kid_command:c
307               {g__tag_struct_kids_#1_seq}
```

check if we get null

```
308               \tl_set:Ne\l__tag_tmpa_tl
309                 {\use:e{\seq_item:cn {g__tag_struct_kids_#1_seq} {1}}}
310               \tl_if_eq:NNF\l__tag_tmpa_tl \c__tag_struct_null_tl
311                 {
312                   \__tag_struct_prop_gput:nne
313                     {#1}
```

107

```
314                              {K}
315                              {
316                                \seq_item:cn
317                                  {
318                                    g__tag_struct_kids_#1_seq
319                                  }
320                                  {1}
321                              }
322                          }
323                      }
324                      {
325                        \__tag_struct_prop_gput:nne
326                          {#1}
327                          {K}
328                          {
329                            \seq_item:cn
330                              {
331                                g__tag_struct_kids_#1_seq
332                              }
333                              {1}
334                          }
335                      }
336                  } %
337              }
338              { %many kids, use an array
339                \__tag_struct_prop_gput:nne
340                  {#1}
341                  {K}
342                  {
343                    [
344                      \seq_use:cn
345                        {
346                          g__tag_struct_kids_#1_seq
347                        }
348                        {
349                          \c_space_tl
350                        }
351                    ]
352                  }
353              }
354          }
355
```

(*End of 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.

```
356  \cs_new_protected:Npn \__tag_struct_get_dict_content:nN #1 #2 %#1: stucture num
357    {
358      \tl_clear:N #2
359      \seq_map_inline:cn
```

```
360        {
361          c__tag_struct_
362            \int_compare:nNnTF{#1}={0}{StructTreeRoot}{StructElem}
363            _entries_seq
364        }
365        {
366          \tl_put_right:Ne
367            #2
368            {
369                \prop_if_in:cnT
370                  { g__tag_struct_#1_prop }
371                  { ##1 }
372                  {
373                      \c_space_tl/##1~
```

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

```
374                      \cs_if_exist_use:cTF {__tag_struct_format_##1:e}
375                        {
376                          { \prop_item:cn{ g__tag_struct_#1_prop } { ##1 } }
377                        }
378                        {
379                          \prop_item:cn{ g__tag_struct_#1_prop } { ##1 }
380                        }
381                  }
382            }
383        }
384    }
```

(*End of 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:

```
385 \cs_new:Nn\__tag_struct_format_Ref:n{[#1]}
386 \cs_generate_variant:Nn\__tag_struct_format_Ref:n{e}
```

(*End of 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.

```
387 \cs_new_protected:Npn \__tag_struct_write_obj:n #1 % #1 is the struct num
388   {
389     \pdf_object_if_exist:nTF { __tag/struct/#1 }
390       {
```

It can happen that a structure is not used and so has not parent. Simply ignoring it is problematic as it is also recorded in the IDTree, so we make an artifact out of it.

```
391          \prop_get:cnNF { g__tag_struct_#1_prop } {P}\l__tag_tmpb_tl
392            {
393              \prop_gput:cne { g__tag_struct_#1_prop } {P}{\pdf_object_ref:n { __tag/struct/0  }
394              \prop_gput:cne { g__tag_struct_#1_prop } {S}{/Artifact}
395              \seq_if_empty:cF {g__tag_struct_kids_#1_seq}
396                {
397                    \msg_warning:nnee
398                      {tag}
399                      {struct-orphan}
```

```
400                       { #1 }
401                         {\seq_count:c{g__tag_struct_kids_#1_seq}}
402                     }
403                   }
404             \__tag_struct_fill_kid_key:n { #1 }
405             \__tag_struct_get_dict_content:nN { #1 } \l__tag_tmpa_tl
406             \exp_args:Ne
407                   \pdf_object_write:nne
408                     { __tag/struct/#1 }
409                     {dict}
410                     {
411                       \l__tag_tmpa_tl\c_space_tl
412                       /ID~\__tag_struct_get_id:n{#1}
413                     }
414
415         }
416         {
417           \msg_error:nnn { tag } { struct-no-objnum } { #1}
418         }
419     }
```

(*End of 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:nne
          { 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:
```

```
420  \cs_new_protected:Npn \__tag_struct_insert_annot:nn #1 #2 %#1 object reference to the annotat
421                                                            %#2 structparent number
422    {
423      \bool_if:NT \g__tag_active_struct_bool
424        {
425          %get the number of the parent structure:
426          \seq_get:NNF
427            \g__tag_struct_stack_seq
428            \l__tag_struct_stack_parent_tmpa_tl
```

```
429              {
430                  \msg_error:nn { tag } { struct-faulty-nesting }
431              }
432          %put the obj number of the annot in the kid entry, this also creates
433          %the OBJR object
434          \__tag_property_record:nn {@tag@objr@page@#2 }{ tagabspage }
435          \__tag_struct_kid_OBJR_gput_right:eee
436              {
437                  \l__tag_struct_stack_parent_tmpa_tl
438              }
439              {
440                  #1 %
441              }
442              {
443                  \pdf_pageobject_ref:n { \__tag_property_ref:nnn {@tag@objr@page@#2 }{ tagabspage }
444              }
445          % add the parent obj number to the parent tree:
446          \exp_args:Nne
447          \__tag_parenttree_add_objr:nn
448              {
449                  #2
450              }
451              {
452                  \pdf_object_ref:e { __tag/struct/\l__tag_struct_stack_parent_tmpa_tl }
453              }
454          % increase the int:
455          \int_gincr:N \c@g__tag_parenttree_obj_int
456          }
457      }
```

(*End of 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`.

```
458  \cs_new:Npn \__tag_get_data_struct_tag:
459      {
460          \exp_args:Ne
461          \tl_tail:n
462              {
463                  \prop_item:cn {g__tag_struct_\g__tag_struct_stack_current_tl _prop}{S}
464              }
465      }
```

(*End of 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`.

```
466  \cs_new:Npn \__tag_get_data_struct_id:
467      {
468          \__tag_struct_get_id:n {\g__tag_struct_stack_current_tl}
469      }
470  ⟨/package⟩
```

(*End of 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

```
471 ⟨*base⟩
472 \cs_new:Npn \__tag_get_data_struct_num:
473   {
474     \g__tag_struct_stack_current_tl
475   }
476 ⟨/base⟩
```

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

`\__tag_get_data_struct_counter:` this command allows `\tag_get:n` to get the current state of the structure counter with the keyword `struct_counter`. By comparing the numbers it can be used to check the number of structure commands in a piece of code.

```
477 ⟨*base⟩
478 \cs_new:Npn \__tag_get_data_struct_counter:
479   {
480     \int_use:N \c@g__tag_struct_abs_int
481   }
482 ⟨/base⟩
```

(*End of definition for* `\__tag_get_data_struct_counter:`.)

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)`

```
483 ⟨*package⟩
484 \keys_define:nn { __tag / struct }
485   {
486     label .tl_set:N      = \l__tag_struct_key_label_tl,
487     stash .bool_set:N    = \l__tag_struct_elem_stash_bool,
488     parent .code:n       =
489       {
490         \bool_lazy_and:nnTF
491           {
492             \prop_if_exist_p:c { g__tag_struct_\int_eval:n {#1}_prop }
493           }
494           {
495             \int_compare_p:nNn {#1}<{\c@g__tag_struct_abs_int}
496           }
497           { \tl_set:Ne \l__tag_struct_stack_parent_tmpa_tl { \int_eval:n {#1} } }
498           {
499             \msg_warning:nnee { tag } { struct-unknown }
500               { \int_eval:n {#1} }
501               { parent~key~ignored }
502           }
503       },
504     parent .default:n    = {-1},
505     tag    .code:n       = % S property
506       {
```

112

```
507      \seq_set_split:Nne \l__tag_tmpa_seq { / } {#1/\prop_item:Ne\g__tag_role_tags_NS_prop{#
508      \tl_gset:Ne \g__tag_struct_tag_tl   { \seq_item:Nn\l__tag_tmpa_seq {1} }
509      \tl_gset:Ne \g__tag_struct_tag_NS_tl{ \seq_item:Nn\l__tag_tmpa_seq {2} }
510      \__tag_check_structure_tag:N \g__tag_struct_tag_tl
511    },
512  title .code:n       = % T property
513    {
514      \str_set_convert:Nnnn
515        \l__tag_tmpa_str
516        { #1 }
517        { default }
518        { utf16/hex }
519      \__tag_struct_prop_gput:nne
520        { \int_use:N \c@g__tag_struct_abs_int }
521        { T }
522        { <\l__tag_tmpa_str> }
523    },
524  title-o .code:n        = % T property
525    {
526      \str_set_convert:Nonn
527        \l__tag_tmpa_str
528        { #1 }
529        { default }
530        { utf16/hex }
531      \__tag_struct_prop_gput:nne
532        { \int_use:N \c@g__tag_struct_abs_int }
533        { T }
534        { <\l__tag_tmpa_str> }
535    },
536  alt .code:n      = % Alt property
537    {
538     \tl_if_empty:oF{#1}
539       {
540        \str_set_convert:Noon
541          \l__tag_tmpa_str
542          { #1 }
543          { default }
544          { utf16/hex }
545        \__tag_struct_prop_gput:nne
546          { \int_use:N \c@g__tag_struct_abs_int }
547          { Alt }
548          { <\l__tag_tmpa_str> }
549       }
550    },
551  alttext .meta:n = {alt=#1},
552  actualtext .code:n  = % ActualText property
553    {
554      \tl_if_empty:oF{#1}
555        {
556         \str_set_convert:Noon
557           \l__tag_tmpa_str
558           { #1 }
559           { default }
560           { utf16/hex }
```

```
561            \__tag_struct_prop_gput:nne
562              { \int_use:N \c@g__tag_struct_abs_int }
563              { ActualText }
564              { <\l__tag_tmpa_str>}
565          }
566      },
567    lang .code:n       = % Lang property
568      {
569        \__tag_struct_prop_gput:nne
570          { \int_use:N \c@g__tag_struct_abs_int }
571          { Lang }
572          { (#1) }
573      },
```

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

```
574    ref .code:n        = % ref property
575      {
576        \tl_clear:N\l__tag_tmpa_tl
577        \clist_map_inline:on {#1}
578          {
579            \tl_put_right:Ne \l__tag_tmpa_tl
580              {~\__tag_property_ref:en{tagpdfstruct-##1}{tagstructobj} }
581          }
582        \__tag_struct_gput_data_ref:ee
583          { \int_use:N \c@g__tag_struct_abs_int } {\l__tag_tmpa_tl}
584      },
585    E .code:n          = % E property
586      {
587        \str_set_convert:Nnon
588          \l__tag_tmpa_str
589          { #1 }
590          { default }
591          { utf16/hex }
592        \__tag_struct_prop_gput:nne
593          { \int_use:N \c@g__tag_struct_abs_int }
594          { E }
595          { <\l__tag_tmpa_str> }
596      },
597  }
```

(*End of definition for* `label` *(struct-key) and others. These functions are documented on page 96.*)

AF␣(struct-key)   keys for the AF keys (associated files). They use commands from l3pdffile! The stream
AFref␣(struct-key)   variants use txt as extension to get the mimetype. TODO: check if this should be
AFinline␣(struct-key)   configurable. For math we will perhaps need another extension. AF/AFref is an array
AFinline-o␣(struct-key)   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

```
598  \int_new:N\g__tag_struct_AFobj_int
```

```
599  \cs_generate_variant:Nn \pdffile_embed_stream:nnN {neN}
600  \cs_new_protected:Npn \__tag_struct_add_inline_AF:nn #1 #2
601  % #1 content, #2 extension
```

\g__tag_struct_AFobj_int

```
602  {
603    \tl_if_empty:nF{#1}
604      {
605        \group_begin:
606        \int_gincr:N \g__tag_struct_AFobj_int
607        \pdffile_embed_stream:neN
608          {#1}
609          {tag-AFfile\int_use:N\g__tag_struct_AFobj_int.#2}
610          \l__tag_tmpa_tl
611        \__tag_struct_add_AF:ee
612          { \int_use:N \c@g__tag_struct_abs_int }
613          { \l__tag_tmpa_tl }
614        \__tag_struct_prop_gput:nne
615          { \int_use:N \c@g__tag_struct_abs_int }
616          { AF }
617          {
618            [
619              \tl_use:c
620                { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
621            ]
622          }
623        \group_end:
624      }
625  }
626
627  \cs_generate_variant:Nn \__tag_struct_add_inline_AF:nn {on}
628  \cs_new_protected:Npn \__tag_struct_add_AF:nn #1 #2 % #1 struct num #2 object reference
629    {
630      \tl_if_exist:cTF
631        {
632          g__tag_struct_#1_AF_tl
633        }
634        {
635          \tl_gput_right:ce
636            { g__tag_struct_#1_AF_tl }
637            {  \c_space_tl #2 }
638        }
639        {
640          \tl_new:c
641            { g__tag_struct_#1_AF_tl }
642          \tl_gset:ce
643            { g__tag_struct_#1_AF_tl }
644            { #2 }
645        }
646    }
647  \cs_generate_variant:Nn \__tag_struct_add_AF:nn {en,ee}
648  \keys_define:nn { __tag / struct }
649  {
650    AF .code:n        = % AF property
651      {
652        \pdf_object_if_exist:eTF {#1}
653          {
654            \__tag_struct_add_AF:ee { \int_use:N \c@g__tag_struct_abs_int }{\pdf_object_ref:e
655            \__tag_struct_prop_gput:nne
```

115

```
656              { \int_use:N \c@g__tag_struct_abs_int }
657              { AF }
658              {
659                [
660                  \tl_use:c
661                    { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
662                ]
663              }
664          }
665          {
666              % message?
667          }
668      },
669    AFref .code:n        = % AF property
670      {
671        \tl_if_empty:eF {#1}
672          {
673            \__tag_struct_add_AF:ee { \int_use:N \c@g__tag_struct_abs_int }{#1}
674            \__tag_struct_prop_gput:nne
675              { \int_use:N \c@g__tag_struct_abs_int }
676              { AF }
677              {
678                [
679                  \tl_use:c
680                    { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
681                ]
682              }
683          }
684      },
685    ,AFinline .code:n =
686      {
687        \__tag_struct_add_inline_AF:nn {#1}{txt}
688      }
689    ,AFinline-o .code:n =
690      {
691        \__tag_struct_add_inline_AF:on {#1}{txt}
692      }
693    ,texsource .code:n =
694      {
695        \group_begin:
696        \pdfdict_put:nnn { l_pdffile/Filespec } {Desc}{(TeX~source)}
697        \pdfdict_put:nnn { l_pdffile/Filespec }{AFRelationship} { /Source }
698        \__tag_struct_add_inline_AF:on {#1}{tex}
699        \group_end:
700      }
701    ,mathml .code:n =
702      {
703        \group_begin:
704        \pdfdict_put:nnn { l_pdffile/Filespec } {Desc}{(mathml~representation)}
705        \pdfdict_put:nnn { l_pdffile/Filespec }{AFRelationship} { /Supplement }
706        \__tag_struct_add_inline_AF:on {#1}{xml}
707        \group_end:
708      }
709  }
```

116

(End of definition for AF *(struct-key) and others. These functions are documented on page 97.)*

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!

```
710 \keys_define:nn { __tag / setup }
711   {
712     root-AF .code:n =
713       {
714         \pdf_object_if_exist:nTF {#1}
715           {
716             \__tag_struct_add_AF:ee { 0 }{\pdf_object_ref:n {#1}}
717             \__tag_struct_prop_gput:nne
718             { 0 }
719             { AF }
720             {
721               [
722                 \tl_use:c
723                   { g__tag_struct_0_AF_tl }
724               ]
725             }
726           }
727           {
728
729           }
730       },
731   }
732 ⟨/package⟩
```

(End of definition for root-AF *(setup-key). This function is documented on page 98.)*

6 User commands

```
733 ⟨base⟩\cs_new_protected:Npn \tag_struct_begin:n #1 {\int_gincr:N \c@g__tag_struct_abs_int}
734 ⟨base⟩\cs_new_protected:Npn \tag_struct_end:{}
735 ⟨base⟩\cs_new_protected:Npn \tag_struct_end:n{}
736 ⟨*package | debug⟩
737 ⟨package⟩\cs_set_protected:Npn \tag_struct_begin:n #1 %#1 key-val
738 ⟨debug⟩\cs_set_protected:Npn \tag_struct_begin:n #1 %#1 key-val
739   {
740 ⟨package⟩\__tag_check_if_active_struct:T
741 ⟨debug⟩\__tag_check_if_active_struct:TF
742       {
743         \group_begin:
744         \int_gincr:N \c@g__tag_struct_abs_int
745         \__tag_prop_new:c  { g__tag_struct_\int_eval:n { \c@g__tag_struct_abs_int }_prop }
746 ⟨debug⟩          \prop_new:c { g__tag_struct_debug_\int_eval:n {\c@g__tag_struct_abs_int}_prop
747         \__tag_new_output_prop_handler:n {\int_eval:n { \c@g__tag_struct_abs_int }}
748         \__tag_seq_new:c  { g__tag_struct_kids_\int_eval:n { \c@g__tag_struct_abs_int }_seq}
749 ⟨debug⟩          \seq_new:c { g__tag_struct_debug_kids_\int_eval:n {\c@g__tag_struct_abs_int}_s
750         \exp_args:Ne
751           \pdf_object_new:n
752             { __tag/struct/\int_eval:n { \c@g__tag_struct_abs_int } }
```

117

```
753            \__tag_struct_prop_gput:nnn
754              { \int_use:N \c@g__tag_struct_abs_int }
755              { Type }
756              { /StructElem }
757          \tl_set:Nn \l__tag_struct_stack_parent_tmpa_tl {-1}
758          \keys_set:nn { __tag / struct} { #1 }

759          \__tag_struct_set_tag_info:eVV
760            { \int_use:N \c@g__tag_struct_abs_int }
761            \g__tag_struct_tag_tl
762            \g__tag_struct_tag_NS_tl
763          \__tag_check_structure_has_tag:n { \int_use:N \c@g__tag_struct_abs_int }
764          \tl_if_empty:NF
765            \l__tag_struct_key_label_tl
766            {
767              \__tag_property_record:eV
768              {tagpdfstruct-\l__tag_struct_key_label_tl}
769              \c__tag_property_struct_clist
770            }
```

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

```
771          \int_compare:nNnT { \l__tag_struct_stack_parent_tmpa_tl } = { -1 }
772            {
773              \seq_get:NNF
774                \g__tag_struct_stack_seq
775                \l__tag_struct_stack_parent_tmpa_tl
776                {
777                  \msg_error:nn { tag } { struct-faulty-nesting }
778                }
779            }
780          \seq_gpush:NV \g__tag_struct_stack_seq          \c@g__tag_struct_abs_int
781          \__tag_role_get:VVNN
782            \g__tag_struct_tag_tl
783            \g__tag_struct_tag_NS_tl
784            \l__tag_struct_roletag_tl
785            \l__tag_struct_roletag_NS_tl
```

to target role and role NS

```
786          \__tag_struct_prop_gput:nne
787            { \int_use:N \c@g__tag_struct_abs_int }
788            { rolemap }
789            {
790              {\l__tag_struct_roletag_tl}{\l__tag_struct_roletag_NS_tl}
791            }
```

we also store which role to use for parent/child test. If the role is one of Part, Div, NonStruct we have to retrieve it from the parent. If the structure is stashed, this must be updated!

```
792          \str_case:VnTF \l__tag_struct_roletag_tl
793            {
794            {Part} {}
795            {Div}  {}
796            {NonStruct} {}
797            }
```

118

```
798            {
799              \prop_get:cnNT
800                { g__tag_struct_ \l__tag_struct_stack_parent_tmpa_tl _prop }
801                { parentrole }
802                \l__tag_get_tmpc_tl
803                {
804                  \__tag_struct_prop_gput:nno
805                    { \int_use:N \c@g__tag_struct_abs_int }
806                    { parentrole }
807                    {
808                      \l__tag_get_tmpc_tl
809                    }
810                }
811            }
812            {
813                \__tag_struct_prop_gput:nne
814                  { \int_use:N \c@g__tag_struct_abs_int }
815                  { parentrole }
816                  {
817                    {\l__tag_struct_roletag_tl}{\l__tag_struct_roletag_NS_tl}
818                  }
819            }
820          \seq_gpush:Ne \g__tag_struct_tag_stack_seq
821            {{\g__tag_struct_tag_tl}{\l__tag_struct_roletag_tl}}
822          \tl_gset:NV   \g__tag_struct_stack_current_tl \c@g__tag_struct_abs_int
823          %\seq_show:N   \g__tag_struct_stack_seq
824          \bool_if:NF
825            \l__tag_struct_elem_stash_bool
826              {
```

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!

```
827                \__tag_struct_get_parentrole:eNN
828                  {\l__tag_struct_stack_parent_tmpa_tl}
829                  \l__tag_get_parent_tmpa_tl
830                  \l__tag_get_parent_tmpb_tl
831                \__tag_check_parent_child:VVVVN
832                  \l__tag_get_parent_tmpa_tl
833                  \l__tag_get_parent_tmpb_tl
834                  \g__tag_struct_tag_tl
835                  \g__tag_struct_tag_NS_tl
836                  \l__tag_parent_child_check_tl
837                \int_compare:nNnT {\l__tag_parent_child_check_tl}<0
838                  {
839                    \prop_get:cnN
840                      { g__tag_struct_ \l__tag_struct_stack_parent_tmpa_tl _prop}
841                      {S}
842                      \l__tag_tmpa_tl
843                    \msg_warning:nneee
844                    { tag }
845                    {role-parent-child}
846                    { \l__tag_get_parent_tmpa_tl/\l__tag_get_parent_tmpb_tl }
847                    { \g__tag_struct_tag_tl/\g__tag_struct_tag_NS_tl  }
```

119

```
848                        { not~allowed~
849                          (struct~\l__tag_struct_stack_parent_tmpa_tl,~\l__tag_tmpa_tl
850                           \c_space_tl-->~struct~\int_eval:n {\c@g__tag_struct_abs_int})
851                        }
852                      \cs_set_eq:NN \l__tag_role_remap_tag_tl \g__tag_struct_tag_tl
853                      \cs_set_eq:NN \l__tag_role_remap_NS_tl \g__tag_struct_tag_NS_tl
854                      \__tag_role_remap:
855                      \cs_gset_eq:NN \g__tag_struct_tag_tl \l__tag_role_remap_tag_tl
856                      \cs_gset_eq:NN  \g__tag_struct_tag_NS_tl \l__tag_role_remap_NS_tl
857                      \__tag_struct_set_tag_info:eVV
858                        { \int_use:N \c@g__tag_struct_abs_int }
859                          \g__tag_struct_tag_tl
860                          \g__tag_struct_tag_NS_tl
861                  }
```

Set the Parent.

```
862                      \__tag_struct_prop_gput:nne
863                      { \int_use:N \c@g__tag_struct_abs_int }
864                      { P }
865                      {
866                        \pdf_object_ref:e { __tag/struct/\l_tag_struct_stack_parent_tmpa_tl }
867                      }

868                %record this structure as kid:
869                %\tl_show:N \g__tag_struct_stack_current_tl
870                %\tl_show:N \l__tag_struct_stack_parent_tmpa_tl
871                \__tag_struct_kid_struct_gput_right:ee
872                    { \l__tag_struct_stack_parent_tmpa_tl }
873                    { \g__tag_struct_stack_current_tl }
874                %\prop_show:c { g__tag_struct_\g__tag_struct_stack_current_tl _prop }
875                %\seq_show:c {g__tag_struct_kids_\l__tag_struct_stack_parent_tmpa_tl _seq}
876            }
```

the debug mode stores in second prop and replaces value with more suitable ones. (If the structure is updated later this gets perhaps lost, but well ...) This must be done outside of the stash boolean.

```
877 ⟨debug⟩              \prop_gset_eq:cc
878 ⟨debug⟩                { g__tag_struct_debug_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
879 ⟨debug⟩                { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
880 ⟨debug⟩              \prop_gput:cne
881 ⟨debug⟩                { g__tag_struct_debug_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
882 ⟨debug⟩                { P }
883 ⟨debug⟩                {
884 ⟨debug⟩                   \bool_if:NTF \l__tag_struct_elem_stash_bool
885 ⟨debug⟩                     {no~parent:~stashed}
886 ⟨debug⟩                     {
887 ⟨debug⟩                        parent~structure:~\l__tag_struct_stack_parent_tmpa_tl\c_space_tl =~
888 ⟨debug⟩                        \l__tag_get_parent_tmpa_tl
889 ⟨debug⟩                     }
890 ⟨debug⟩                }
891 ⟨debug⟩              \prop_gput:cne
892 ⟨debug⟩                { g__tag_struct_debug_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
893 ⟨debug⟩                { NS }
894 ⟨debug⟩                { \g__tag_struct_tag_NS_tl }
```

```
895      %\prop_show:c { g__tag_struct_\g__tag_struct_stack_current_tl _prop }
896      %\seq_show:c {g__tag_struct_kids_\l__tag_struct_stack_parent_tmpa_tl _seq}
897 ⟨debug⟩ \__tag_debug_struct_begin_insert:n { #1 }
898          \group_end:
899        }
900 ⟨debug⟩{ \__tag_debug_struct_begin_ignore:n { #1 }}
901    }
902 ⟨package⟩\cs_set_protected:Nn \tag_struct_end:
903 ⟨debug⟩\cs_set_protected:Nn \tag_struct_end:
904    { %take the current structure num from the stack:
905      %the objects are written later, lua mode hasn't all needed info yet
906      %\seq_show:N \g__tag_struct_stack_seq
907 ⟨package⟩\__tag_check_if_active_struct:T
908 ⟨debug⟩\__tag_check_if_active_struct:TF
909        {
910          \seq_gpop:NN   \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
911          \seq_gpop:NNTF \g__tag_struct_stack_seq \l__tag_tmpa_tl
912            {
913              \__tag_check_info_closing_struct:o { \g__tag_struct_stack_current_tl }
914            }
915            { \__tag_check_no_open_struct: }
916          % get the previous one, shouldn't be empty as the root should be there
917          \seq_get:NNTF \g__tag_struct_stack_seq \l__tag_tmpa_tl
918            {
919              \tl_gset:NV   \g__tag_struct_stack_current_tl \l__tag_tmpa_tl
920            }
921            {
922              \__tag_check_no_open_struct:
923            }
924          \seq_get:NNT \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
925            {
926              \tl_gset:Ne \g__tag_struct_tag_tl
927                { \exp_last_unbraced:NV\use_i:nn \l__tag_tmpa_tl }
928              \prop_get:NVNT\g__tag_role_tags_NS_prop \g__tag_struct_tag_tl\l__tag_tmpa_tl
929                {
930                  \tl_gset:Ne \g__tag_struct_tag_NS_tl { \l__tag_tmpa_tl }
931                }
932            }
933 ⟨debug⟩\__tag_debug_struct_end_insert:
934        }
935 ⟨debug⟩{\__tag_debug_struct_end_ignore:}
936    }
937
938 \cs_set_protected:Npn \tag_struct_end:n #1
939  {
940 ⟨debug⟩     \__tag_check_if_active_struct:T{\__tag_debug_struct_end_check:n{#1}}
941    \tag_struct_end:
942  }
943 ⟨/package | debug⟩
```

(*End of definition for* \tag_struct_begin:n *and* \tag_struct_end:. *These functions are documented on page 95.*)

\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.

```
944 ⟨base⟩\cs_new_protected:Npn \tag_struct_use:n #1 {}
945 ⟨*package | debug⟩
946 \cs_set_protected:Npn \tag_struct_use:n #1 %#1 is the label
947   {
948     \__tag_check_if_active_struct:T
949       {
950         \prop_if_exist:cTF
951           { g__tag_struct_\__tag_property_ref:enn{tagpdfstruct-#1}{tagstruct}{unknown}_prop }
952           {
953             \__tag_check_struct_used:n {#1}
954             %add the label structure as kid to the current structure (can be the root)
955             \__tag_struct_kid_struct_gput_right:ee
956               { \g__tag_struct_stack_current_tl }
957               { \__tag_property_ref:enn{tagpdfstruct-#1}{tagstruct}{0} }
958             %add the current structure to the labeled one as parents
959             \__tag_prop_gput:cne
960               { g__tag_struct_\__tag_property_ref:enn{tagpdfstruct-#1}{tagstruct}{0}_prop }
961               { P }
962               {
963                 \pdf_object_ref:e { __tag/struct/\g__tag_struct_stack_current_tl }
964               }
```

debug code

```
965 ⟨debug⟩                \prop_gput:cne
966 ⟨debug⟩                  { g__tag_struct_debug_\__tag_property_ref:enn{tagpdfstruct-#1}{tagstruct}{
967 ⟨debug⟩                  { P }
968 ⟨debug⟩                  {
969 ⟨debug⟩                    parent~structure:~\g__tag_struct_stack_current_tl\c_space_tl=~
970 ⟨debug⟩                    \g__tag_struct_tag_tl
971 ⟨debug⟩                  }
```

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:

```
972                \__tag_struct_get_parentrole:eNN
973                  {\__tag_property_ref:enn{tagpdfstruct-#1}{tagstruct}{0}}
974                  \l__tag_tmpa_tl
975                  \l__tag_tmpb_tl
976                \__tag_check_parent_child:VVVVN
977                  \g__tag_struct_tag_tl
978                  \g__tag_struct_tag_NS_tl
979                  \l__tag_tmpa_tl
980                  \l__tag_tmpb_tl
981                  \l__tag_parent_child_check_tl
982                \int_compare:nNnT {\l__tag_parent_child_check_tl}<0
983                  {
984                    \cs_set_eq:NN \l__tag_role_remap_tag_tl \g__tag_struct_tag_tl
985                    \cs_set_eq:NN \l__tag_role_remap_NS_tl \g__tag_struct_tag_NS_tl
986                    \__tag_role_remap:
987                    \cs_gset_eq:NN \g__tag_struct_tag_tl \l__tag_role_remap_tag_tl
988                    \cs_gset_eq:NN  \g__tag_struct_tag_NS_tl \l__tag_role_remap_NS_tl
989                    \__tag_struct_set_tag_info:eVV
990                      { \int_use:N \c@g__tag_struct_abs_int }
991                      \g__tag_struct_tag_tl
992                      \g__tag_struct_tag_NS_tl
993                  }
```

122

```
994                    }
995                    {
996                        \msg_warning:nnn{ tag }{struct-label-unknown}{#1}
997                    }
998                }
999            }
1000 ⟨/package | debug⟩
```

(*End of definition for* \tag_struct_use:n. *This function is documented on page 95.*)

\tag_struct_use_num:n This command allows to use a stashed structure in another place. differently to the previous command it doesn't use a label but directly a structure number to find the parent. TODO: decide how it should be guarded. Probably by the struct-check.

```
1001 ⟨base⟩\cs_new_protected:Npn \tag_struct_use_num:n #1 {}
1002 ⟨*package | debug⟩
1003 \cs_set_protected:Npn \tag_struct_use_num:n #1 %#1 is structure number
1004    {
1005        \__tag_check_if_active_struct:T
1006          {
1007            \prop_if_exist:cTF
1008              { g__tag_struct_#1_prop } %
1009              {
1010                \prop_get:cnNT
1011                  {g__tag_struct_#1_prop}
1012                  {P}
1013                  \l__tag_tmpa_tl
1014                  {
1015                    \msg_warning:nnn { tag } {struct-used-twice} {#1}
1016                  }
1017                %add the \#1 structure as kid to the current structure (can be the root)
1018                \__tag_struct_kid_struct_gput_right:ee
1019                  { \g__tag_struct_stack_current_tl }
1020                  { #1 }
1021                %add the current structure to \#1 as parent
1022                 \__tag_struct_prop_gput:nne
1023                  { #1 }
1024                  { P }
1025                  {
1026                    \pdf_object_ref:e { __tag/struct/\g__tag_struct_stack_current_tl }
1027                  }
1028 ⟨debug⟩          \prop_gput:cne
1029 ⟨debug⟩            { g__tag_struct_debug_#1_prop }
1030 ⟨debug⟩            { P }
1031 ⟨debug⟩            {
1032 ⟨debug⟩               parent~structure:~\g__tag_struct_stack_current_tl\c_space_tl=~
1033 ⟨debug⟩               \g__tag_struct_tag_tl
1034 ⟨debug⟩            }
```

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:

```
1035                \__tag_struct_get_parentrole:eNN
1036                  {#1}
1037                  \l__tag_tmpa_tl
1038                  \l__tag_tmpb_tl
```

```
1039                    \__tag_check_parent_child:VVVVN
1040                      \g__tag_struct_tag_tl
1041                      \g__tag_struct_tag_NS_tl
1042                      \l__tag_tmpa_tl
1043                      \l__tag_tmpb_tl
1044                      \l__tag_parent_child_check_tl
1045                  \int_compare:nNnT {\l__tag_parent_child_check_tl}<0
1046                    {
1047                      \cs_set_eq:NN \l__tag_role_remap_tag_tl \g__tag_struct_tag_tl
1048                      \cs_set_eq:NN \l__tag_role_remap_NS_tl \g__tag_struct_tag_NS_tl
1049                      \__tag_role_remap:
1050                      \cs_gset_eq:NN \g__tag_struct_tag_tl \l__tag_role_remap_tag_tl
1051                      \cs_gset_eq:NN  \g__tag_struct_tag_NS_tl \l__tag_role_remap_NS_tl
1052                      \__tag_struct_set_tag_info:eVV
1053                        { \int_use:N \c@g__tag_struct_abs_int }
1054                          \g__tag_struct_tag_tl
1055                          \g__tag_struct_tag_NS_tl
1056                    }
1057              }
1058              {
1059                  \msg_warning:nnn{ tag }{struct-label-unknown}{#1}
1060              }
1061          }
1062      }
1063 ⟨/package | debug⟩
```

(*End of definition for* \tag_struct_use_num:n. *This function is documented on page 95.*)

\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.

```
1064 ⟨*package⟩
1065 \cs_new:Npn \tag_struct_object_ref:n #1
1066   {
1067     \pdf_object_ref:n {__tag/struct/#1}
1068   }
1069 \cs_generate_variant:Nn \tag_struct_object_ref:n {e}
```

(*End of definition for* \tag_struct_object_ref:n. *This function is documented on page 95.*)

\tag_struct_gput:nnn   This is a command that allows to update the data of a structure. This often can't done simply by replacing the value, as we have to preserve and extend existing content. We use therefore dedicated functions adjusted to the key in question. 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 which updates the Ref key (an array)

```
1070 \cs_new_protected:Npn \tag_struct_gput:nnn #1 #2 #3
1071   {
1072     \cs_if_exist_use:cF {__tag_struct_gput_data_#2:nn}
1073       { %warning??
1074         \use_none:nn
1075       }
1076       {#1}{#3}
1077   }
1078 \cs_generate_variant:Nn \tag_struct_gput:nnn {ene,nne}
1079 ⟨/package⟩
```

124

(End of definition for `\tag_struct_gput:nnn`*. This function is documented on page 96.)*

`\__tag_struct_gput_data_ref:nn`

```
1080 ⟨∗package⟩
1081 \cs_new_protected:Npn \__tag_struct_gput_data_ref:nn #1 #2
1082   % #1 receiving struct num, #2 list of object ref
1083   {
1084     \prop_get:cnN
1085       { g__tag_struct_#1_prop }
1086       {Ref}
1087       \l__tag_get_tmpc_tl
1088     \__tag_struct_prop_gput:nne
1089       { #1 }
1090       { Ref }
1091       { \quark_if_no_value:NF\l__tag_get_tmpc_tl { \l__tag_get_tmpc_tl\c_space_tl }#2 }
1092   }
1093 \cs_generate_variant:Nn \__tag_struct_gput_data_ref:nn {ee}
```

(End of definition for `\__tag_struct_gput_data_ref:nn`*.)*

`\tag_struct_insert_annot:nn`
`\tag_struct_insert_annot:ee`
`\tag_struct_insert_annot:ee`
`\tag_struct_parent_int:`

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.

```
1094 \cs_new_protected:Npn \tag_struct_insert_annot:nn #1 #2 %#1 should be an object reference
1095                                                          %#2 struct parent num
1096   {
1097     \__tag_check_if_active_struct:T
1098       {
1099         \__tag_struct_insert_annot:nn {#1}{#2}
1100       }
1101   }
1102
1103 \cs_generate_variant:Nn \tag_struct_insert_annot:nn {xx,ee}
1104 \cs_new:Npn \tag_struct_parent_int: {\int_use:c { c@g__tag_parenttree_obj_int }}
1105
1106 ⟨/package⟩
1107
```

(End of definition for `\tag_struct_insert_annot:nn` *and* `\tag_struct_parent_int:`*. These functions are documented on page 95.)*

7 Attributes and attribute classes

```
1108 ⟨∗header⟩
1109 \ProvidesExplPackage {tagpdf-attr-code} {2024-02-02} {0.98u}
1110   {part of tagpdf - code related to attributes and attribute classes}
1111 ⟨/header⟩
```

7.1 Variables

`\g__tag_attr_entries_prop`
`\g__tag_attr_class_used_seq`
`\g__tag_attr_objref_prop`
`\l__tag_attr_value_tl`

`\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

125

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`

```
1112 ⟨*package⟩
1113 \prop_new:N \g__tag_attr_entries_prop
1114 \seq_new:N  \g__tag_attr_class_used_seq
1115 \tl_new:N   \l__tag_attr_value_tl
1116 \prop_new:N \g__tag_attr_objref_prop %will contain obj num of used attributes
```

(*End of 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.

    TODO: consider to put them directly in the ClassMap, that is perhaps more effective.

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

```
1117 \cs_new_protected:Npn \__tag_attr_new_entry:nn #1 #2 %#1:name, #2: content
1118   {
1119     \prop_gput:Nen \g__tag_attr_entries_prop
1120       {\pdf_name_from_unicode_e:n{#1}}{#2}
1121   }
1122
1123 \keys_define:nn { __tag / setup }
1124   {
1125     newattribute .code:n =
1126       {
1127          \__tag_attr_new_entry:nn #1
1128       }
1129   }
```

(*End of definition for* `\__tag_attr_new_entry:nn` *and* `newattribute` *(setup-key). This function is documented on page 98.*)

`attribute-class␣(struct-key)`   attribute-class has to store the used attribute names so that they can be added to the ClassMap later.

```
1130 \keys_define:nn { __tag / struct }
1131   {
1132     attribute-class .code:n =
1133       {
1134          \clist_set:Ne \l__tag_tmpa_clist { #1 }
1135          \seq_set_from_clist:NN \l__tag_tmpb_seq \l__tag_tmpa_clist
```

126

we convert the names into pdf names with slash

```
1136          \seq_set_map_e:NNn \l__tag_tmpa_seq \l__tag_tmpb_seq
1137            {
1138              \pdf_name_from_unicode_e:n {##1}
1139            }
1140          \seq_map_inline:Nn \l__tag_tmpa_seq
1141            {
1142              \prop_if_in:NnF \g__tag_attr_entries_prop {##1}
1143                {
1144                  \msg_error:nnn { tag } { attr-unknown } { ##1 }
1145                }
1146              \seq_gput_left:Nn\g__tag_attr_class_used_seq { ##1 }
1147            }
1148          \tl_set:Ne \l__tag_tmpa_tl
1149            {
1150              \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[}
1151              \seq_use:Nn \l__tag_tmpa_seq  { \c_space_tl  }
1152              \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{]}
1153            }
1154          \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 0 }
1155            {
1156              \__tag_struct_prop_gput:nne
1157                { \int_use:N \c@g__tag_struct_abs_int }
1158                { C }
1159                { \l__tag_tmpa_tl }
1160             %\prop_show:c  { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
1161            }
1162        }
1163    }
```

(*End of definition for* `attribute-class` (struct-key)*. This function is documented on page 98.*)

attribute␣(struct-key)

```
1164 \keys_define:nn { __tag / struct }
1165   {
1166     attribute .code:n  = % A property (attribute, value currently a dictionary)
1167       {
1168          \clist_set:Ne            \l__tag_tmpa_clist { #1 }
1169          \clist_if_empty:NF \l__tag_tmpa_clist
1170          {
1171              \seq_set_from_clist:NN \l__tag_tmpb_seq \l__tag_tmpa_clist
```

we convert the names into pdf names with slash

```
1172              \seq_set_map_e:NNn \l__tag_tmpa_seq \l__tag_tmpb_seq
1173                {
1174                  \pdf_name_from_unicode_e:n {##1}
1175                }
1176              \tl_set:Ne \l__tag_attr_value_tl
1177                {
1178                  \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[}%]
1179                }
1180              \seq_map_inline:Nn \l__tag_tmpa_seq
1181                {
1182                  \prop_if_in:NnF \g__tag_attr_entries_prop {##1}
```

127

```
1183                     {
1184                         \msg_error:nnn { tag } { attr-unknown } { ##1 }
1185                     }
1186                 \prop_if_in:NnF \g__tag_attr_objref_prop {##1}
1187                   {%\prop_show:N \g__tag_attr_entries_prop
1188                     \pdf_object_unnamed_write:ne
1189                       { dict }
1190                       {
1191                         \prop_item:Nn\g__tag_attr_entries_prop {##1}
1192                       }
1193                     \prop_gput:Nne \g__tag_attr_objref_prop {##1} {\pdf_object_ref_last:}
1194                   }
1195                 \tl_put_right:Ne \l__tag_attr_value_tl
1196                   {
1197                     \c_space_tl
1198                     \prop_item:Nn \g__tag_attr_objref_prop {##1}
1199                   }
1200        %      \tl_show:N \l__tag_attr_value_tl
1201                 }
1202            \tl_put_right:Ne \l__tag_attr_value_tl
1203              { %[
1204                \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{]}%
1205              }
1206        %      \tl_show:N \l__tag_attr_value_tl
1207             \__tag_struct_prop_gput:nne
1208               { \int_use:N \c@g__tag_struct_abs_int }
1209               { A }
1210               { \l__tag_attr_value_tl }
1211         }
1212     },
1213   }
1214 ⟨/package⟩
```

(*End of definition for* `attribute` (struct-key). *This function is documented on page* *98*.)

Part VIII
The **tagpdf-luatex.def** Driver for luatex
Part of the tagpdf package

```
1 ⟨@@=tag⟩
2 ⟨∗luatex⟩
3 \ProvidesExplFile {tagpdf-luatex.def} {2024-02-02} {0.98u}
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 of 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       = "0.98u",       --TAGVERSION
70      date          = "2024-02-02", --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 of
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 fonthashes      = fonts.hashes
152 local identifiers     = fonthashes.identifiers
153 local fontid          = font.id
154
155 local HLIST           = node.id("hlist")
156 local VLIST           = node.id("vlist")
157 local RULE            = node.id("rule")
158 local DISC            = node.id("disc")
159 local GLUE            = node.id("glue")
160 local GLYPH           = node.id("glyph")
161 local KERN            = node.id("kern")
162 local PENALTY         = node.id("penalty")
163 local LOCAL_PAR       = node.id("local_par")
164 local MATH            = node.id("math")
```

Now we setup the main table structure. ltx is used by other latex code too!

```
165 ltx              = ltx          or { }
166 ltx.__tag        = ltx.__tag        or { }
167 ltx.__tag.mc     = ltx.__tag.mc     or { } -- mc data
168 ltx.__tag.struct = ltx.__tag.struct or { } -- struct data
169 ltx.__tag.tables = ltx.__tag.tables or { } -- tables created with new prop and new seq.
170                                       -- wasn't a so great idea ...
```

```
171                                              -- g__tag_role_tags_seq used by tag<-> is in this tabl
172                                              -- used for pure lua tables too now!
173 ltx.__tag.page     = ltx.__tag.page   or  { } -- page data, currently only i->{0->mcnum,1->mcr
174 ltx.__tag.trace    = ltx.__tag.trace  or  { } -- show commands
175 ltx.__tag.func     = ltx.__tag.func   or  { } -- functions
176 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.

```
177 local __tag_log =
178  function (message,loglevel)
179   if (loglevel or 3) <= tex.count["l__tag_loglevel_int"] then
180    texio.write_nl("tagpdf: ".. message)
181   end
182  end
183
184 ltx.__tag.trace.log = __tag_log
```

(*End of 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.

```
185 function ltx.__tag.trace.show_seq (seq)
186  if (type(seq) == "table") then
187   for i,v in ipairs(seq) do
188    __tag_log ("[" .. i .. "] => " .. tostring(v),1)
189   end
190  else
191   __tag_log ("sequence " .. tostring(seq) .. " not found",1)
192  end
193 end
```

(*End of 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.

```
194 local __tag_pairs_prop =
195  function  (prop)
196      local a = {}
197      for n in pairs(prop) do tableinsert(a, n) end
198      table.sort(a)
199      local i = 0                -- iterator variable
200      local iter = function ()   -- iterator function
201        i = i + 1
202        if a[i] == nil then return nil
203        else return a[i], prop[a[i]]
204        end
205      end
206      return iter
```

133

```
207    end
208
209
210  function ltx.__tag.trace.show_prop (prop)
211   if (type(prop) == "table") then
212    for i,v in __tag_pairs_prop (prop) do
213      __tag_log ("[" .. i .. "] => " .. tostring(v),1)
214    end
215   else
216     __tag_log ("prop " .. tostring(prop) .. " not found or not a table",1)
217   end
218   end
```

(*End of 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

```
219  function ltx.__tag.trace.show_mc_data (num,loglevel)
220   if ltx.__tag and ltx.__tag.mc and ltx.__tag.mc[num] then
221    for k,v in pairs(ltx.__tag.mc[num]) do
222     __tag_log ("mc"..num..": "..tostring(k).."=>"..tostring(v),loglevel)
223    end
224    if ltx.__tag.mc[num]["kids"] then
225    __tag_log ("mc" .. num .. " has " .. #ltx.__tag.mc[num]["kids"] .. " kids",loglevel)
226     for k,v in ipairs(ltx.__tag.mc[num]["kids"]) do
227      __tag_log ("mc ".. num .. " kid "..k.." =>" .. v.kid.." on page " ..v.page,loglevel)
228     end
229    end
230   else
231    __tag_log ("mc"..num.." not found",loglevel)
232   end
233   end
```

(*End of 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.

```
234  function ltx.__tag.trace.show_all_mc_data (min,max,loglevel)
235   for i = min, max do
236    ltx.__tag.trace.show_mc_data (i,loglevel)
237   end
238   texio.write_nl("")
239   end
```

(*End of 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.

```
240  function ltx.__tag.trace.show_struct_data (num)
241   if ltx.__tag and ltx.__tag.struct and ltx.__tag.struct[num] then
242    for k,v in ipairs(ltx.__tag.struct[num]) do
243     __tag_log ("struct "..num..": "..tostring(k).."=>"..tostring(v),1)
244    end
245   else
246    __tag_log ("struct "..num.." not found ",1)
247   end
248   end
```

134

(*End of 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.

```
249 local __tag_get_mc_cnt_type_tag = function (n)
250   local mccnt      =  nodegetattribute(n,mccntattributeid)  or -1
251   local mctype     =  nodegetattribute(n,mctypeattributeid)  or -1
252   local tag        =  ltx.__tag.func.get_tag_from(mctype)
253   return mccnt,mctype,tag
254 end
```

(*End of 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.

```
255 local function __tag_get_mathsubtype  (mathnode)
256   if mathnode.subtype == 0 then
257    subtype = "beginmath"
258   else
259    subtype = "endmath"
260   end
261   return subtype
262 end
```

(*End of 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.

```
263 ltx.__tag.tables.role_tag_attribute = {}
264 ltx.__tag.tables.role_attribute_tag = {}
```

(*End of definition for* `ltx.__tag.tables.role_tag_attribute`.)

`ltx.__tag.func.alloctag`

```
265 local __tag_alloctag =
266  function (tag)
267    if not ltx.__tag.tables.role_tag_attribute[tag] then
268     table.insert(ltx.__tag.tables.role_attribute_tag,tag)
269     ltx.__tag.tables.role_tag_attribute[tag]=#ltx.__tag.tables.role_attribute_tag
270     __tag_log  ("Add "..tag.." "..ltx.__tag.tables.role_tag_attribute[tag],3)
271    end
272  end
273 ltx.__tag.func.alloctag = __tag_alloctag
```

(*End of definition for* `ltx.__tag.func.alloctag`.)

135

__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.

```
274 local __tag_get_num_from =
275  function (tag)
276   if ltx.__tag.tables.role_tag_attribute[tag] then
277     a= ltx.__tag.tables.role_tag_attribute[tag]
278   else
279     a= -1
280   end
281   return a
282  end
283
284 ltx.__tag.func.get_num_from = __tag_get_num_from
285
286 function ltx.__tag.func.output_num_from (tag)
287   local num = __tag_get_num_from (tag)
288   tex.sprint(catlatex,num)
289   if num == -1 then
290     __tag_log ("Unknown tag "..tag.." used")
291   end
292 end
```

(*End of 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.

```
293 local __tag_get_tag_from =
294  function  (num)
295   if ltx.__tag.tables.role_attribute_tag[num] then
296    a = ltx.__tag.tables.role_attribute_tag[num]
297   else
298    a= "UNKNOWN"
299   end
300  return a
301 end
302
303 ltx.__tag.func.get_tag_from = __tag_get_tag_from
304
305 function ltx.__tag.func.output_tag_from (num)
306   tex.sprint(catlatex,__tag_get_tag_from (num))
307 end
```

(*End of definition for* __tag_get_tag_from, ltx.__tag.func.get_tag_from, *and* ltx.__tag.func.output_-tag_from.)

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.

```
308 function ltx.__tag.func.store_mc_data (num,key,data)
309  ltx.__tag.mc[num] = ltx.__tag.mc[num] or { }
310  ltx.__tag.mc[num][key] = data
311  __tag_log  ("INFO TEX-STORE-MC-DATA: "..num.." => "..tostring(key).." => "..tostring(data),3)
312 end
```

136

(*End of 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.

```
313 function ltx.__tag.func.store_mc_label (label,num)
314   ltx.__tag.mc["labels"] = ltx.__tag.mc["labels"] or { }
315   ltx.__tag.mc.labels[label] = num
316 end
```

(*End of 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.

```
317 function ltx.__tag.func.store_mc_kid (mcnum,kid,page)
318   ltx.__tag.trace.log("INFO TAG-STORE-MC-KID: "..mcnum.." => " .. kid.." on page " .. page,3)
319   ltx.__tag.mc[mcnum]["kids"] = ltx.__tag.mc[mcnum]["kids"] or { }
320   local kidtable = {kid=kid,page=page}
321   tableinsert(ltx.__tag.mc[mcnum]["kids"], kidtable )
322 end
```

(*End of 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.

```
323 function ltx.__tag.func.mc_num_of_kids (mcnum)
324   local num = 0
325   if ltx.__tag.mc[mcnum] and ltx.__tag.mc[mcnum]["kids"] then
326     num = #ltx.__tag.mc[mcnum]["kids"]
327   end
328   ltx.__tag.trace.log ("INFO MC-KID-NUMBERS: " .. mcnum .. "has " .. num .. "KIDS",4)
329   return num
330 end
```

(*End of definition for* `ltx.__tag.func.mc_num_of_kids`.)

3.2 Functions to insert the pdf literals

__tag_backend_create_emc_node
__tag_insert_emc_node
This insert the emc node. We support also dvips and dvipdfmx backend

```
331 local __tag_backend_create_emc_node
332 if tex.outputmode == 0 then
333   if token.get_macro("c_sys_backend_str") == "dvipdfmx" then
334     function __tag_backend_create_emc_node ()
335       local emcnode = nodenew("whatsit","special")
336         emcnode.data = "pdf:code EMC"
337       return emcnode
338     end
339   else -- assume a dvips variant
340     function __tag_backend_create_emc_node ()
341       local emcnode = nodenew("whatsit","special")
342         emcnode.data = "ps:SDict begin mark /EMC pdfmark end"
343       return emcnode
344     end
345   end
346 else -- pdf mode
```

```
347   function __tag_backend_create_emc_node ()
348     local emcnode = nodenew("whatsit","pdf_literal")
349       emcnode.data = "EMC"
350       emcnode.mode=1
351     return emcnode
352   end
353 end
354
355 local function __tag_insert_emc_node (head,current)
356   local emcnode= __tag_backend_create_emc_node()
357   head = node.insert_before(head,current,emcnode)
358   return head
359 end
```

(*End of definition for* __tag_backend_create_emc_node *and* __tag_insert_emc_node.)

This inserts a simple bmc node

```
360 local __tag_backend_create_bmc_node
361 if tex.outputmode == 0 then
362  if token.get_macro("c_sys_backend_str") == "dvipdfmx" then
363   function __tag_backend_create_bmc_node (tag)
364     local bmcnode = nodenew("whatsit","special")
365     bmcnode.data = "pdf:code /"..tag.." BMC"
366     return bmcnode
367   end
368  else -- assume a dvips variant
369   function __tag_backend_create_bmc_node (tag)
370     local bmcnode = nodenew("whatsit","special")
371     bmcnode.data = "ps:SDict begin mark/"..tag.." BMC pdfmark end"
372     return bmcnode
373   end
374  end
375 else -- pdf mode
376   function __tag_backend_create_bmc_node (tag)
377     local bmcnode = nodenew("whatsit","pdf_literal")
378     bmcnode.data = "/"..tag.." BMC"
379     bmcnode.mode=1
380     return bmcnode
381   end
382 end
383
384 local function __tag_insert_bmc_node (head,current,tag)
385   local bmcnode = __tag_backend_create_bmc_node (tag)
386   head = node.insert_before(head,current,bmcnode)
387   return head
388 end
```

(*End of definition for* __tag_backend_create_bmc_node *and* __tag_insert_bmc_node.)

This inserts a bcd node with a fix dict. TODO: check if this is still used, now that we create properties.

```
389 local __tag_backend_create_bdc_node
390
391 if tex.outputmode == 0 then
```

138

```
392  if token.get_macro("c_sys_backend_str") == "dvipdfmx" then
393   function __tag_backend_create_bdc_node (tag,dict)
394     local bdcnode = nodenew("whatsit","special")
395     bdcnode.data = "pdf:code /"..tag.."<<"..dict..">> BDC"
396     return bdcnode
397   end
398  else -- assume a dvips variant
399   function __tag_backend_create_bdc_node (tag,dict)
400     local bdcnode = nodenew("whatsit","special")
401     bdcnode.data = "ps:SDict begin mark/"..tag.."<<"..dict..">> BDC pdfmark end"
402     return bdcnode
403   end
404  end
405 else -- pdf mode
406   function __tag_backend_create_bdc_node (tag,dict)
407     local bdcnode = nodenew("whatsit","pdf_literal")
408     bdcnode.data = "/"..tag.."<<"..dict..">> BDC"
409     bdcnode.mode=1
410     return bdcnode
411   end
412 end
413
414 local function __tag_insert_bdc_node (head,current,tag,dict)
415   bdcnode= __tag_backend_create_bdc_node (tag,dict)
416   head = node.insert_before(head,current,bdcnode)
417   return head
418 end
```

(*End of definition for* `__tag_backend_create_bdc_node` *and* `__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 $n_{\sqcup}0_{\sqcup}R$, if the object doesn't exist, n is 0. TODO: is uses internal l3pdf commands, this should be properly supported by l3pdf

```
419 local function __tag_pdf_object_ref (name)
420   local tokenname = 'c__pdf_backend_object_'..name..'_int'
421   local object = token.create(tokenname).index..' 0 R'
422   return object
423 end
424 ltx.__tag.func.pdf_object_ref=__tag_pdf_object_ref
```

(*End of 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.

```
425 local function __tag_show_spacemark (head,current,color,height)
426   local markcolor = color or "1 0 0"
427   local markheight = height or 10
428   local pdfstring
429   if tex.outputmode == 0 then
430    -- ignore dvi mode for now
431   else
```

```
432   pdfstring = node.new("whatsit","pdf_literal")
433       pdfstring.data =
434       string.format("q "..markcolor.." RG "..markcolor.." rg 0.4 w 0 %g m 0 %g l S Q",-
   3,markheight)
435       head = node.insert_after(head,current,pdfstring)
436   return head
437  end
438 end
```

(*End of definition for* `__tag_show_spacemark.`)

`__tag_fakespace`
`ltx.__tag.func.fakespace`

This is used to define a lua version of `\pdffakespace`

```
439 local function __tag_fakespace()
440     tex.setattribute(iwspaceattributeid,1)
441     tex.setattribute(iwfontattributeid,font.current())
442 end
443 ltx.__tag.func.fakespace = __tag_fakespace
```

(*End of 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.

```
444 --[[ a function to mark up places where real space chars should be inserted
445     it only sets an attribute.
446 --]]
447
448 local function __tag_mark_spaces (head)
449   local inside_math = false
450   for n in nodetraverse(head) do
451     local id = n.id
452     if id == GLYPH then
453       local glyph = n
454       default_currfontid = glyph.font
455       if glyph.next and (glyph.next.id == GLUE)
456         and not inside_math  and (glyph.next.width >0)
457       then
458         nodesetattribute(glyph.next,iwspaceattributeid,1)
459         nodesetattribute(glyph.next,iwfontattributeid,glyph.font)
460       -- for debugging
461       if ltx.__tag.trace.showspaces then
462         __tag_show_spacemark (head,glyph)
463       end
464     elseif glyph.next and (glyph.next.id==KERN) and not inside_math then
465       local kern = glyph.next
466       if kern.next and (kern.next.id== GLUE)  and (kern.next.width >0)
467       then
468        nodesetattribute(kern.next,iwspaceattributeid,1)
469        nodesetattribute(kern.next,iwfontattributeid,glyph.font)
470       end
471      end
472     --  look also back
473     if glyph.prev and (glyph.prev.id == GLUE)
474         and not inside_math
```

```
475        and (glyph.prev.width >0)
476        and not nodehasattribute(glyph.prev,iwspaceattributeid)
477      then
478        nodesetattribute(glyph.prev,iwspaceattributeid,1)
479        nodesetattribute(glyph.prev,iwfontattributeid,glyph.font)
480      -- for debugging
481       if ltx.__tag.trace.showspaces then
482        __tag_show_spacemark (head,glyph)
483       end
484      end
485    elseif id == PENALTY then
486      local glyph = n
487      -- ltx.__tag.trace.log ("PENALTY ".. n.subtype.."VALUE"..n.penalty,3)
488      if glyph.next and (glyph.next.id == GLUE)
489        and not inside_math  and (glyph.next.width >0) and n.subtype==0
490      then
491        nodesetattribute(glyph.next,iwspaceattributeid,1)
492        --  changed 2024-01-18, issue #72
493        nodesetattribute(glyph.next,iwfontattributeid,default_currfontid)
494      -- for debugging
495       if ltx.__tag.trace.showspaces then
496        __tag_show_spacemark (head,glyph)
497       end
498      end
499    elseif id == MATH then
500      inside_math = (n.subtype == 0)
501    end
502   end
503   return head
504 end
```

(*End of definition for* __tag_mark_spaces.)

Theses functions add/remove the function which marks the spaces to the callbacks pre_linebreak_filter and hpack_filter

```
505 local function __tag_activate_mark_space ()
506  if not luatexbase.in_callback ("pre_linebreak_filter","markspaces") then
507   luatexbase.add_to_callback("pre_linebreak_filter",__tag_mark_spaces,"markspaces")
508   luatexbase.add_to_callback("hpack_filter",__tag_mark_spaces,"markspaces")
509  end
510 end
511
512 ltx.__tag.func.markspaceon=__tag_activate_mark_space
513
514 local function __tag_deactivate_mark_space ()
515  if luatexbase.in_callback ("pre_linebreak_filter","markspaces") then
516   luatexbase.remove_from_callback("pre_linebreak_filter","markspaces")
517   luatexbase.remove_from_callback("hpack_filter","markspaces")
518  end
519 end
520
521 ltx.__tag.func.markspaceoff=__tag_deactivate_mark_space
```

(*End of 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.

```
522 local default_space_char = nodenew(GLYPH)
523 local default_fontid      = fontid("TU/lmr/m/n/10")
524 local default_currfontid = fontid("TU/lmr/m/n/10")
525 default_space_char.char  = 32
526 default_space_char.font  = default_fontid
```

And a function to check as best as possible if a font has a space:

```
527 local function __tag_font_has_space (fontid)
528 t= fonts.hashes.identifiers[fontid]
529 if luaotfload.aux.slot_of_name(fontid,"space")
530    or t.characters and t.characters[32] and t.characters[32]["unicode"]==32
531 then
532    return true
533 else
534    return false
535 end
536 end
```

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.

```
537 local function __tag_space_chars_shipout (box)
538 local head = box.head
539   if head then
540     for n in node.traverse(head) do
541       local spaceattr = nodegetattribute(n,iwspaceattributeid)  or -1
542       if n.id == HLIST  then -- enter the hlist
543           __tag_space_chars_shipout (n)
544       elseif n.id == VLIST then -- enter the vlist
545           __tag_space_chars_shipout (n)
546       elseif n.id == GLUE then
547         if ltx.__tag.trace.showspaces and spaceattr==1  then
548           __tag_show_spacemark (head,n,"0 1 0")
549         end
550         if spaceattr==1  then
551           local space
552           local space_char = node.copy(default_space_char)
553           local curfont    = nodegetattribute(n,iwfontattributeid)
554           ltx.__tag.trace.log ("INFO SPACE-FUNCTION-FONT: ".. tostring(curfont),3)
555           if curfont and
556             -- luaotfload.aux.slot_of_name(curfont,"space")
557             __tag_font_has_space (curfont)
558           then
559             space_char.font=curfont
560           end
561           head, space = node.insert_before(head, n, space_char) --
562           n.width     = n.width - space.width
563           space.attr  = n.attr
564         end
565       end
566     end
567     box.head = head
568   end
```

142

```
569 end
570
571 function ltx.__tag.func.space_chars_shipout (box)
572   __tag_space_chars_shipout (box)
573 end
```

(*End of 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.

```
574 function ltx.__tag.func.mc_insert_kids (mcnum,single)
575   if ltx.__tag.mc[mcnum] then
576   ltx.__tag.trace.log("INFO TEX-MC-INSERT-KID-TEST: " .. mcnum,4)
577    if ltx.__tag.mc[mcnum]["kids"] then
578     if #ltx.__tag.mc[mcnum]["kids"] > 1 and single==1 then
579      tex.sprint("[")
580     end
581     for i,kidstable in ipairs( ltx.__tag.mc[mcnum]["kids"] ) do
582      local kidnum  = kidstable["kid"]
583      local kidpage = kidstable["page"]
584      local kidpageobjnum = pdfpageref(kidpage)
585      ltx.__tag.trace.log("INFO TEX-MC-INSERT-KID: " .. mcnum ..
586                          " insert KID " ..i..
587                          " with num " .. kidnum ..
588                          " on page " .. kidpage.."/"..kidpageobjnum,3)
589     tex.sprint(catlatex,"<</Type /MCR /Pg "..kidpageobjnum .. " 0 R /MCID "..kidnum.. ">> " )
590     end
591     if #ltx.__tag.mc[mcnum]["kids"] > 1 and single==1 then
592      tex.sprint("]")
593     end
594    else
595     -- this is typically not a problem, e.g. empty hbox in footer/header can
596     -- trigger this warning.
597     ltx.__tag.trace.log("WARN TEX-MC-INSERT-NO-KIDS: "..mcnum.." has no kids",2)
598     if single==1 then
599       tex.sprint("null")
600     end
601    end
602   else
603    ltx.__tag.trace.log("WARN TEX-MC-INSERT-MISSING: "..mcnum.." doesn't exist",0)
604   end
605 end
```

(*End of 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.

```
606 function ltx.__tag.func.store_struct_mcabs (structnum,mcnum)
607  ltx.__tag.struct[structnum]=ltx.__tag.struct[structnum] or { }
```

```
608  ltx.__tag.struct[structnum]["mc"]=ltx.__tag.struct[structnum]["mc"] or { }
609  -- a structure can contain more than on mc chunk, the content should be ordered
610  tableinsert(ltx.__tag.struct[structnum]["mc"],mcnum)
611  ltx.__tag.trace.log("INFO TEX-MC-INTO-STRUCT: "..
612                  mcnum.." inserted in struct "..structnum,3)
613  -- but every mc can only be in one structure
614  ltx.__tag.mc[mcnum]= ltx.__tag.mc[mcnum] or { }
615  ltx.__tag.mc[mcnum]["parent"] = structnum
616  end
617
```

(*End of 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.

```
618  -- pay attention: lua counts arrays from 1, tex pages from one
619  -- mcid and arrays in pdf count from 0.
620  function ltx.__tag.func.store_mc_in_page (mcnum,mcpagecnt,page)
621  ltx.__tag.page[page] = ltx.__tag.page[page] or {}
622  ltx.__tag.page[page][mcpagecnt] = mcnum
623  ltx.__tag.trace.log("INFO TAG-MC-INTO-PAGE: page " .. page ..
624                  ": inserting MCID " .. mcpagecnt .. " => " .. mcnum,3)
625  end
```

(*End of definition for* `ltx.__tag.func.store_mc_in_page`.)

`ltx.__tag.func.update_mc_attributes` This updates the mc-attributes of a box. It should only be used on boxes which don't contain structure elements. The arguments are a box, the mc-num and the type (as a number)

```
626  local function __tag_update_mc_attributes (head,mcnum,type)
627  for n in node.traverse(head) do
628    node.set_attribute(n,mccntattributeid,mcnum)
629    node.set_attribute(n,mctypeattributeid,type)
630    if n.id == HLIST or n.id == VLIST then
631      __tag_update_mc_attributes (n.list,mcnum,type)
632    end
633  end
634  return head
635  end
636  ltx.__tag.func.update_mc_attributes = __tag_update_mc_attributes
```

(*End of definition for* `ltx.__tag.func.update_mc_attributes`.)

`ltx.__tag.func.mark_page_elements` This is the main traversing function. See the lua comment for more details.

```
637  --[[
638      Now follows the core function
639      It wades through the shipout box and checks the attributes
640      ARGUMENTS
641      box: is a box,
642      mcpagecnt: num, the current page cnt of mc (should start at -1 in shipout box), needed fo
643      mccntprev: num, the attribute cnt of the previous node/whatever - if different we have a
644      mcopen: num, records if some bdc/emc is open
645      These arguments are only needed for log messages, if not present are replaces by fix strin
646      name: string to describe the box
```

144

```
647    mctypeprev: num, the type attribute of the previous node/whatever

649    there are lots of logging messages currently. Should be cleaned up in due course.
650    One should also find ways to make the function shorter.
651 --]]

653 function ltx.__tag.func.mark_page_elements (box,mcpagecnt,mccntprev,mcopen,name,mctypeprev)
654   local name = name or ("SOMEBOX")
655   local mctypeprev = mctypeprev or -1
656   local abspage = status.total_pages + 1  -- the real counter is increased
657                                           -- inside the box so one off
658                                           -- if the callback is not used. (???)
659   ltx.__tag.trace.log ("INFO TAG-ABSPAGE: " .. abspage,3)
660   ltx.__tag.trace.log ("INFO TAG-ARGS: pagecnt".. mcpagecnt..
661                     " prev "..mccntprev ..
662                     " type prev "..mctypeprev,4)
663   ltx.__tag.trace.log ("INFO TAG-TRAVERSING-BOX: ".. tostring(name)..
664                     " TYPE ".. node.type(node.getid(box)),3)
665   local head = box.head -- ShipoutBox is a vlist?
666   if head then
667     mccnthead, mctypehead,taghead = __tag_get_mc_cnt_type_tag (head)
668     ltx.__tag.trace.log ("INFO TAG-HEAD: " ..
669                     node.type(node.getid(head))..
670                     " MC"..tostring(mccnthead)..
671                     " => TAG " .. tostring(mctypehead)..
672                     " => ".. tostring(taghead),3)
673   else
674     ltx.__tag.trace.log ("INFO TAG-NO-HEAD: head is "..
675                      tostring(head),3)
676   end
677   for n in node.traverse(head) do
678     local mccnt, mctype, tag = __tag_get_mc_cnt_type_tag (n)
679     local spaceattr = nodegetattribute(n,iwspaceattributeid)  or -1
680     ltx.__tag.trace.log ("INFO TAG-NODE: "..
681                      node.type(node.getid(n))..
682                     " MC".. tostring(mccnt)..
683                     " => TAG ".. tostring(mctype)..
684                     " => " ..  tostring(tag),3)
685     if n.id == HLIST
686     then -- enter the hlist
687      mcopen,mcpagecnt,mccntprev,mctypeprev=
688       ltx.__tag.func.mark_page_elements (n,mcpagecnt,mccntprev,mcopen,"INTERNAL HLIST",mctype
689     elseif n.id == VLIST then -- enter the vlist
690      mcopen,mcpagecnt,mccntprev,mctypeprev=
691       ltx.__tag.func.mark_page_elements (n,mcpagecnt,mccntprev,mcopen,"INTERNAL VLIST",mctype
692     elseif n.id == GLUE and not n.leader then -- at glue real space chars are inserted, but t
693                                        -- been done if the previous shipout wandering, so here it
694     elseif n.id == LOCAL_PAR then  -- local_par is ignored
695     elseif n.id == PENALTY then    -- penalty is ignored
696     elseif n.id == KERN then       -- kern is ignored
697      ltx.__tag.trace.log ("INFO TAG-KERN-SUBTYPE: "..
698        node.type(node.getid(n)).." "..n.subtype,4)
699     else
700      -- math is currently only logged.
```

145

```lua
701      -- we could mark the whole as math
702      -- for inner processing the mlist_to_hlist callback is probably needed.
703      if n.id == MATH then
704       ltx.__tag.trace.log("INFO TAG-MATH-SUBTYPE: "..
705         node.type(node.getid(n)).." "..__tag_get_mathsubtype(n),4)
706      end
707      -- endmath
708      ltx.__tag.trace.log("INFO TAG-MC-COMPARE: current "..
709               mccnt.." prev "..mccntprev,4)
710      if mccnt~=mccntprev then -- a new mc chunk
711       ltx.__tag.trace.log ("INFO TAG-NEW-MC-NODE: "..
712                          node.type(node.getid(n))..
713                          " MC"..tostring(mccnt)..
714                          " <=> PREVIOUS "..tostring(mccntprev),4)
715       if mcopen~=0 then -- there is a chunk open, close it (hope there is only one ...
716        box.list=__tag_insert_emc_node (box.list,n)
717        mcopen = mcopen - 1
718        ltx.__tag.trace.log ("INFO TAG-INSERT-EMC: " ..
719          mcpagecnt .. " MCOPEN = " .. mcopen,3)
720        if mcopen ~=0 then
721         ltx.__tag.trace.log ("WARN TAG-OPEN-MC: " .. mcopen,1)
722        end
723       end
724       if ltx.__tag.mc[mccnt] then
725        if ltx.__tag.mc[mccnt]["artifact"] then
726         ltx.__tag.trace.log("INFO TAG-INSERT-ARTIFACT: "..
727                          tostring(ltx.__tag.mc[mccnt]["artifact"]),3)
728         if ltx.__tag.mc[mccnt]["artifact"] == "" then
729          box.list = __tag_insert_bmc_node (box.list,n,"Artifact")
730         else
731          box.list = __tag_insert_bdc_node (box.list,n,"Artifact", "/Type /"..ltx.__tag.mc[mcc
732         end
733        else
734         ltx.__tag.trace.log("INFO TAG-INSERT-TAG: "..
735                          tostring(tag),3)
736         mcpagecnt = mcpagecnt +1
737         ltx.__tag.trace.log ("INFO TAG-INSERT-BDC: "..mcpagecnt,3)
738         local dict= "/MCID "..mcpagecnt
739         if ltx.__tag.mc[mccnt]["raw"] then
740          ltx.__tag.trace.log("INFO TAG-USE-RAW: "..
741            tostring(ltx.__tag.mc[mccnt]["raw"]),3)
742          dict= dict .. " " .. ltx.__tag.mc[mccnt]["raw"]
743         end
744         if ltx.__tag.mc[mccnt]["alt"] then
745          ltx.__tag.trace.log("INFO TAG-USE-ALT: "..
746             tostring(ltx.__tag.mc[mccnt]["alt"]),3)
747          dict= dict .. " " .. ltx.__tag.mc[mccnt]["alt"]
748         end
749         if ltx.__tag.mc[mccnt]["actualtext"] then
750          ltx.__tag.trace.log("INFO TAG-USE-ACTUALTEXT: "..
751            tostring(ltx.__tag.mc[mccnt]["actualtext"]),3)
752          dict= dict .. " " .. ltx.__tag.mc[mccnt]["actualtext"]
753         end
754         box.list = __tag_insert_bdc_node (box.list,n,tag, dict)
```

146

```
755        ltx.__tag.func.store_mc_kid (mccnt,mcpagecnt,abspage)
756        ltx.__tag.func.store_mc_in_page(mccnt,mcpagecnt,abspage)
757        ltx.__tag.trace.show_mc_data (mccnt,3)
758      end
759      mcopen = mcopen + 1
760    else
761      if tagunmarkedbool.mode == truebool.mode then
762        ltx.__tag.trace.log("INFO TAG-NOT-TAGGED: this has not been tagged, using artifact",2)
763        box.list = __tag_insert_bmc_node (box.list,n,"Artifact")
764        mcopen = mcopen + 1
765      else
766        ltx.__tag.trace.log("WARN TAG-NOT-TAGGED: this has not been tagged",1)
767      end
768    end
769    mccntprev = mccnt
770   end
771  end -- end if
772  end -- end for
773  if head then
774    mccnthead, mctypehead,taghead = __tag_get_mc_cnt_type_tag (head)
775    ltx.__tag.trace.log ("INFO TAG-ENDHEAD: " ..
776                          node.type(node.getid(head))..
777                          " MC"..tostring(mccnthead)..
778                          " => TAG "..tostring(mctypehead)..
779                          " => "..tostring(taghead),4)
780  else
781    ltx.__tag.trace.log ("INFO TAG-ENDHEAD: ".. tostring(head),4)
782  end
783  ltx.__tag.trace.log ("INFO TAG-QUITTING-BOX "..
784                        tostring(name)..
785                        " TYPE ".. node.type(node.getid(box)),4)
786  return mcopen,mcpagecnt,mccntprev,mctypeprev
787 end
788
```

(*End of 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.

```
789 function ltx.__tag.func.mark_shipout (box)
790  mcopen = ltx.__tag.func.mark_page_elements (box,-1,-100,0,"Shipout",-1)
791  if mcopen~=0 then -- there is a chunk open, close it (hope there is only one ...
792    local emcnode = __tag_backend_create_emc_node ()
793    local list = box.list
794    if list then
795      list = node.insert_after (list,node.tail(list),emcnode)
796      mcopen = mcopen - 1
797      ltx.__tag.trace.log ("INFO SHIPOUT-INSERT-LAST-EMC: MCOPEN " .. mcopen,3)
798    else
799      ltx.__tag.trace.log ("WARN SHIPOUT-UPS: this shouldn't happen",0)
800    end
801    if mcopen ~=0 then
802      ltx.__tag.trace.log ("WARN SHIPOUT-MC-OPEN: " .. mcopen,1)
```

147

```
803    end
804   end
805 end
```

(*End of 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.

```
806 function ltx.__tag.func.fill_parent_tree_line (page)
807     -- we need to get page-> i=kid -> mcnum -> structnum
808     -- pay attention: the kid numbers and the page number in the parent tree start with 0!
809     local numsentry =""
810     local pdfpage = page-1
811     if ltx.__tag.page[page] and ltx.__tag.page[page][0] then
812     mcchunks=#ltx.__tag.page[page]
813     ltx.__tag.trace.log("INFO PARENTTREE-NUM:  page "..
814                 page.." has "..mcchunks.."+1 Elements ",4)
815     for i=0,mcchunks do
816     -- what does this log??
817      ltx.__tag.trace.log("INFO PARENTTREE-CHUNKS:  "..
818        ltx.__tag.page[page][i],4)
819     end
820     if mcchunks == 0 then
821      -- only one chunk so no need for an array
822      local mcnum  = ltx.__tag.page[page][0]
823      local structnum = ltx.__tag.mc[mcnum]["parent"]
824      local propname  = "g__tag_struct_"..structnum.."_prop"
825      --local objref   = ltx.__tag.tables[propname]["objref"] or "XXXX"
826      local objref = __tag_pdf_object_ref('__tag/struct/'..structnum)
827      ltx.__tag.trace.log("INFO PARENTTREE-STRUCT-OBJREF:  =====>"..
828        tostring(objref),5)
829      numsentry = pdfpage .. " [".. objref .. "]"
830      ltx.__tag.trace.log("INFO PARENTTREE-NUMENTRY: page " ..
831        page.. " num entry = ".. numsentry,3)
832     else
833      numsentry = pdfpage .. " ["
834       for i=0,mcchunks do
835        local mcnum  = ltx.__tag.page[page][i]
836        local structnum = ltx.__tag.mc[mcnum]["parent"] or 0
837        local propname  = "g__tag_struct_"..structnum.."_prop"
838        --local objref   = ltx.__tag.tables[propname]["objref"] or "XXXX"
839        local objref = __tag_pdf_object_ref('__tag/struct/'..structnum)
840        numsentry = numsentry .. " ".. objref
841       end
842      numsentry = numsentry .. "] "
843      ltx.__tag.trace.log("INFO PARENTTREE-NUMENTRY: page " ..
844        page.. " num entry = ".. numsentry,3)
845     end
846     else
847       ltx.__tag.trace.log ("INFO PARENTTREE-NO-DATA: page "..page,3)
848     end
```

148

```
849      return numsentry
850  end
851
852  function ltx.__tag.func.output_parenttree (abspage)
853   for i=1,abspage do
854    line = ltx.__tag.func.fill_parent_tree_line (i) .. "^^J"
855    tex.sprint(catlatex,line)
856   end
857  end
```

(*End of definition for* `ltx.__tag.func.fill_parent_tree_line` *and* `ltx.__tag.func.output_parenttree`.)

858 ⟨/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} {2024-02-02} {0.98u}
4   {part of tagpdf - code related to roles and structure names}
5 ⟨/header⟩
```

1   Code related to roles and structure names

₆ ⟨∗package⟩

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 of 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 of 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 of 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 of 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 of 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_tag_namespace_tmpb_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_tag_namespace_tmpb_tl
15 \tl_new:N \l__tag_role_role_tmpa_tl
16 \tl_new:N \l__tag_role_role_namespace_tmpa_tl
17 \seq_new:N\l__tag_role_tmpa_seq
```

(*End of 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`
`\g__tag_role_rolemap_prop`

This is the object which contains the normal RoleMap. It is probably not needed in pdf 2.0 but currently kept.

```
18 \pdfdict_new:n {g__tag_role/RoleMap_dict}
19 \prop_new:N \g__tag_role_rolemap_prop
```

(*End of 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`

```
20 \pdf_version_compare:NnTF < {2.0}
21   {
22     \cs_new_protected:Npn \__tag_role_NS_new:nnn #1 #2 #3
23       {
24         \prop_new:c { g__tag_role_NS_#1_prop }
25         \prop_new:c { g__tag_role_NS_#1_class_prop }
26         \prop_gput:Nne \g__tag_role_NS_prop {#1}{}
27       }
28   }
29   {
30     \cs_new_protected:Npn \__tag_role_NS_new:nnn #1 #2 #3
31       {
32         \prop_new:c { g__tag_role_NS_#1_prop }
33         \prop_new:c { g__tag_role_NS_#1_class_prop }
34         \pdf_object_new:n {tag/NS/#1}
35         \pdfdict_new:n     {g__tag_role/Namespace_#1_dict}
36         \pdf_object_new:n {__tag/RoleMapNS/#1}
37         \pdfdict_new:n     {g__tag_role/RoleMapNS_#1_dict}
38         \pdfdict_gput:nnn
39           {g__tag_role/Namespace_#1_dict}
40           {Type}
41           {/Namespace}
42         \pdf_string_from_unicode:nnN{utf8/string}{#2}\l__tag_tmpa_str
43         \tl_if_empty:NF \l__tag_tmpa_str
44           {
45             \pdfdict_gput:nne
46               {g__tag_role/Namespace_#1_dict}
47               {NS}
48               {\l__tag_tmpa_str}
49           }
50         %RoleMapNS is added in tree
51         \tl_if_empty:nF  {#3}
```

```
52        {
53          \pdfdict_gput:nne{g__tag_role/Namespace_#1_dict}
54          {Schema}{#3}
55        }
56        \prop_gput:Nne \g__tag_role_NS_prop {#1}{\pdf_object_ref:n{tag/NS/#1}~}
57      }
58  }
```

(*End of 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`

```
59  \str_const:Ne \c__tag_role_userNS_id_str
60    { data:,
61      \int_to_Hex:n{\int_rand:n {65535}}
62      \int_to_Hex:n{\int_rand:n {65535}}
63      -
64      \int_to_Hex:n{\int_rand:n {65535}}
65      -
66      \int_to_Hex:n{\int_rand:n {65535}}
67      -
68      \int_to_Hex:n{\int_rand:n {65535}}
69      -
70      \int_to_Hex:n{\int_rand:n {16777215}}
71      \int_to_Hex:n{\int_rand:n {16777215}}
72    }
```

(*End of definition for* `\c__tag_role_userNS_id_str`.)

Now we setup the standard names spaces. The mathml space is loaded also for pdf < 2.0 but not added to RoleMap unless a boolean is set to true with `tagpdf-setup{mathml-tags}`.

```
73  \bool_new:N \g__tag_role_add_mathml_bool
74  \__tag_role_NS_new:nnn {pdf}    {http://iso.org/pdf/ssn}{}
75  \__tag_role_NS_new:nnn {pdf2}   {http://iso.org/pdf2/ssn}{}
76  \__tag_role_NS_new:nnn {mathml}{http://www.w3.org/1998/Math/MathML}{}
77  \__tag_role_NS_new:nnn {latex} {https://www.latex-project.org/ns/dflt/2022}{}
78  \__tag_role_NS_new:nnn {latex-book} {https://www.latex-project.org/ns/book/2022}{}
79  \__tag_role_NS_new:nnn {latex-inline} {https://www.latex-project.org/ns/inline/2022}{}
80  \exp_args:Nne
81    \__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.

```
82  \pdf_version_compare:NnTF < {2.0}
83    {
84      \sys_if_engine_luatex:TF
```

154

```
85      {
86        \cs_new_protected:Npn \__tag_role_alloctag:nnn #1 #2 #3 %#1 tagname, ns, type
87          {
88            \lua_now:e { ltx.__tag.func.alloctag ('#1') }
89            \prop_gput:Nnn \g__tag_role_tags_NS_prop    {#1}{#2}
90            \prop_gput:cnn {g__tag_role_NS_#2_prop}  {#1}{{}{}}
91            \prop_gput:Nnn \g__tag_role_tags_class_prop {#1}{#3}
92            \prop_gput:cnn {g__tag_role_NS_#2_class_prop}  {#1}{--UNUSED--}
93          }
94      }
95      {
96        \cs_new_protected:Npn \__tag_role_alloctag:nnn #1 #2 #3
97          {
98            \prop_gput:Nnn \g__tag_role_tags_NS_prop    {#1}{#2}
99            \prop_gput:cnn {g__tag_role_NS_#2_prop}  {#1}{{}{}}
100           \prop_gput:Nnn \g__tag_role_tags_class_prop {#1}{#3}
101           \prop_gput:cnn {g__tag_role_NS_#2_class_prop}  {#1}{--UNUSED--}
102         }
103     }
104   }
105   {
106    \sys_if_engine_luatex:TF
107      {
108        \cs_new_protected:Npn \__tag_role_alloctag:nnn #1 #2 #3 %#1 tagname, ns, type
109          {
110            \lua_now:e { ltx.__tag.func.alloctag ('#1') }
111            \prop_gput:Nnn \g__tag_role_tags_NS_prop    {#1}{#2}
112            \prop_gput:cnn {g__tag_role_NS_#2_prop}  {#1}{{}{}}
113            \prop_gput:Nnn \g__tag_role_tags_class_prop {#1}{--UNUSED--}
114            \prop_gput:cnn {g__tag_role_NS_#2_class_prop}  {#1}{#3}
115          }
116      }
117      {
118        \cs_new_protected:Npn \__tag_role_alloctag:nnn #1 #2 #3
119          {
120            \prop_gput:Nnn \g__tag_role_tags_NS_prop    {#1}{#2}
121            \prop_gput:cnn {g__tag_role_NS_#2_prop}  {#1}{{}{}}
122            \prop_gput:Nnn \g__tag_role_tags_class_prop {#1}{--UNUSED--}
123            \prop_gput:cnn {g__tag_role_NS_#2_class_prop}  {#1}{#3}
124          }
125      }
126   }
127 \cs_generate_variant:Nn  \__tag_role_alloctag:nnn {nnV}
```

(*End of 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.

```
128 \cs_new_protected:Nn \__tag_role_add_tag:nn % (new) name, reference to old
129   {
```

155

checks and messages

```
130        \__tag_check_add_tag_role:nn {#1}{#2}
131        \prop_if_in:NnF \g__tag_role_tags_NS_prop {#1}
132          {
133            \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
134              {
135                \msg_info:nnn { tag }{new-tag}{#1}
136              }
137          }
```

now the addition

```
138        \prop_get:NnN \g__tag_role_tags_class_prop {#2}\l__tag_tmpa_tl
139        \quark_if_no_value:NT \l__tag_tmpa_tl
140          {
141            \tl_set:Nn\l__tag_tmpa_tl{--UNKNOWN--}
142          }
143        \__tag_role_alloctag:nnV {#1}{user}\l__tag_tmpa_tl
```

We resolve rolemapping recursively so that all targets are stored as standard tags.

```
144        \tl_if_empty:nF { #2 }
145          {
146            \prop_get:NnN \g__tag_role_rolemap_prop {#2}\l__tag_tmpa_tl
147            \quark_if_no_value:NTF \l__tag_tmpa_tl
148              {
149                \prop_gput:Nne \g__tag_role_rolemap_prop {#1}{\tl_to_str:n{#2}}
150              }
151              {
152                \prop_gput:NnV \g__tag_role_rolemap_prop {#1}\l__tag_tmpa_tl
153              }
154          }
155    }
156  \cs_generate_variant:Nn \__tag_role_add_tag:nn {VV,ne}
```

(*End of 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:nnNN

```
157  \pdf_version_compare:NnT < {2.0}
158    {
159      \cs_new:Npn \__tag_role_get:nnNN #1#2#3#4 %#1 tag, #2 NS, #3 tlvar which hold the role tag
160        {
161          \prop_get:NnNF \g__tag_role_rolemap_prop {#1}#3
162            {
163              \tl_set:Nn #3 {#1}
164            }
165          \tl_set:Nn #4 {}
166        }
167      \cs_generate_variant:Nn \__tag_role_get:nnNN {VVNN}
168    }
169
```

(*End of definition for* __tag_role_get:nnNN.)

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:nne {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:cne { 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       }
```

We also store into the pdf 1.7 rolemapping so that we can add that as fallback for pdf 1.7 processor

```
206     \tl_if_empty:nF { #3 }
207       {
208         \tl_if_eq:nnF{#1}{#3}
209           {
210             \prop_get:NnN \g__tag_role_rolemap_prop {#3}\l__tag_tmpa_tl
211             \quark_if_no_value:NTF \l__tag_tmpa_tl
212               {
213                 \prop_gput:Nne \g__tag_role_rolemap_prop {#1}{\tl_to_str:n{#3}}
```

157

```
214                    }
215                    {
216                       \prop_gput:NnV \g__tag_role_rolemap_prop {#1}\l__tag_tmpa_tl
217                    }
218                  }
219              }
220          }
221    \cs_generate_variant:Nn \__tag_role_add_tag:nnnn {VVVV}
```

(*End of 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)

```
222    \pdf_version_compare:NnF < {2.0}
223    {
224      \cs_new:Npn \__tag_role_get:nnNN #1#2#3#4
225        %#1 tag, #2 NS,
226        %#3 tlvar which hold the role tag
227        %#4 tlvar which hold the name of the target NS
228        {
229          \prop_get:cnNTF {g__tag_role_NS_#2_prop} {#1}\l__tag_tmpa_tl
230            {
231              \tl_set:Ne #3 {\exp_last_unbraced:NV\use_i:nn   \l__tag_tmpa_tl}
232              \tl_set:Ne #4 {\exp_last_unbraced:NV\use_ii:nn  \l__tag_tmpa_tl}
233            }
234            {
235              \tl_set:Nn #3 {#1}
236              \tl_set:Nn #4 {#2}
237            }
238        }
239      \cs_generate_variant:Nn \__tag_role_get:nnNN {VVNN}
240    }
```

(*End of definition for* `\__tag_role_get:nnNN`.)

1.4 Helper command to read the data from files

In this section we setup the helper command to read namespace files.

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 special name spaces shouldn't update the default role or add to the rolemap again, they only store the values for later uses. We use a boolean here.

```
241    \bool_new:N\l__tag_role_update_bool
242    \bool_set_true:N \l__tag_role_update_bool
243    \pdf_version_compare:NnTF < {2.0}
244    {
245      \cs_new_protected:Npn \__tag_role_read_namespace_line:nw #1#2,#3,#4,#5,#6\q_stop %
246      % #1 NS, #2 tag, #3 rolemap, #4 NS rolemap #5 type
247        {
248          \tl_if_empty:nF { #2 }
249            {
250              \bool_if:NTF \l__tag_role_update_bool
```

158

```
251           {
252            \tl_if_empty:nTF {#5}
253              {
254                \prop_get:NnN \g__tag_role_tags_class_prop  {#3}\l__tag_tmpa_tl
255                \quark_if_no_value:NT \l__tag_tmpa_tl
256                  {
257                    \tl_set:Nn\l__tag_tmpa_tl{--UNKNOWN--}
258                  }
259              }
260              {
261                \tl_set:Nn \l__tag_tmpa_tl {#5}
262              }
263            \__tag_role_alloctag:nnV {#2}{#1}\l__tag_tmpa_tl
264            \tl_if_eq:nnF {#2}{#3}
265              {
266                \__tag_role_add_tag:nn {#2}{#3}
267              }
268            \prop_gput:cnn {g__tag_role_NS_#1_prop}  {#2}{{#3}{}}
269          }
270          {
271            \prop_gput:cnn {g__tag_role_NS_#1_prop}  {#2}{{#3}{}}
272            \prop_gput:cnn {g__tag_role_NS_#1_class_prop}  {#2}{--UNUSED--}
273          }
274        }
275      }
276  }
277  {
278    \cs_new_protected:Npn \__tag_role_read_namespace_line:nw #1#2,#3,#4,#5,#6\q_stop %
279    % #1 NS, #2 tag, #3 rolemap, #4 NS rolemap #5 type
280      {
281        \tl_if_empty:nF {#2}
282          {
283          \tl_if_empty:nTF {#5}
284            {
285              \prop_get:cnN { g__tag_role_NS_#4_class_prop } {#3}\l__tag_tmpa_tl
286              \quark_if_no_value:NT \l__tag_tmpa_tl
287                {
288                  \tl_set:Nn\l__tag_tmpa_tl{--UNKNOWN--}
289                }
290            }
291            {
292              \tl_set:Nn \l__tag_tmpa_tl {#5}
293            }
294          \__tag_role_alloctag:nnV {#2}{#1}\l__tag_tmpa_tl
295          \bool_lazy_and:nnT
296            { ! \tl_if_empty_p:n {#3} }{! \str_if_eq_p:nn {#1}{pdf2}}
297            {
298              \__tag_role_add_tag:nnnn {#2}{#1}{#3}{#4}
299            }
300          \prop_gput:cnn {g__tag_role_NS_#1_prop}  {#2}{{#3}{#4}}
301        }
302    }
303  }
```

(End of definition for `\__tag_role_read_namespace_line:nw`*.)*

__tag_role_read_namespace:nn   This command reads a namespace file in the format tagpdf-ns-XX.def

```
304 \cs_new_protected:Npn \__tag_role_read_namespace:nn #1 #2 %name of namespace #2 name of file
305   {
306     \prop_if_exist:cF {g__tag_role_NS_#1_prop}
307       { \msg_warning:nnn {tag}{namespace-unknown}{#1} }
308     \file_if_exist:nTF { tagpdf-ns-#2.def }
309     {
310       \ior_open:Nn \g_tmpa_ior {tagpdf-ns-#2.def}
311       \msg_info:nnn {tag}{read-namespace}{#2}
312       \ior_map_inline:Nn \g_tmpa_ior
313         {
314            \__tag_role_read_namespace_line:nw {#1} ##1,,,,\q_stop
315         }
316       \ior_close:N\g_tmpa_ior
317     }
318     {
319      \msg_info:nnn{tag}{namespace-missing}{#2}
320     }
321   }
322
```

(*End of definition for* __tag_role_read_namespace:nn.)

__tag_role_read_namespace:n   This command reads the default namespace file.

```
323 \cs_new_protected:Npn \__tag_role_read_namespace:n #1 %name of namespace
324   {
325     \__tag_role_read_namespace:nn {#1}{#1}
326   }
```

(*End of 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: if two namespace define the same tag, the last one defines which one is used if the namespace is not explicitly given.

```
327 \__tag_role_read_namespace:n {pdf}
328 \__tag_role_read_namespace:n {pdf2}
329 \__tag_role_read_namespace:n {mathml}
```

in pdf 1.7 the following namespaces should only store the settings for later use:

```
330 \bool_set_false:N\l__tag_role_update_bool
331 \__tag_role_read_namespace:n {latex-inline}
332 \__tag_role_read_namespace:n {latex-book}
333 \bool_set_true:N\l__tag_role_update_bool
334 \__tag_role_read_namespace:n {latex}
335 \__tag_role_read_namespace:nn {latex} {latex-lab}
336 \__tag_role_read_namespace:n {pdf}
337 \__tag_role_read_namespace:n {pdf2}
```

But is the class provides a \chapter command then we switch

```
338 \pdf_version_compare:NnTF < {2.0}
339   {
340     \hook_gput_code:nnn {begindocument}{tagpdf}
341       {
342         \cs_if_exist:NT \chapter
```

```
343              {
344                \prop_map_inline:cn{g__tag_role_NS_latex-book_prop}
345                  {
346                    \__tag_role_add_tag:ne {#1}{\use_i:nn #2\c_empty_tl\c_empty_tl}
347                  }
348              }
349          }
350    }
351    {
352      \hook_gput_code:nnn {begindocument}{tagpdf}
353        {
354          \cs_if_exist:NT \chapter
355            {
356              \prop_map_inline:cn{g__tag_role_NS_latex-book_prop}
357                {
358                  \prop_gput:Nnn \g__tag_role_tags_NS_prop     { #1 }{ latex-book }
359                }
360            }
361        }
362    }
```

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.

```
363 \intarray_new:Nn \g__tag_role_parent_child_intarray {6000}
```

(*End of definition for* \g__tag_role_parent_child_intarray.)

\c__tag_role_rules_prop   These two properties map the rule strings to numbers and back. There are in tagpdf-
\c__tag_role_rules_num_prop   data.dtx near the csv files for easier maintenance.

(*End of 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!

```
364 \cs_new_protected:Npn \__tag_store_parent_child_rule:nnn #1 #2 #3 % num parent, num child, #3
365   {
366     \intarray_gset:Nnn \g__tag_role_parent_child_intarray
367       { #1#2 }{0\prop_item:Nn\c__tag_role_rules_prop{#3}}
368   }
```

(*End of 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

```
369 \int_zero:N  \l__tag_tmpa_int
```

Open the file depending on the PDF version

```
370  \pdf_version_compare:NnTF < {2.0}
371    {
372      \ior_open:Nn \g_tmpa_ior {tagpdf-parent-child.csv}
373    }
374    {
375      \ior_open:Nn \g_tmpa_ior {tagpdf-parent-child-2.csv}
376    }
```

Now the main loop over the file

```
377  \ior_map_inline:Nn \g_tmpa_ior
378    {
```

ignore lines containing only comments

```
379      \tl_if_empty:nF{#1}
380        {
```

count the lines ...

```
381          \int_incr:N\l__tag_tmpa_int
```

put the line into a seq. Attention! empty cells are dropped.

```
382          \seq_set_from_clist:Nn\l__tag_tmpa_seq { #1 }
383          \int_compare:nNnTF {\l__tag_tmpa_int}=1
```

This handles the header line. It gives the tags 2-digit numbers

```
384            {
385              \seq_map_indexed_inline:Nn \l__tag_tmpa_seq
386                {
387                  \prop_gput:Nne\g__tag_role_index_prop
388                    {##2}
389                    {\int_compare:nNnT{##1}<{10}{0}##1}
390                }
391            }
```

now the data lines.

```
392            {
393              \seq_set_from_clist:Nn\l__tag_tmpa_seq { #1 }
```

get the name of the child tag from the first column

```
394              \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`

```
395              \prop_get:NVN \g__tag_role_index_prop \l__tag_tmpa_tl \l__tag_tmpb_tl
```

remove column 2+3

```
396              \seq_pop_left:NN\l__tag_tmpa_seq\l__tag_tmpa_tl
397              \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.

```
398              \seq_map_indexed_inline:Nn \l__tag_tmpa_seq
399                {
400                  \exp_args:Nne
401                  \__tag_store_parent_child_rule:nnn {##1}{\l__tag_tmpb_tl}{ ##2 }
402                }
403            }
404        }
405    }
```

close the read handle.

```
406  \ior_close:N\g_tmpa_ior
```

The Root, Hn and mathml tags are special and need to be added explicitly

```
407  \prop_get:NnN\g__tag_role_index_prop{StructTreeRoot}\l__tag_tmpa_tl
408  \prop_gput:Nne\g__tag_role_index_prop{Root}{\l__tag_tmpa_tl}
409  \prop_get:NnN\g__tag_role_index_prop{Hn}\l__tag_tmpa_tl
410  \pdf_version_compare:NnTF < {2.0}
411    {
412      \int_step_inline:nn{6}
413        {
414          \prop_gput:Nne\g__tag_role_index_prop{H#1}{\l__tag_tmpa_tl}
415        }
416    }
417    {
418      \int_step_inline:nn{10}
419        {
420          \prop_gput:Nne\g__tag_role_index_prop{H#1}{\l__tag_tmpa_tl}
421        }
```

all mathml tags are currently handled identically

```
422      \prop_get:NnN\g__tag_role_index_prop {mathml}\l__tag_tmpa_tl
423      \prop_get:NnN\g__tag_role_index_prop {math}\l__tag_tmpb_tl
424      \prop_map_inline:Nn \g__tag_role_NS_mathml_prop
425        {
426          \prop_gput:NnV\g__tag_role_index_prop{#1}\l__tag_tmpa_tl
427        }
428      \prop_gput:NnV\g__tag_role_index_prop{math}\l__tag_tmpb_tl
429    }
```

1.6.2 Retrieving the parent-child rule

__tag_role_get_parent_child_rule:nnnN This command retrieves the rule (as a number) and stores it in the tl-var. It assumes that the tag in #1 is a standard tag after role mapping for which a rule exist and is *not* one of Part, Div, NonStruct as the real parent has already been identified. #3 can be used to pass along data about the original tags and is only used in messages.

TODO check temporary variables. Check if the tl-var should be fix.

```
430  \tl_new:N \l__tag_parent_child_check_tl
431  \cs_new_protected:Npn \__tag_role_get_parent_child_rule:nnnN #1 #2 #3 #4
432    % #1 parent (string) #2 child (string) #3 text for messages (eg. about Div or Rolemapping)
433    % #4 tl for state
434    {
435      \prop_get:NnN \g__tag_role_index_prop{#1}\l__tag_tmpa_tl
436      \prop_get:NnN \g__tag_role_index_prop{#2}\l__tag_tmpb_tl
437      \bool_lazy_and:nnTF
438        { ! \quark_if_no_value_p:N \l__tag_tmpa_tl }
439        { ! \quark_if_no_value_p:N \l__tag_tmpb_tl }
440        {
```

Get the rule from the intarray

```
441          \tl_set:Ne#4
442            {
443              \intarray_item:Nn
444                \g__tag_role_parent_child_intarray
```

163

```
445                 {\l__tag_tmpa_tl\l__tag_tmpb_tl}
446             }
```

If the state is something is wrong …

```
447         \int_compare:nNnT
448             {#4} = {\prop_item:Nn\c__tag_role_rules_prop{}}
449             {
450                 %warn ?
```

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.

```
451                 }
```

This is the message, this can perhaps go into debug mode.

```
452         \group_begin:
453         \int_compare:nNnT {\l__tag_tmpa_int*\l__tag_loglevel_int} > { 0 }
454             {
455                 \prop_get:NVNF\c__tag_role_rules_num_prop #4 \l__tag_tmpa_tl
456                     {
457                         \tl_set:Nn \l__tag_tmpa_tl {unknown}
458                     }
459                 \tl_set:Nn \l__tag_tmpb_tl {#1}
460                 \msg_note:nneee
461                     { tag }
462                     { role-parent-child }
463                     { #1 }
464                     { #2 }
465                     {
466                         #4~(='\l__tag_tmpa_tl')
467                             \iow_newline:
468                             #3
469                     }
470             }
471         \group_end:
472     }
473     {
474         \tl_set:Nn#4 {0}
475         \msg_warning:nneee
476             { tag }
477             {role-parent-child}
478             { #1 }
479             { #2 }
480             { unknown! }
481     }
482 }
483 \cs_generate_variant:Nn\__tag_role_get_parent_child_rule:nnnN {VVVN,VVnN}
```

(*End of definition for* __tag_role_get_parent_child_rule:nnnN.)

__tag_check_parent_child:nnnnN This commands translates rolemaps its arguments and then calls __tag_role_get_-parent_child_rule:nnnN. It does not try to resolve inheritance of Div etc but instead warns that the rule can not be detected in this case. 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.

```
484 \pdf_version_compare:NnTF < {2.0}
```

164

```
485    {
486    \cs_new_protected:Npn \__tag_check_parent_child:nnnnN #1 #2 #3 #4 #5
487    %#1 parent tag,#2 NS, #3 child tag, #4 NS, #5 tl var
488      {
```

for debugging messages we store the arguments.

```
489        \prop_put:Nnn \l__tag_role_debug_prop {parent} {#1}
490        \prop_put:Nnn \l__tag_role_debug_prop {child}  {#3}
```

get the standard tags through rolemapping if needed at first the parent

```
491        \prop_get:NnNTF \g__tag_role_index_prop {#1}\l__tag_tmpa_tl
492          {
493            \tl_set:Nn \l__tag_tmpa_tl {#1}
494          }
495          {
496            \prop_get:NnNF \g__tag_role_rolemap_prop {#1}\l__tag_tmpa_tl
497              {
498                \tl_set:Nn \l__tag_tmpa_tl {\q_no_value}
499              }
500          }
```

now the child

```
501        \prop_get:NnNTF \g__tag_role_index_prop {#3}\l__tag_tmpb_tl
502          {
503            \tl_set:Nn \l__tag_tmpb_tl {#3}
504          }
505          {
506            \prop_get:NnNF \g__tag_role_rolemap_prop {#3}\l__tag_tmpb_tl
507              {
508                \tl_set:Nn \l__tag_tmpb_tl {\q_no_value}
509              }
510          }
```

if we got tags for parent and child we call the checking command

```
511        \bool_lazy_and:nnTF
512          { ! \quark_if_no_value_p:N \l__tag_tmpa_tl }
513          { ! \quark_if_no_value_p:N \l__tag_tmpb_tl }
514          {
515            \__tag_role_get_parent_child_rule:VVnN
516              \l__tag_tmpa_tl \l__tag_tmpb_tl
517              {Rolemapped~from:~'#1'~-->~'#3'}
518              #5
519          }
520          {
521            \tl_set:Nn #5 {0}
522            \msg_warning:nneee
523              { tag }
524              {role-parent-child}
525              { #1 }
526              { #3 }
527              { unknown! }
528          }
529      }
530    \cs_new_protected:Npn \__tag_check_parent_child:nnN #1#2#3
531      {
532        \__tag_check_parent_child:nnnnN {#1}{}{#2}{}#3
```

165

```
533        }
534     }
```
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.
```
535     {
536      \cs_new_protected:Npn \__tag_check_parent_child:nnN #1 #2 #3
537        {
538           \prop_get:NnN\g__tag_role_tags_NS_prop {#1}\l__tag_role_tag_namespace_tmpa_tl
539           \prop_get:NnN\g__tag_role_tags_NS_prop {#2}\l__tag_role_tag_namespace_tmpb_tl
540           \str_if_eq:nnT{#2}{MC}{\tl_clear:N \l__tag_role_tag_namespace_tmpb_tl}
541           \bool_lazy_and:nnTF
542             { ! \quark_if_no_value_p:N \l__tag_role_tag_namespace_tmpa_tl }
543             { ! \quark_if_no_value_p:N \l__tag_role_tag_namespace_tmpb_tl }
544             {
545               \__tag_check_parent_child:nVnVN
546                 {#1}\l__tag_role_tag_namespace_tmpa_tl
547                 {#2}\l__tag_role_tag_namespace_tmpb_tl
548                 #3
549             }
550             {
551               \tl_set:Nn #3 {0}
552               \msg_warning:nneee
553               { tag }
554               {role-parent-child}
555               { #1 }
556               { #2 }
557               { unknown! }
558             }
559        }
```
and now the real command.
```
560      \cs_new_protected:Npn \__tag_check_parent_child:nnnnN #1 #2 #3 #4 #5 %tag,NS,tag,NS, tl var
561        {
562           \prop_put:Nnn \l__tag_role_debug_prop {parent} {#1/#2}
563           \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 rolemapping from the namespace
```
564           \tl_if_empty:nTF  {#2}
565             {
566               \tl_set:Nn \l__tag_tmpa_tl {#1}
567             }
568             {
569               \prop_get:cnNTF
570                 { g__tag_role_NS_#2_prop }
571                 {#1}
572                 \l__tag_tmpa_tl
573                 {
574                   \tl_set:Ne \l__tag_tmpa_tl {\tl_head:N\l__tag_tmpa_tl}
575                   \tl_if_empty:NT\l__tag_tmpa_tl
576                     {
577                       \tl_set:Nn \l__tag_tmpa_tl {#1}
578                     }
```

166

```
579                    }
580                    {
581                       \tl_set:Nn \l__tag_tmpa_tl {\q_no_value}
582                    }
583                 }
```

and the same for the child If the namespace is empty, we assume a standard tag, otherwise we retrieve the rolemapping from the namespace

```
584             \tl_if_empty:nTF  {#4}
585               {
586                 \tl_set:Nn \l__tag_tmpb_tl {#3}
587               }
588               {
589                 \prop_get:cnNTF
590                   { g__tag_role_NS_#4_prop }
591                   {#3}
592                   \l__tag_tmpb_tl
593                   {
594                     \tl_set:Ne \l__tag_tmpb_tl { \tl_head:N\l__tag_tmpb_tl }
595                     \tl_if_empty:NT\l__tag_tmpb_tl
596                       {
597                         \tl_set:Nn \l__tag_tmpb_tl {#3}
598                       }
599                   }
600                   {
601                     \tl_set:Nn \l__tag_tmpb_tl {\q_no_value}
602                   }
603               }
```

and now get the relation

```
604             \bool_lazy_and:nnTF
605               { ! \quark_if_no_value_p:N \l__tag_tmpa_tl }
606               { ! \quark_if_no_value_p:N \l__tag_tmpb_tl }
607               {
608                 \__tag_role_get_parent_child_rule:VVnN
609                   \l__tag_tmpa_tl \l__tag_tmpb_tl
610                   {Rolemapped~from~'#1/#2'~--->~'#3\str_if_empty:nF{#4}{/#4}'}
611                   #5
612               }
613               {
614                 \tl_set:Nn #5 {0}
615                 \msg_warning:nneee
616                   { tag }
617                   {role-parent-child}
618                   { #1 }
619                   { #3 }
620                   { unknown! }
621               }
622         }
623     }
624 \cs_generate_variant:Nn\__tag_check_parent_child:nnN {VVN}
625 \cs_generate_variant:Nn\__tag_check_parent_child:nnnnN {VVVVN,nVnVN,VVnnN}
626 ⟨/package⟩
```

(*End of definition for* `__tag_check_parent_child:nnnnN`.)

\tag_check_child:nn_TF_

```
627 ⟨base⟩\prg_new_protected_conditional:Npnn \tag_check_child:nn #1 #2 {T,F,TF}{\prg_return_true:}
628 ⟨*package⟩
629 \prg_set_protected_conditional:Npnn \tag_check_child:nn #1 #2 {T,F,TF}
630   {
631     \seq_get:NN\g__tag_struct_stack_seq\l__tag_tmpa_tl
632     \__tag_struct_get_parentrole:eNN
633       {\l__tag_tmpa_tl}
634       \l__tag_get_parent_tmpa_tl
635       \l__tag_get_parent_tmpb_tl
636     \__tag_check_parent_child:VVnnN
637       \l__tag_get_parent_tmpa_tl
638       \l__tag_get_parent_tmpb_tl
639       {#1}{#2}
640       \l__tag_parent_child_check_tl
641     \int_compare:nNnTF { \l__tag_parent_child_check_tl } < {0}
642       {\prg_return_false:}
643       {\prg_return_true:}
644   }
```

(*End of definition for* \tag_check_child:nnTF. *This function is documented on page 150.*)

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

```
645 \tl_new:N \l__tag_role_remap_tag_tl
646 \tl_new:N \l__tag_role_remap_NS_tl
```

(*End of 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?

```
647 \cs_new_protected:Npn \__tag_role_remap: { }
```

(*End of definition for* __tag_role_remap:.)

__tag_role_remap_id: This is copy in case we have to restore the main command.

```
648 \cs_set_eq:NN \__tag_role_remap_id: \__tag_role_remap:
```

(*End of definition for* __tag_role_remap_id:.)

__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.

```
649 \pdf_version_compare:NnTF < {2.0}
650   {
651     \cs_new_protected:Npn \__tag_role_remap_inline:
652       {
```

168

```
653        \prop_get:cVNT { g__tag_role_NS_latex-inline_prop }\l__tag_role_remap_tag_tl\l__tag_tn
654          {
655            \tl_set:Ne\l__tag_role_remap_tag_tl
656              {
657                \exp_last_unbraced:NV\use_i:nn \l__tag_tmpa_tl
658              }
659            \tl_set:Ne\l__tag_role_remap_NS_tl
660              {
661                \exp_last_unbraced:NV\use_ii:nn \l__tag_tmpa_tl
662              }
663          }
664        \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
665          {
666            \msg_note:nne { tag } { role-remapping }{ \l__tag_role_remap_tag_tl }
667          }
668      }
669  }
670  {
671    \cs_new_protected:Npn \__tag_role_remap_inline:
672      {
673        \prop_get:cVNT { g__tag_role_NS_latex-inline_prop }\l__tag_role_remap_tag_tl\l__tag_tn
674          {
675            \tl_set:Nn\l__tag_role_remap_NS_tl {latex-inline}
676          }
677        \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
678          {
679            \msg_note:nne { tag } { role-remapping }{ \l__tag_role_remap_tag_tl/latex-
    inline }
680          }
681      }
682  }
```

(*End of 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)

```
683  \keys_define:nn { __tag / tag-role }
684    {
685      ,tag .tl_set:N = \l__tag_role_tag_tmpa_tl
686      ,tag-namespace   .tl_set:N = \l__tag_role_tag_namespace_tmpa_tl
687      ,role .tl_set:N = \l__tag_role_role_tmpa_tl
688      ,role-namespace .tl_set:N = \l__tag_role_role_namespace_tmpa_tl
689    }
690
691  \keys_define:nn { __tag / setup }
692    {
693       mathml-tags .bool_gset:N = \g__tag_role_add_mathml_bool
694      ,add-new-tag .code:n =
695        {
696          \keys_set_known:nnnN
```

169

```
697         {__tag/tag-role}
698         {
699           tag-namespace=user,
700           role-namespace=, %so that we can test for it.
701          #1
702         }{__tag/tag-role}\l_tmpa_tl
703       \tl_if_empty:NF \l_tmpa_tl
704         {
705           \exp_args:NNno \seq_set_split:Nnn \l_tmpa_seq { / } {\l_tmpa_tl/}
706           \tl_set:Ne \l__tag_role_tag_tmpa_tl  { \seq_item:Nn \l_tmpa_seq {1} }
707           \tl_set:Ne \l__tag_role_role_tmpa_tl { \seq_item:Nn \l_tmpa_seq {2} }
708         }
709       \tl_if_empty:NT \l__tag_role_role_namespace_tmpa_tl
710         {
711           \prop_get:NVNTF
712             \g__tag_role_tags_NS_prop
713             \l__tag_role_role_tmpa_tl
714             \l__tag_role_role_namespace_tmpa_tl
715             {
716               \prop_if_in:NVF\g__tag_role_NS_prop \l__tag_role_role_namespace_tmpa_tl
717                {
718                  \tl_set:Nn \l__tag_role_role_namespace_tmpa_tl {user}
719                }
720             }
721             {
722               \tl_set:Nn \l__tag_role_role_namespace_tmpa_tl {user}
723             }
724         }
725       \pdf_version_compare:NnTF < {2.0}
726        {
727         %TODO add check for emptyness?
728           \__tag_role_add_tag:VV
729              \l__tag_role_tag_tmpa_tl
730              \l__tag_role_role_tmpa_tl
731        }
732        {
733          \__tag_role_add_tag:VVVV
734             \l__tag_role_tag_tmpa_tl
735             \l__tag_role_tag_namespace_tmpa_tl
736             \l__tag_role_role_tmpa_tl
737             \l__tag_role_role_namespace_tmpa_tl
738        }
739      }
740    }
741 ⟨/package⟩
```

(*End of definition for* tag (rolemap-key) *and others. These functions are documented on page 150.*)

Part X

The **tagpdf-space** module
Code related to real space chars
Part of the tagpdf package

<div style="border-top">interwordspace␣(setup-key)</div>

This key allows to activate/deactivate the real space chars if the engine supports it. The allowed values are `true`, `on`, `false`, `off`.

<div style="border-top">show-spaces␣(setup-key)</div>

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} {2024-02-02} {0.98u}
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.

<div style="color:red">interwordspace␣(setup-key)
show-spaces␣(setup-key)</div>

```
6  ⟨*package⟩
7  \keys_define:nn { __tag / setup }
8    {
9      interwordspace .choices:nn = { true, on }
10       { \msg_warning:nne {tag}{sys-no-interwordspace}{\c_sys_engine_str}  },
11     interwordspace .choices:nn = { false, off }
12       { \msg_warning:nne {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 of definition for* interwordspace (setup-key) *and* show-spaces (setup-key). *These functions are documented on page*)

__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 of 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.

186