tagpdf – LaTeX kernel code for PDF tagging[*]

Ulrike Fischer[†]

Released 2026-03-19

Contents

---

[*]This file describes v0.99z, last revised 2026-03-19.

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

The `tagpdf` main module
Part of the tagpdf package
Ulrike Fischer
Version 0.99z, released 2026-03-19

Part I

\tag_suspend:n	\tag_suspend:n {⟨*label*⟩}
\tag_resume:n	\tag_resume:n {⟨*label*⟩}
\tag_stop:n	\tag_stop:n {⟨*label*⟩} (*deprecated*)
\tag_start:n	\tag_start:n {⟨*label*⟩} (*deprecated*)

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_resume:n will not restart tagging. ⟨*label*⟩ is only used in debugging messages to allow to follow the nesting and to identify which code is disabling the tagging. The label is not expanded so can be a single token, e.g. \caption. \tag_suspend:n and \tag_resume:n are the l3-layer variants of \SuspendTagging and \ResumeTagging and will be provided by the kernel in the next release.

\tag_stop:	*deprecated* These are variants of the above commands without the debuging level. They
\tag_start:	are now deprecated and it is recommended to use the kernel command \SuspendTagging,
\tagstop	\ResumeTagging, \tag_suspend:n and \tag_resume:n instead.
\tagstart	

activate/spaces (*setup key*) activate/spaces activates the additional parsing needed for interword spaces. It replaces the deprecated key interwordspace.

activate/mc (*setup key*) A key to activate the marked content code. It should be used only in special cases, e.g.
~~vate-mc (deprecated)~~ (*setup key*) for debugging.

activate/tree (*setup key*) This key activates the code that finalize the various trees. It should be used only in
~~te-tree (deprecated)~~ (*setup key*) special cases, e.g. for debugging.

activate/struct (*setup key*) This key activates the code for structures. It should be used only in special cases, e.g.
~~-struct (deprecated)~~ (*setup key*) for debugging.

activate/all (*setup key*) This is a meta key for the three previous keys and is normally what should be used to
~~ate-all (deprecated)~~ (*setup key*) activate tagging.

activate/struct-dest (*setup key*) The key allows to suppress the creation of structure destinations
~~ct-dest (deprecated)~~ (*setup key*)
debug/log (*setup key*) The debug/log key takes currently the values none, v, vv, vvv, all. More details are in tagpdf-checks.

activate/tagunmarked (*setup key*) This key allows to set if (in luamode) unmarked text should be marked up as artifact.
~~nmarked (deprecated)~~ (*setup key*) The initial value is true.

activate/softhyphen (*setup key*) This key allows to activates automatic handling of hyphens inserted by hyphenation. It is only used in luamode. It accepts the following values:

- false and off deactivates the automatic handling.

- char Replaces the hyphens by U+00AD if the font contains this glyph. Note that in some newer fonts U+00AD is an invisible character and so this could led to disappearing hyphens.

- **artifact** (default value) This keeps the hard hyphen U+002D but surrounds it by an Artifact-MC. This only works if the document is tagged.

- **true**, **on** do the same as **artifact**.

If tagging is not activate the softhyphen handling is not activated, by default. The hyphen stays a hard-hyphen. If is possible to use the **char** mode with

`\DocumentMetadata{tagging=off,tagging-setup={activate/softhyphen=char},}`

As `\tagpdfsetup` is deactivated at the begin of the class, there is no interface to switch in the document.

If tagging is active the default is **artifact** mode, this requires that **tagunmarked** is active too. The mode can be switched at any time with `\tagpdfsetup`. Switching on and off the softhyphen is a global operation, a switch between artifact and char more is a local operation.

llect-link-artifacts (*setup key*)   PDF/UA requires that link annotations are in a Link (or Reference) structure element even if there are connected to text that is marked as artifact, as is the text in the header and footer. Fulfilling this requirement has sadly the bad side effect that a screen reader will, e.g., read suddenly "Link" between paragraphs when it encounters a header at a page break. Leaving the annotations untagged gives a better reading but leads to complains of the UA-validators. With lualatex it is possible to detect such untagged link annotations and to move them into a structure at the end of the document, so that they at least no longer interrupt the reading. The setting is on by default, but can be deactivated (in the preamble) with this key. It knows currently only the value **off**.

page/tabsorder (*setup key*)
bsorder (deprecated) (*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} {2026-03-19} {0.99z}
4   { LaTeX kernel code for PDF tagging }
5
6 \IfPDFManagementActiveF
7   {
8     \PackageError{tagpdf}
9     {
10       PDF~resource~management~is~no~active!\MessageBreak
11       tagpdf~will~no~work.
12     }
```

```
13        {
14          Activate~it~with \MessageBreak
15          \string\DocumentMetadata{<options>}\MessageBreak
16          before~\string\documentclass
17        }
18      }
19  ⟨/package⟩
```
<*debug>
```
20  \ProvidesExplPackage {tagpdf-debug} {2026-03-19} {0.99z}
21    { debug code for tagpdf }
22  \@ifpackageloaded{tagpdf}{}{\PackageWarning{tagpdf-debug}{tagpdf~not~loaded,~quitting}\endinp
```
</debug> We map the internal module name "tag" to "tagpdf" in messages.
```
23  ⟨∗package⟩
24  \prop_gput:Nnn \g_msg_module_name_prop { tag }{ tagpdf }
25  ⟨/package⟩
```
Debug mode has its special mapping:
```
26  ⟨∗debug⟩
27  \prop_gput:Nnn \g_msg_module_type_prop { tag / debug} {}
28  \prop_gput:Nnn \g_msg_module_name_prop { tag / debug }{tagpdf~DEBUG}
29  ⟨/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
```
30  ⟨∗base⟩
31  \ProvidesExplPackage {tagpdf-base} {2026-03-19} {0.99z}
32    {part of tagpdf - provide base, no-op versions of the user commands }
33  ⟨/base⟩
```

3   Package options

The boolean is kept for now for compatibility, can go in 2026.
```
34  ⟨∗package⟩
35  \bool_new:N\g__tag_mode_lua_bool
36  \sys_if_engine_luatex:T {\bool_gset_true:N \g__tag_mode_lua_bool}
37  \DeclareOption {luamode}   { }
38  \DeclareOption {genericmode}{ }
39  \ProcessOptions
```

4   Packages

To be on the safe side for now, load also the base definitions
```
40  \RequirePackage{tagpdf-base}
41  ⟨/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:

```
42 ⟨*base⟩
43 \cs_new_protected:Npn \__tag_whatsits: {}
44 \AddToHook{begindocument}
45  {
46   \str_case:onF { \c_sys_backend_str }
47    {
48     { luatex  } { \cs_set_protected:Npn \__tag_whatsits: {} }
49     { dvisvgm } { \cs_set_protected:Npn \__tag_whatsits: {} }
50    }
51    {
52      \cs_set_protected:Npn \__tag_whatsits: {\tex_special:D {} }
53    }
54  }
55 ⟨/base⟩
```

4.1 a LastPage label

With LaTeX 2025-06-01 we no longer need a special version as the label is now written directly. To avoid problems with the xr package, we undefine the label command before reading the aux-file.

```
56 ⟨*package⟩
57 \AddToHook{begindocument/before}{\cs_undefine:N\r__tagtag@LastPage}
58 \AddToHook{enddocument/afterlastpage}
59  {\property_record:nn{@tag@LastPage}{abspage,tagmcabs,tagstruct}}
```

5 Variables

`\l__tag_tmpa_tl`
`\l__tag_tmpb_tl`
`\l__tag_tmpc_tl`
`\l_tag_tmp_unused_tl`   `\l_tag_Ref_tmpa_tl`
`\l__tag_get_tmpc_tl`
`\l__tag_get_parent_tmpa_tl`
`\l__tag_get_parent_tmpb_tl`
`\l__tag_get_parent_tmpc_tl`
`\l__tag_get_child_tmpa_tl`
`\l__tag_get_child_tmpb_tl`
`\l__tag_get_child_tmpc_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`

A few temporary variables

```
60 \tl_new:N    \l__tag_tmpa_tl
61 \tl_new:N    \l__tag_tmpb_tl
62 \tl_new:N    \l__tag_tmpc_tl
63 \tl_new:N    \l__tag_tmp_unused_tl
64 \tl_new:N    \l__tag_Ref_tmpa_tl
65 \tl_new:N    \l__tag_get_tmpc_tl
66 \tl_new:N    \l__tag_get_parent_tmpa_tl
67 \tl_new:N    \l__tag_get_parent_tmpb_tl
68 \tl_new:N    \l__tag_get_parent_tmpc_tl
69 \tl_new:N    \l__tag_get_child_tmpa_tl
70 \tl_new:N    \l__tag_get_child_tmpb_tl
71 \tl_new:N    \l__tag_get_child_tmpc_tl
72 \str_new:N   \l__tag_tmpa_str
73 \prop_new:N  \l__tag_tmpa_prop
74 \seq_new:N   \l__tag_tmpa_seq
75 \seq_new:N   \l__tag_tmpb_seq
76 \clist_new:N \l__tag_tmpa_clist
77 \int_new:N   \l__tag_tmpa_int
78 \box_new:N   \l__tag_tmpa_box
79 \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`

```
80 \clist_const:Nn \c__tag_property_mc_clist      {tagabspage,tagmcabs,tagmcid}
81 \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.

```
82 \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, 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.

```
83 \bool_new:N \g__tag_active_space_bool
84 \bool_new:N \g__tag_active_mc_bool
85 \bool_new:N \g__tag_active_tree_bool
86 \bool_new:N \g__tag_active_struct_bool
87 \bool_new:N \g__tag_active_struct_dest_bool
88 \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.

```
89 \bool_new:N \l__tag_active_mc_bool
90 \bool_set_true:N \l__tag_active_mc_bool
91 \bool_new:N \l__tag_active_struct_bool
92 \bool_set_true:N \l__tag_active_struct_bool
93 \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.

```
94 \bool_new:N \g__tag_tagunmarked_bool
```

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

`\g__tag_softhyphen_bool`   This boolean controls if the code should try to automatically handle hyphens from hyphenation. It is currently only used in luamode.

```
95 \bool_new:N \g__tag_softhyphen_bool
96 \bool_gset_true:N \g__tag_softhyphen_bool
```

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

`\g__tag_unique_cnt_int` If tagpdf has to create unique names (e.g. for object names when embedding files) it can use this integer to get an unique name. At every use it should be increased

```
97 \int_new:N \g__tag_unique_cnt_int
```

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

6 Variants of l3 commands

```
98  \prg_generate_conditional_variant:Nnn \pdf_object_if_exist:n {e}{T,F,TF}
99  \cs_generate_variant:Nn \pdf_object_ref:n {e}
100 \cs_generate_variant:Nn \pdfannot_dict_put:nnn {nne}
101 \cs_generate_variant:Nn \pdffile_embed_stream:nnn {nee,oee}
102 \cs_generate_variant:Nn \prop_gput:Nnn {Nee,Nen} %** unneeded
103 \cs_generate_variant:Nn \prop_put:Nnn  {Nee}     %** unneeded
104 \cs_generate_variant:Nn \prop_item:Nn {No,Ne}    %**  unneeded
105 \cs_generate_variant:Nn \seq_set_split:Nnn{Nno}
106 \cs_generate_variant:Nn \str_set_convert:Nnnn {Nonn, Noon, Nnon }
107 \cs_generate_variant:Nn \clist_map_inline:nn {on}
108 \cs_generate_variant:Nn \pdffile_embed_file:nnn {eee}
```

7 Label and Reference commands

The code uses mostly the kernel properties but need a few local variants.

`\__tag_property_record:nn` The command to record a property while preserving the spaces similar to the standard `\label`.

```
109    \cs_new_protected:Npn \__tag_property_record:nn #1#2
110      {
111        \@bsphack
112        \property_record:nn{#1}{#2}
113        \@esphack
114      }
115
```

And a few variants

```
116 \cs_generate_variant:Nn \property_ref:nnn {enn}
117 \cs_generate_variant:Nn \property_ref:nn  {en}
118 \cs_generate_variant:Nn \__tag_property_record:nn {en,eo}
```

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

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

```
119 \cs_new:Npn \__tag_property_ref_lastpage:nn #1 #2
120   {
121     \property_ref:nnn {@tag@LastPage}{#1}{#2}
122   }
```

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

8 Setup label attributes

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`.

```
123 \property_new:nnnn
124   { tagstruct } { now }
125   {1} { \int_use:N \c@g__tag_struct_abs_int }
126 \property_new:nnnn  { tagstructobj } { now }  {}
127   {
128     \pdf_object_ref_indexed:nn { __tag/struct } { \c@g__tag_struct_abs_int }
129   }
130 \property_new:nnnn
131   { tagabspage } { shipout }
132   {0} { \int_use:N \g_shipout_readonly_int }
133 \property_new:nnnn  { tagmcabs } { now }
134   {0} { \int_use:N \c@g__tag_MCID_abs_int }
135
136 \flag_new:n { __tag/mcid }
137 \property_new:nnnn  {tagmcid } { shipout }
138   {0} { \flag_height:n { __tag/mcid } }
139
```

(*End of definition for* `tagstruct` *and others. These functions are documented on page 8.*)

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.

```
140 \cs_set_eq:NN \__tag_prop_new:N          \prop_new:N
141 \cs_set_eq:NN \__tag_prop_new_linked:N \prop_new_linked:N
142 \cs_set_eq:NN \__tag_seq_new:N           \seq_new:N
143 \cs_set_eq:NN \__tag_prop_gput:Nnn       \prop_gput:Nnn
144 \cs_set_eq:NN \__tag_seq_gput_right:Nn \seq_gput_right:Nn
145 \cs_set_eq:NN \__tag_seq_gput_left:Nn  \seq_gput_left:Nn
146 \cs_set_eq:NN \__tag_seq_item:cn         \seq_item:cn
147 \cs_set_eq:NN \__tag_prop_item:cn        \prop_item:cn
148 \cs_set_eq:NN \__tag_seq_show:N          \seq_show:N
149 \cs_set_eq:NN \__tag_prop_show:N         \prop_show:N
150 % cnx temporary needed for latex-lab-graphic code
151 \cs_generate_variant:Nn \__tag_prop_gput:Nnn      { Nen, Nee, Nne, Nno, cnn, cen, cne, cno, c
152 \cs_generate_variant:Nn \__tag_seq_gput_right:Nn  { Ne , No, cn, ce }
153 \cs_generate_variant:Nn \__tag_seq_gput_left:Nn   { ce }
154 \cs_generate_variant:Nn \__tag_prop_new:N   { c }
155 \cs_generate_variant:Nn \__tag_seq_new:N    { c }
156 \cs_generate_variant:Nn \__tag_seq_show:N   { c }
157 \cs_generate_variant:Nn \__tag_prop_show:N  { c }
```

158 ⟨/package⟩

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

10   General tagging commands

\tag_suspend:n
\tag_resume:n
\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. The label is not expand so can e.g. be a single command token.

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.

```
159 ⟨*package | debug⟩
160 ⟨package⟩\int_new:N \l__tag_tag_stop_int
```

\l__tag_tag_stop_int

```
161 \cs_set_protected:Npn \tag_stop:
162   {
163 ⟨debug⟩      \msg_note:nne {tag / debug }{tag-suspend}{ \int_use:N \l__tag_tag_stop_int }
164     \int_incr:N \l__tag_tag_stop_int
165     \bool_set_false:N \l__tag_active_struct_bool
166     \bool_set_false:N \l__tag_active_mc_bool
167     \bool_set_false:N \l__tag_active_socket_bool
168     \__tag_stop_para_ints:
169   }
170 \cs_set_protected:Npn \tag_start:
171   {
172     \int_if_zero:nF { \l__tag_tag_stop_int } { \int_decr:N \l__tag_tag_stop_int }
173     \int_if_zero:nT { \l__tag_tag_stop_int }
174       {
175         \bool_set_true:N \l__tag_active_struct_bool
176         \bool_set_true:N \l__tag_active_mc_bool
177         \bool_set_true:N \l__tag_active_socket_bool
178         \__tag_start_para_ints:
179       }
180 ⟨debug⟩     \msg_note:nne {tag / debug }{tag-resume}{ \int_use:N \l__tag_tag_stop_int }
181   }
182 \cs_set_eq:NN\tagstop\tag_stop:
183 \cs_set_eq:NN\tagstart\tag_start:
184 \cs_set_protected:Npn \tag_suspend:n #1
185   {
186 ⟨debug⟩     \msg_note:nnee {tag / debug }{tag-suspend}
187 ⟨debug⟩        { \int_use:N \l__tag_tag_stop_int }{\exp_not:n{#1}}
188     \int_incr:N \l__tag_tag_stop_int
189     \bool_set_false:N \l__tag_active_struct_bool
190     \bool_set_false:N \l__tag_active_mc_bool
191     \bool_set_false:N \l__tag_active_socket_bool
192     \__tag_stop_para_ints:
193   }
194 \cs_set_eq:NN \tag_stop:n \tag_suspend:n
```

14

```
195  \cs_set_protected:Npn \tag_resume:n #1
196    {
197      \int_if_zero:nF { \l__tag_tag_stop_int } { \int_decr:N \l__tag_tag_stop_int }
198      \int_if_zero:nT { \l__tag_tag_stop_int }
199        {
200          \bool_set_true:N \l__tag_active_struct_bool
201          \bool_set_true:N \l__tag_active_mc_bool
202          \bool_set_true:N \l__tag_active_socket_bool
203          \__tag_start_para_ints:
204        }
205 ⟨debug⟩      \msg_note:nnee {tag / debug }{tag-resume}
206 ⟨debug⟩        { \int_use:N \l__tag_tag_stop_int }{\exp_not:n{#1}}
207    }
208  \cs_set_eq:NN \tag_start:n \tag_resume:n
209 ⟨/package | debug⟩
210 ⟨∗base⟩
211  \cs_new_protected:Npn \tag_stop:{}
212  \cs_new_protected:Npn \tag_start:{}
213  \cs_new_protected:Npn \tagstop{}
214  \cs_new_protected:Npn \tagstart{}
215  \cs_new_protected:Npn \tag_stop:n  #1 {}
216  \cs_new_protected:Npn \tag_start:n #1 {}
```

Until the commands are provided by the kernel we provide them here too

```
217  \cs_set_eq:NN \tag_suspend:n \tag_stop:n
218  \cs_set_eq:NN \tag_resume:n  \tag_start:n
219 ⟨/base⟩
```

(*End of definition for* `\tag_suspend:n` *and others. These functions are documented on page 7.*)

11   Handling link artifacts

Links that should be artifacts, e.g. in header or an overlay frame, can not be simply kept untagged as the validators then complain about the link annotations that are not in a Link structure. But tagging them gives a bad reading experience as they can interrupt the text. Probably long term the best is not to interrupt tagging inside an Artifact structure and to move these structures to the end of the structure tree to keep them out of the way. But with luatex we have an easy option to detect untagged link annotations and so can move them to the end of the structure tree too and this is done here.

```
220 ⟨∗package⟩
221  \cs_new_protected:Npn \__tag_activate_linkbin:
222    {
223      \sys_if_engine_luatex:T
224        {
225          \tagstructbegin{tag=Link,stash}
226          \tl_const:Ne \c__tag_struct_link_bin_tl {\int_use:N\c@g__tag_struct_abs_int}%
227          \tagstructend
228          \AddToHook{tagpdf/finish/before}[tagpdf/linkbin]
229           {
230             \par
231             \int_compare:nNnT {\directlua{tex.sprint(ltx.__tag.func.linkbin())}}>{0}
232              {
233                \tagstructbegin{tag=Artifact,title=Ignored~link~annotations}
```

15

```
234            \tag_struct_use_num:o{\c__tag_struct_link_bin_tl}
235            \tagstructend
236          }
237        }
238      }
239    }
240 \AddToHook{begindocument/end}[tagpdf/linkbin]{\__tag_activate_linkbin:}
241 ⟨/package⟩
```

12   Keys for tagpdfsetup

TODO: the log-levels must be sorted

activate/mc (*setup key*)   Keys to (globally) activate tagging. `activate/spaces` activates the additional parsing
activate/tree (*setup key*)   needed for interword spaces. It is defined in tagpdf-space. `activate/struct-dest` allows
activate/struct (*setup key*)   to activate or suppress structure destinations.
activate/all (*setup key*)
activate/struct-dest (*setup key*)

```
242 ⟨*package⟩
243 \keys_define:nn { __tag / setup }
244   {
245     activate/mc      .bool_gset:N = \g__tag_active_mc_bool,
246     activate/tree    .bool_gset:N = \g__tag_active_tree_bool,
247     activate/struct .bool_gset:N = \g__tag_active_struct_bool,
248     activate/all     .meta:n =
249       {activate/mc={#1},activate/tree={#1},activate/struct={#1}},
250     activate/all .default:n = true,
251     activate/struct-dest .bool_gset:N = \g__tag_active_struct_dest_bool,
```

old, deprecated names

```
252     activate-mc      .bool_gset:N = \g__tag_active_mc_bool,
253     activate-tree    .bool_gset:N = \g__tag_active_tree_bool,
254     activate-struct .bool_gset:N = \g__tag_active_struct_bool,
255     activate-all     .meta:n =
256       {activate/mc={#1},activate/tree={#1},activate/struct={#1}},
257     activate-all .default:n = true,
258     no-struct-dest .bool_gset_inverse:N = \g__tag_active_struct_dest_bool,
```

debug/show (*setup key*)   Subkeys/values are defined in various other places.

```
259     debug/show             .choice:,
```

debug/log (*setup key*)   The `log` takes currently the values `none`, `v`, `vv`, `vvv`, `all`. The description of the log
debug/uncompress (*setup key*)   levels is in tagpdf-checks.
log (deprecated) (*setup key*)
compress (deprecated) (*setup key*)

```
260     debug/log             .choice:,
261     debug/log / none      .code:n = {\int_set:Nn \l__tag_loglevel_int { 0 }},
262     debug/log / v         .code:n =
263       {
264         \int_set:Nn \l__tag_loglevel_int { 1 }
265         \cs_set_protected:Nn \__tag_check_typeout_v:n { \iow_term:e {##1} }
266       },
267     debug/log / vv        .code:n = {\int_set:Nn \l__tag_loglevel_int { 2 }},
268     debug/log / vvv       .code:n = {\int_set:Nn \l__tag_loglevel_int { 3 }},
269     debug/log / all       .code:n = {\int_set:Nn \l__tag_loglevel_int { 10 }},
270     debug/uncompress .code:n = { \pdf_uncompress:  },
```

deprecated but still needed as the documentmetadata key argument uses it.

```
271    log              .meta:n = {debug/log={#1}},
272    uncompress       .code:n = { \pdf_uncompress:  },
```

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

```
273    activate/tagunmarked     .bool_gset:N = \g__tag_tagunmarked_bool,
274    activate/tagunmarked     .initial:n  = true,
```

deprecated name

```
275    tagunmarked .bool_gset:N = \g__tag_tagunmarked_bool,
```

activate/softhyphen (*setup key*) This key activates (in luamode) the handling of soft hyphens.

```
276    activate/softhyphen .choice:,
277    activate/softhyphen/false .code:n = {\bool_gset_false:N \g__tag_softhyphen_bool},
278    activate/softhyphen/off .code:n = {\bool_gset_false:N \g__tag_softhyphen_bool},
279    activate/softhyphen/char  .code:n =
280     {
281       \bool_gset_true:N \g__tag_softhyphen_bool
282       \sys_if_engine_luatex:T
283        {
284         \lua_now:e
285          {
286            tex.setattribute
287             (
288               luatexbase.attributes.l__tag_softhyphen_attr,
289               1
290             )
291          }
292        }
293     },
294    activate/softhyphen/artifact   .code:n =
295     {
296       \bool_gset_true:N \g__tag_softhyphen_bool
297       \sys_if_engine_luatex:T
298        {
299         \lua_now:e
300          {
301            tex.setattribute
302             (
303               luatexbase.attributes.l__tag_softhyphen_attr,
304               -2147483647
305             )
306          }
307        }
308     },
309    activate/softhyphen/true   .code:n =
310     {
311       \bool_gset_true:N \g__tag_softhyphen_bool
312       \sys_if_engine_luatex:T
313        {
314         \lua_now:e
315          {
316            tex.setattribute
```

17

```
317                 (
318                   luatexbase.attributes.l__tag_softhyphen_attr,
319                   -2147483647
320                 )
321             }
322           }
323         },
324       activate/softhyphen/on   .code:n =
325        {
326          \bool_gset_true:N \g__tag_softhyphen_bool
327          \sys_if_engine_luatex:T
328           {
329            \lua_now:e
330             {
331               tex.setattribute
332               (
333                 luatexbase.attributes.l__tag_softhyphen_attr,
334                 -2147483647
335               )
336            }
337          }
338        },
339       activate/softhyphen      .default:n  = artifact,
```

llect-link-artifacts (*setup key*)

```
340       activate/collect-link-artifacts/off .code:n =
341         {\RemoveFromHook{begindocument/end}[tagpdf/linkbin]},
```

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

```
342       page/tabsorder          .choice:,
343       page/tabsorder / row        .code:n =
344         \pdfmanagement_add:nnn { Page } {Tabs}{/R},
345       page/tabsorder / column     .code:n =
346         \pdfmanagement_add:nnn { Page } {Tabs}{/C},
347       page/tabsorder / structure .code:n =
348         \pdfmanagement_add:nnn { Page } {Tabs}{/S},
349       page/tabsorder / none       .code:n =
350         \pdfmanagement_remove:nn {Page} {Tabs},
351       page/tabsorder          .initial:n = structure,
```

deprecated name

```
352       tabsorder .meta:n = {page/tabsorder={#1}},
353    }
```

13 loading of engine/more dependent code

```
354 \sys_if_engine_luatex:T
355   {
356     \file_input:n {tagpdf-luatex.def}
357   }
358 ⟨/package⟩
```

```
359 ⟨∗mcloading⟩
360 \bool_if:NTF \g__tag_mode_lua_bool
361   {
362     \RequirePackage {tagpdf-mc-code-lua}
363   }
364   {
365     \RequirePackage {tagpdf-mc-code-generic} %
366   }
367 ⟨/mcloading⟩
368 ⟨∗debug⟩
369 \bool_if:NTF \g__tag_mode_lua_bool
370   {
371     \RequirePackage {tagpdf-debug-lua}
372   }
373   {
374     \RequirePackage {tagpdf-debug-generic} %
375   }
376 ⟨/debug⟩
```

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

Ulrike Fischer
Version 0.99z, released 2026-03-19

Part II

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:NTF` ⋆

`\tag_if_box_tagged:NTF` ⟨box⟩ {⟨true code⟩} {⟨false code⟩}

This tests if a box contains tagging commands. It relies currently on that the code, that saved the box, correctly sets 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 responsibility 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

`\tag_if_in_p:n` ⋆
`\tag_if_in:nTF` ⋆

`\tag_if_in:n {⟨standard structure type⟩}`

This command tests if we are currently "in" a specific structure by checking if the argument is on the stack. The argument must be a standard structure type as no role mapping is applied.

The main use case is to check if there is an open P structure that should be closed.

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.
`tree-statistic`	Message issued at the end of the compilation showing the number of objects to write
`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-unknown-NS`
`role-tag`
`new-tag`
`role-parent-child-result`
`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} {2026-03-19} {0.99z}
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* 22.)

mc-tag-missing   If the tag is missing

```
8 \msg_new:nnn { tag } {mc-tag-missing} { MC-tag~missing;~#1~used~instead~-~mcid~#2 }
```

(*End of definition for* mc-tag-missing. *This function is documented on page* 22.)

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* 22.)

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* 22.)

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* 22.)

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

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

(*End of definition for* mc-pushed *and* mc-popped. *These functions are documented on page* 22.)

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* 22.)

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 23.*)

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 23.*)

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 23.*)

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 23.*)

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 23.*)

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 23.*)

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 23.*)

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 23.*)

struct-Ref-unknown This message is issued at the end, when the Ref keys are updated. TODO: in debug mode it should report more info about the structure.

```
42 \msg_new:nnn { tag } {struct-Ref-unknown}
43   {
44       #1~has~no~related~structure.\\
45      /Ref~not~updated.
46   }
```

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

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

```
47 \msg_new:nnn { tag } {tree-struct-still-open}
48    {
49     There~are~still~open~structures~on~the~stack!\\
50     The~stack~contains~\seq_use:Nn\g__tag_struct_tag_stack_seq{,}.\\
51     The~structures~are~automatically~closed,\\
52     but~their~nesting~can~be~wrong.
53    }
```

(*End of definition for* `tree-struct-still-open`. *This function is documented on page* `23`.)

tree-statistic Message issued at the end showing the estimated number of structures and MC-childs

```
54 \msg_new:nnn { tag } {tree-statistic}
55    {
56     Finalizing~the~tagging~structure:\\
57     Writing~out~\c_tilde_str
58     \int_use:N\c@g__tag_struct_abs_int\c_space_tl~structure~objects\\
59     with~\c_tilde_str
60     \int_use:N\c@g__tag_MCID_abs_int\c_space_tl'MC'~leaf~nodes.\\
61     Be~patient~if~there~are~lots~of~objects!
62    }
63 ⟨/package⟩
```

(*End of definition for* `tree-statistic`. *This function is documented on page* `23`.)

The following messages are only needed in debug mode.

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

```
64 ⟨∗debug⟩
65 \msg_new:nnn { tag/debug } { show-struct }
66    {
67     ==========================\\
68     The~structure~#1~
69     \tl_if_empty:nTF {#2}
70       { is~empty \\>~ . }
71       { contains: #2  }
72     \\
73    }
74 \msg_new:nnn { tag/debug } { show-kids }
75    {
76     The~structure~has~the~following~kids:
77     \tl_if_empty:nTF {#2}
78       { \\>~ NONE }
79       { #2  }
80     \\
81     =========================
```

26

```
82      }
83   ⟨/debug⟩
```
(*End of definition for* `show-struct` *and* `show-kids`*. These functions are documented on page 23.*)

3.3 Attributes

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

attr-unknown

```
84  ⟨*package⟩
85  \msg_new:nnn { tag } {attr-unknown}  { attribute~#1~is~unknown}
```
(*End of definition for* `attr-unknown`*. This function is documented on page 23.*)

3.4 Roles

role-missing   Warning message if either the tag or the role is missing
role-unknown
```
86  \msg_new:nnn { tag } {role-missing}     { tag~#1~has~no~role~assigned  }
87  \msg_new:nnn { tag } {role-unknown}     { role~#1~is~not~known  }
88  \msg_new:nnn { tag } {role-unknown-tag} { tag~#1~is~not~known  }
89  \msg_new:nnn { tag } {role-unknown-NS}  { \tl_if_empty:nTF{#1}{Empty~NS}{NS~#1~is~not~known}
```
role-unknown-tag
role-unknown-NS
(*End of definition for* `role-missing` *and others. These functions are documented on page 23.*)

role-parent-child-check   This is an info message that inform which elements are checked, typically used to show the original tags, not the rolemapped one.
```
90  \msg_new:nnn { tag } {role-parent-child-check}
91    { Checking~Parent-Child~'#1'~-->~'#2' }
```
(*End of definition for* `role-parent-child-check`*. This function is documented on page ??.*)

role-parent-child-result   This is info and warning message about the containment rules between child and parent tags.
```
92  \msg_new:nnn { tag } {role-parent-child-result}
93    { Parent-Child~'#1'~-->~'#2'.\\Relation~is~#3~\msg_line_context:}
```
(*End of definition for* `role-parent-child-result`*. This function is documented on page 23.*)

role-struct-parent-child-forbidden   The most important message is that the relation is not allowed between two structures. Argument #1 is the parent structure number, #2 is the child structure number, #3 NS:tag info of the parent (TODO perhaps rolemapped), #4 NS:tag of the child. (TODO)
```
94  \msg_new:nnn { tag } {role-struct-parent-child-forbidden}
95    {
96      Parent-Child~'#3'~-->~'#4'.\\
97      Relation~is~not~allowed! ~\msg_line_context:\\
98      struct~#1,~
99      \exp_last_unbraced:Ne\use_i:nn  { \prop_item:cn{ g__tag_struct_#1_prop}{tag} }
100     \c_space_tl-->\c_space_tl
101     struct~#2,~
102     \exp_last_unbraced:Ne\use_i:nn  { \prop_item:cn{ g__tag_struct_#2_prop}{tag} }
103   }
```
(*End of definition for* `role-struct-parent-child-forbidden`*. This function is documented on page ??.*)

27

role-MC-child-forbidden In case that MC is forbidden we use a special message. Argument #1 is the parent structure number. #2 NS:tag of the parent,

```
104 \msg_new:nnn { tag } {role-MC-child-forbidden}
105   {
106     Parent-Child~'#2'~-->~'MC~(real~content)'.\\
107     Relation~is~not~allowed! ~\msg_line_context:\\
108     struct~#1,~
109     \exp_last_unbraced:Ne\use_i:nn { \prop_item:cn{ g__tag_struct_#1_prop}{tag} }
110   }
```

(*End of definition for* role-MC-child-forbidden. *This function is documented on page* **??**.)

role-parent-child-forbidden The most important message is that the relation is not allowed. Argument #1 is the parent structure number. #2 NS:tag of the parent, #3 NS:tag of the child.

```
111 \msg_new:nnn { tag } {role-parent-child-forbidden}
112   {
113     Parent-Child~'#2'~-->~'#3'.\\
114     Relation~is~not~allowed! ~\msg_line_context:\\
115     struct~#1,~\prop_item:cn{ g__tag_struct_#1_prop}{S}
116     \c_space_tl
117     \str_if_eq:nnF{#3}{MC~(realcontent)}
118       {-->~struct~\int_eval:n {\c@g__tag_struct_abs_int}}
119   }
```

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

__tag_check_forbidden_parent_child:nnnn

```
120 \cs_new_protected:Npn \__tag_check_forbidden_parent_child:nnnn #1#2#3#4
121 % #1 check number, #2 number of parent struct
122 % #3 parent info,  #4 child info
123  {
124    \int_compare:nNnT {#1 } <0
125      {
126        \msg_warning:nneee
127          { tag }
128          {role-parent-child-forbidden}
129          { #2}
130          { #3 }
131          { #4 }
132      }
133  }
134 \cs_generate_variant:Nn \__tag_check_forbidden_parent_child:nnnn {nnee}
135
136 % new with structure numbers:
137 \cs_new_protected:Npn \__tag_check_struct_forbidden_parent_child:nnn #1#2#3
138 % #1 check number,
139 % #2 number of parent struct
140 % #3 number of child struct
141  {
142    \int_compare:nNnT {#1 } <0
143      {
144        \prop_get:cnN {g__tag_struct_#2_prop}{parentrole}\l__tag_get_parent_tmpc_tl
145        \prop_get:cnN {g__tag_struct_#3_prop}{rolemap}\l__tag_get_child_tmpc_tl
146        \msg_warning:nneeee
147          { tag }
```

```
148              {role-struct-parent-child-forbidden}
149              { #2 }
150              { #3 }
151              {
152                \exp_last_unbraced:No \use_ii:nn  { \l__tag_get_parent_tmpc_tl }
153                :
154                \exp_last_unbraced:No \use_i:nn  {\l__tag_get_parent_tmpc_tl }
155              }
156              {
157                \exp_last_unbraced:No \use_ii:nn { \l__tag_get_child_tmpc_tl }
158                :
159                \exp_last_unbraced:No \use_i:nn { \l__tag_get_child_tmpc_tl }
160              }
161          }
162    }
163  \cs_generate_variant:Nn\__tag_check_struct_forbidden_parent_child:nnn{onn}
```
(End of definition for `\__tag_check_forbidden_parent_child:nnnn`*.)*

role-parent-child-unresolved   If a structure is stashed and then used later and its root is one of Part, Div or NonStruct, then we can not check the parent-child rules. This would require to know all children. In this case we only warn. resolved a Argument #1 is the parent structure number. #2 NS:tag of the parent, #3 NS:tag of the child.

```
164  \msg_new:nnn { tag } {role-parent-child-unresolved}
165    {
166      Structure~\int_eval:n {\c@g__tag_struct_abs_int}~was~moved~into~structure~#1.\\
167      Parent-Child~'#2'~--->~'#3'~can~not~checked.
168    }
```
(End of definition for `role-parent-child-unresolved`*. This function is documented on page* **??***.)*

__tag_check_unresolved_parent_child:nnnn

```
169  \cs_new_protected:Npn \__tag_check_unresolved_parent_child:nnnn #1#2#3#4
170  % #1 check number, #2 number of parent struct
171  % #3 parent info, #4 child info
172    {
173      \int_compare:nNnT { #1 } = {\c__tag_role_rule_checkparent_tl}
174        {
175          \msg_warning:nneee
176            { tag }
177            {role-parent-child-unresolved}
178            { #2 }
179            { #3 }
180            { #4 }
181        }
182    }
```
(End of definition for `\__tag_check_unresolved_parent_child:nnnn`*.)*

tag/check/parent-child   Sockets used around the parent-child checks so that we can disable them.
tag/check/parent-child-end
```
183  \socket_new:nn{tag/check/parent-child}{1}
184  \socket_new:nn{tag/check/parent-child-end}{0}
185  \socket_new_plug:nnn {tag/check/parent-child-end}{check}
186    {
187      \sys_if_engine_luatex:T
188        {
```

```
189        \lua_now:e
190          {
191            ltx.__tag.func.check_parent_child_rules ( 2 )
192          }
193      }
194    }
```

And a key to disable the check

```
195 \keys_define:nn { __tag / setup}
196    {
197      debug / parent-child-check .choice:,
198      debug / parent-child-check / on .code:n =
199       {
200         \socket_assign_plug:nn {tag/check/parent-child}{identity}
201       },
202      debug / parent-child-check / off .code:n=
203       {
204         \socket_assign_plug:nn {tag/check/parent-child}{noop}
205         \socket_assign_plug:nn {tag/check/parent-child-end}{noop}
206       },
207      debug / parent-child-check / atend .code:n=
208       {
209         \socket_assign_plug:nn {tag/check/parent-child}{noop}
210         \socket_assign_plug:nn {tag/check/parent-child-end}{check}
211       }
212    }
```

(*End of definition for* `tag/check/parent-child` *and* `tag/check/parent-child-end`*. These functions are documented on page* **??***.*)

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

```
213 \msg_new:nnn { tag } {role-remapping}
214    { remapping~tag~to~#1 }
```

(*End of definition for* `role-remapping`*. This function is documented on page* *23.*)

role-tag Info messages.
new-tag

```
215 \msg_new:nnn { tag } {role-tag}          { mapping~tag~#1~to~role~#2  }
216 \msg_new:nnn { tag } {new-tag}           { adding~new~tag~#1 }
217 \msg_new:nnn { tag } {read-namespace}    { reading~namespace~definitions~tagpdf-
    ns-#1.def }
218 \msg_new:nnn { tag } {namespace-missing}{ namespace~definitions~tagpdf-ns-#1.def~not~found }
219 \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* *23.*)

3.5 Miscellaneous

wrong-pdfversion Used a begin document if the pdfversion has been changed after the reading.

```
220 \msg_new:nnn { tag } {wrong-pdfversion}
221    {
222      The~PDF~version~has~changed~after~the~loading~of~tagpdf~from~#1~to~#2.\\
223      The~structure~will~be~faulty.\\
224      Trying~to~revert~to~#1.
225    }
```

(*End of definition for* `wrong-pdfversion`*. This function is documented on page* **??***.*)

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

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

(*End of definition for* `tree-mcid-index-wrong`. *This function is documented on page 23.*)

sys-no-interwordspace   Currently only pdflatex and lualatex have some support for real spaces.

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

(*End of definition for* `sys-no-interwordspace`. *This function is documented on page 23.*)

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

```
230 \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.

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

(*End of definition for* `para-hook-count-wrong`. *This function is documented on page 23.*)

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`.

```
235 ⟨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 20.*)

5   PDF version check

```
236 ⟨*package⟩
237 \tl_const:Ne\c__tag_check_pdfversion_tl {\pdf_version:}
238 \AddToHook{begindocument/before}
239   {
240     \tl_if_eq:eeF{\c__tag_check_pdfversion_tl}{\pdf_version:}
241       {
242         \msg_error:nnee {tag}{wrong-pdfversion}{\c__tag_check_pdfversion_tl}{\pdf_version:}
243         \pdf_version_gset:e {\c__tag_check_pdfversion_tl}
244         \pdf_object_unnamed_write:nn{dict}{}
245       }
246   }
247 ⟨/package⟩
```

6 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.

```
248 ⟨*base⟩
249 \cs_if_exist:NF\tag_if_active:T
250  {
251    \prg_new_conditional:Npnn \tag_if_active: { p , T , TF, F }
252      { \prg_return_false: }
253  }
254 ⟨/base⟩
255 ⟨*package⟩
256 \prg_set_conditional:Npnn \tag_if_active: { p , T , TF, F }
257   {
258      \bool_lazy_all:nTF
259        {
260          {\g__tag_active_struct_bool}
261          {\g__tag_active_mc_bool}
262          {\g__tag_active_tree_bool}
263          {\l__tag_active_struct_bool}
264          {\l__tag_active_mc_bool}
265        }
266        {
267          \prg_return_true:
268        }
269        {
270          \prg_return_false:
271        }
272  }
273 ⟨/package⟩
```
(*End of definition for* `\tag_if_active:TF`*. This function is documented on page 20.*)

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

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 responsibility 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.

```
274 ⟨*base⟩
275 \prg_new_conditional:Npnn \tag_if_box_tagged:N #1 {p,T,F,TF}
276     {
277        \tl_if_exist:cTF {l_tag_box_\int_use:N #1_tl}
278          {
279            \int_compare:nNnTF {0\tl_use:c{l_tag_box_\int_use:N #1_tl}}>{0}
280            { \prg_return_true:  }
281            { \prg_return_false: }
282          }
283          {
284            \prg_return_false:
285            % warning??
286          }
287     }
288 ⟨/base⟩
```
(*End of definition for* `\tag_if_box_tagged:NTF`*. This function is documented on page 20.*)

\tag_if_in:n This tests if we are in a specific structure, so if the structure is part of the current stack. The argument must be a standard structure type as no role mapping is applied. The main use case is to test if we are in a paragraph P.

```
289 ⟨*package⟩
290 \prg_new_conditional:Npnn\tag_if_in:n #1 {T,F,TF}
291 {
292   \seq_if_in:NnTF \g__tag_struct_roletag_stack_seq {#1}
293     {\prg_return_true:}
294     {\prg_return_false:}
295 }
296 ⟨/package⟩
```
(*End of definition for* \tag_if_in:n. *This function is documented on page* **??**.)

7 Internal checks

These are checks used in various places in the code.

7.1 checks for active tagging

__tag_check_if_active_mc:*TF*
__tag_check_if_active_struct:*TF*

This checks if mc are active.

```
297 ⟨*package⟩
298 \prg_new_conditional:Npnn \__tag_check_if_active_mc: {T,F,TF}
299   {
300     \bool_lazy_and:nnTF { \g__tag_active_mc_bool } { \l__tag_active_mc_bool }
301       {
302         \prg_return_true:
303       }
304       {
305         \prg_return_false:
306       }
307   }
308 \prg_new_conditional:Npnn \__tag_check_if_active_struct: {T,F,TF}
309   {
310     \bool_lazy_and:nnTF { \g__tag_active_struct_bool } { \l__tag_active_struct_bool }
311       {
312         \prg_return_true:
313       }
314       {
315         \prg_return_false:
316       }
317   }
```
(*End of definition for* __tag_check_if_active_mc:TF *and* __tag_check_if_active_struct:TF.)

7.2 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.

```
318 \cs_new_protected:Npn \__tag_check_structure_has_tag:n #1 %#1 struct num
319   {
320     \prop_get:cnNF
```

```
321        { g__tag_struct_#1_prop }
322        {S}
323        \l__tag_tmp_unused_tl
324        {
325          \msg_error:nn { tag } {struct-missing-tag}
326        }
327    }
```

(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. This always used with commands, so the argument is N.

```
328 \cs_new_protected:Npn \__tag_check_structure_tag:N #1
329    {
330      \prop_get:NoNF \g__tag_role_tags_NS_prop {#1}\l__tag_tmp_unused_tl
331        {
332          \msg_warning:nne { tag } {role-unknown-tag} {#1}
333        }
334    }
```

(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.

```
335 \cs_new_protected:Npn \__tag_check_info_closing_struct:n #1 %#1 struct num
336    {
337      \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
338        {
339          \msg_info:nnn { tag } {struct-show-closing} {#1}
340        }
341    }
342
343 \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.

```
344 \cs_new_protected:Npn \__tag_check_no_open_struct:
345    {
346      \msg_error:nn { tag } {struct-faulty-nesting}
347    }
```

(End of definition for __tag_check_no_open_struct:.)

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

```
348 \cs_new_protected:Npn \__tag_check_struct_used:n #1 %#1 label
349    {
350      \prop_get:cnNT
351        {g__tag_struct_\property_ref:enn{tagpdfstruct-#1}{tagstruct}{unknown}_prop}
352        {parentnum}
353        \l__tag_tmpa_tl
354        {
355          \msg_warning:nnn { tag } {struct-used-twice} {#1}
356        }
357    }
```

(End of definition for __tag_check_struct_used:n.)

7.3 Checks related to roles

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

```
358 \cs_new_protected:Npn \__tag_check_add_tag_role:nn #1 #2 %#1 tag, #2 role
359   {
360     \tl_if_empty:nTF {#2}
361       {
362         \msg_error:nnn { tag } {role-missing} {#1}
363       }
364       {
365         \prop_get:NnNTF \g__tag_role_tags_NS_prop {#2} \l__tag_tmpa_tl
366           {
367             \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
368               {
369                 \msg_info:nnnn { tag } {role-tag} {#1} {#2}
370               }
371           }
372           {
373             \msg_error:nnn { tag } {role-unknown} {#2}
374           }
375       }
376   }
```

Similar with a namespace

```
377 \cs_new_protected:Npn \__tag_check_add_tag_role:nnn #1 #2 #3 %#1 tag/NS, #2 role #3 namespace
378   {
379     \tl_if_empty:nTF {#2}
380       {
381         \msg_error:nnn { tag } {role-missing} {#1}
382       }
383       {
384         \prop_get:cnNTF { g__tag_role_NS_#3_prop } {#2} \l__tag_tmpa_tl
385           {
386             \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
387               {
388                 \msg_info:nnnn { tag } {role-tag} {#1} {#2/#3}
389               }
390           }
391           {
392             \msg_error:nnn { tag } {role-unknown} {#2/#3}
393           }
394       }
395   }
```

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

7.4 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).

```
396 \cs_new_protected:Npn \__tag_check_mc_if_nested:
397   {
398     \__tag_mc_if_in:T
399       {
400         \msg_warning:nne { tag } {mc-nested} { \__tag_get_mc_abs_cnt: }
```

```
401        }
402      }
403
404  \cs_new_protected:Npn \__tag_check_mc_if_open:
405    {
406      \__tag_mc_if_in:F
407        {
408          \msg_warning:nne { tag } {mc-not-open} { \__tag_get_mc_abs_cnt: }
409        }
410    }
```

(*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.

```
411  \cs_new_protected:Npn \__tag_check_mc_pushed_popped:nn #1 #2
412    {
413      \int_compare:nNnT
414        { \l__tag_loglevel_int } ={ 2 }
415        { \msg_info:nne {tag}{mc-#1}{#2} }
416      \int_compare:nNnT
417        { \l__tag_loglevel_int } > { 2 }
418        {
419          \msg_info:nne {tag}{mc-#1}{#2}
420          \seq_log:N \g__tag_mc_stack_seq
421        }
422    }
```

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

`\__tag_check_mc_tag:N` This checks if the mc has a (known) tag, if it is empty (e.g. if due to a call to the output routine, see issue https://github.com/latex3/tagpdf/issues/111) then we fall back to the structure name.

```
423  \cs_new_protected:Npn \__tag_check_mc_tag:N #1  %#1 is var with a tag name in it
424    {
425      \tl_if_empty:NTF #1
426        {
427          \tl_set:No #1 { \g__tag_struct_tag_tl }
428          \msg_info:nnee { tag } {mc-tag-missing} { \g__tag_struct_tag_tl }{ \__tag_get_mc_abs_
429        }
430        {
431         \prop_get:NoNF \g__tag_role_tags_NS_prop {#1}\l__tag_tmp_unused_tl
432           {
433             \msg_warning:nne { tag } {role-unknown-tag} {#1}
434           }
435        }
436    }
```

(*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

36

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??

```
437 \cs_new_protected:Npn \__tag_check_init_mc_used:
438   {
439     \intarray_new:Nn \g__tag_check_mc_used_intarray { 65536 }
440     \cs_gset_eq:NN \__tag_check_init_mc_used: \prg_do_nothing:
441   }
```

(*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.

```
442 \cs_new_protected:Npn \__tag_check_mc_used:n #1 %#1 mcid abscnt
443   {
444     \int_compare:nNnT {\l__tag_loglevel_int} > { 2 }
445       {
446         \__tag_check_init_mc_used:
447         \intarray_gset:Nnn \g__tag_check_mc_used_intarray
448           {#1}
449           { \intarray_item:Nn \g__tag_check_mc_used_intarray {#1} + 1 }
450         \int_compare:nNnT
451           {
452             \intarray_item:Nn \g__tag_check_mc_used_intarray {#1}
453           }
454           >
455           { 1 }
456           {
457             \msg_warning:nnn { tag } {mc-used-twice} {#1}
458           }
459       }
460   }
```

(*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.

```
461 \cs_new_protected:Npn \__tag_check_show_MCID_by_page:
462   {
463     \tl_set:Ne \l__tag_tmpa_tl
464       {
465         \__tag_property_ref_lastpage:nn
466           {abspage}
467           {-1}
468       }
469     \int_step_inline:nnnn {1}{1}
470       {
471         \l__tag_tmpa_tl
472       }
473       {
474         \seq_clear:N \l__tag_tmpa_seq
475         \int_step_inline:nnnn
476           {1}
477           {1}
478           {
479             \__tag_property_ref_lastpage:nn
480               {tagmcabs}
481               {-1}
```

37

```
482              }
483            {
484            \int_compare:nT
485              {
486                \property_ref:enn
487                  {mcid-####1}
488                  {tagabspage}
489                  {-1}
490                =
491                ##1
492              }
493            {
494            \seq_gput_right:Ne \l__tag_tmpa_seq
495              {
496                Page##1-####1-
497                \property_ref:enn
498                  {mcid-####1}
499                  {tagmcid}
500                  {-1}
501              }
502            }
503          }
504        \seq_show:N \l__tag_tmpa_seq
505      }
506  }
```
(*End of definition for* `\__tag_check_show_MCID_by_page:`.)

7.5 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.

```
507 \prg_new_conditional:Npnn \__tag_check_if_mc_in_galley: { T,F,TF }
508  {
509    \tl_if_eq:NNTF \l__tag_mc_firstmarks_seq \l__tag_mc_botmarks_seq
510      { \prg_return_false: }
511      { \prg_return_true: }
512  }
```
(*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.

```
513 \prg_new_conditional:Npnn \__tag_check_if_mc_tmb_missing: { T,F,TF }
514  {
```

38

```
515   \bool_if:nTF
516     {
517       \str_if_eq_p:ee {\seq_item:Nn \l__tag_mc_firstmarks_seq {1}}{e-}
518       ||
519       \str_if_eq_p:ee {\seq_item:Nn \l__tag_mc_firstmarks_seq {1}}{b+}
520     }
521     { \prg_return_true: }
522     { \prg_return_false: }
523   }
```

(End of definition for __tag_check_if_mc_tmb_missing:TF.)

_tag_check_if_mc_tme_missing_p:
_tag_check_if_mc_tme_missing:*TF*

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

```
524   \prg_new_conditional:Npnn \__tag_check_if_mc_tme_missing: { T,F,TF }
525   {
526     \str_if_eq:eeTF {\seq_item:Nn \l__tag_mc_botmarks_seq {1}}{b+}
527     { \prg_return_true: }
528     { \prg_return_false: }
529   }
```

(End of definition for __tag_check_if_mc_tme_missing:TF.)

```
530  ⟨/package⟩
531  ⟨∗debug⟩
```

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

```
532  \msg_new:nnn { tag / debug } {mc-begin} { MC~begin~#1~with~options:~\tl_to_str:n{#2}~[\msg_li
533  \msg_new:nnn { tag / debug } {mc-end}    { MC~end~#1~[\msg_line_context:] }
534
535  \cs_new_protected:Npn \__tag_debug_mc_begin_insert:n #1
536    {
537      \int_compare:nNnT { \l__tag_loglevel_int } > {0}
538        {
539          \msg_note:nnnn { tag / debug } {mc-begin} {inserted} { #1 }
540        }
541    }
542  \cs_new_protected:Npn \__tag_debug_mc_begin_ignore:n #1
543    {
544      \int_compare:nNnT { \l__tag_loglevel_int } > {0}
545        {
546          \msg_note:nnnn { tag / debug } {mc-begin } {ignored} { #1 }
547        }
548    }
549  \cs_new_protected:Npn \__tag_debug_mc_end_insert:
550    {
551      \int_compare:nNnT { \l__tag_loglevel_int } > {0}
552        {
553          \msg_note:nnn { tag / debug } {mc-end} {inserted}
554        }
555    }
556  \cs_new_protected:Npn \__tag_debug_mc_end_ignore:
557    {
```

```
558    \int_compare:nNnT { \l__tag_loglevel_int } > {0}
559      {
560        \msg_note:nnn { tag / debug } {mc-end} {ignored}
561      }
562  }
```

And now something for the structures

```
563  \msg_new:nnn { tag / debug } {struct-begin}
564    {
565      Struct~\tag_get:n{struct_num}~begin~#1~with~options:~\tl_to_str:n{#2}~\\[\msg_line_contex
566    }
567  \msg_new:nnn { tag / debug } {struct-end}
568    {
569      Struct~end~#1~[\msg_line_context:]
570    }
571  \msg_new:nnn { tag / debug } {struct-end-wrong}
572    {
573      Struct~end~'#1'~doesn't~fit~start~'#2'~[\msg_line_context:]
574    }
575
576  \cs_new_protected:Npn \__tag_debug_struct_begin_insert:n #1
577  {
578    \int_compare:nNnT { \l__tag_loglevel_int } > {0}
579      {
580        \msg_note:nnnn { tag / debug } {struct-begin} {inserted} { #1 }
581        \seq_log:N \g__tag_struct_tag_stack_seq
582      }
583  }
584  \cs_new_protected:Npn \__tag_debug_struct_begin_ignore:n #1
585  {
586    \int_compare:nNnT { \l__tag_loglevel_int } > {0}
587      {
588        \msg_note:nnnn { tag / debug } {struct-begin } {ignored} { #1 }
589      }
590  }
591  \cs_new_protected:Npn \__tag_debug_struct_end_insert:
592  {
593    \int_compare:nNnT { \l__tag_loglevel_int } > {0}
594      {
595        \msg_note:nnn { tag / debug } {struct-end} {inserted}
596        \seq_log:N \g__tag_struct_tag_stack_seq
597      }
598  }
599  \cs_new_protected:Npn \__tag_debug_struct_end_ignore:
600  {
601    \int_compare:nNnT { \l__tag_loglevel_int } > {0}
602      {
603        \msg_note:nnn { tag / debug } {struct-end } {ignored}
604      }
605  }
606  \cs_new_protected:Npn \__tag_debug_struct_end_check:n #1
607  {
608    \int_compare:nNnT { \l__tag_loglevel_int } > {0}
609      {
610        \seq_get:NNT \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
```

```
611              {
612                \str_if_eq:eeF
613                  {#1}
614                  {\exp_last_unbraced:No \use_i:nn { \l__tag_tmpa_tl }}
615                  {
616                    \msg_warning:nnee { tag/debug }{ struct-end-wrong }
617                      {#1}
618                      {\exp_last_unbraced:No \use_i:nn { \l__tag_tmpa_tl }}
619                  }
620              }
621          }
622  }
```

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

```
623  \msg_new:nnn { tag / debug } {tag-suspend}
624    {
625      \int_if_zero:nTF
626        {#1}
627        {Tagging~suspended}
628        {Tagging~(not)~suspended~(already~inactive)}\\
629      level:~#1~==>~\int_eval:n{#1+1}\tl_if_empty:nF{#2}{,~label:~#2}~[\msg_line_context:]
630    }
631  \msg_new:nnn { tag / debug } {tag-resume}
632    {
633      \int_if_zero:nTF
634        {#1}
635        {Tagging~resumed}
636        {Tagging~(not)~resumed}\\
637      level:~\int_eval:n{#1+1}~==>~#1\tl_if_empty:nF{#2}{,~label:~#2}~[\msg_line_context:]
638    }

639  ⟨/debug⟩
```

7.6   Benchmarks

It doesn't make much sense to do benchmarks in debug mode or in combination with a
log-level as this would slow down the compilation. So we add simple commands that can
be activated if l3benchmark has been loaded. TODO: is a warning needed?

```
640  ⟨*package⟩
641  \cs_new_protected:Npn \__tag_check_benchmark_tic:{}
642  \cs_new_protected:Npn \__tag_check_benchmark_toc:{}
643  \cs_new_protected:Npn \tag_check_benchmark_on:
644    {
645      \cs_if_exist:NT \benchmark_tic:
646        {
647          \cs_set_eq:NN \__tag_check_benchmark_tic: \benchmark_tic:
648          \cs_set_eq:NN \__tag_check_benchmark_toc: \benchmark_toc:
649        }
650    }
651  ⟨/package⟩
```

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

Ulrike Fischer
Version 0.99z, released 2026-03-19

Part III

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 adds 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 \tagmcifinTF{⟨*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 documented in the `tagpdf-struct` module.

4 Debugging

\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 = ⟨*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

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|use

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

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 = ⟨*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

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

44

5.2 Tagging of paragraphs

This makes use of the paragraph hooks in 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.

para/tagging (setup-key)	para/tagging = true|false
paratagging-show (deprecated)	debug/show=para
paratagging (deprecated)	debug/show=paraOff

The `para/tagging` key can be used in `\tagpdfsetup` and enable/disables tagging of paragraphics. `debug/show=para` puts small colored 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`
`\tagpdfparaOff` These commands allow to enable/disable para tagging too and are a bit faster then `\tagpdfsetup`. But I'm not sure if the names are good.

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

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

5.3 Header and footer

Header and footer are automatically tagged as artifact: They are surrounded by an artifact-mc and inside tagging is stopped. If some real content is in the header and footer, tagging must be restarted there explicitly. The behaviour can be changed with the following key. The key accepts the values `true` (the default), `false` which disables the header tagging code. This can be useful if the page style is empty (it then avoids empty mc-chunks) or if the head and foot should be tagged in some special way. The last value, `pagination`, is like `true` but additionally adds an artifact structure with an pagination attribute.

page/exclude-header-footer (setup-key)	page/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:nnn
\UseTaggingSocket
```

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

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 is expects a socket starting with `tagsupport/` but the socket name is specified without this prefix, i.e.,

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

.

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.

Usually, these sockets have (beside the default plug defined for every socket) one additional plug defined and directly assigned. This plug is used when tagging is active. There may be more plugs, e.g., tagging with special debugging or special behaviour depending on the class or PDF version etc., but right now it is usually just on or off.

When tagging is suspended they all have the same predefined behaviour: The sockets with zero arguments do nothing. The sockets with one argument gobble their argument. The sockets with two arguments will drop their first argument and pass the second unchanged.

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`.

For special cases like in some `\halign` contexts we need a fully expandable version of the commend. For these cases, `\UseExpandableTaggingSocket` can be used. To allow being expandable, it does not output any debugging information if `\DebugSocketsOn` is in effect and therefore should be avoided whenever possible.

The L3 programming layer versions `\tag_socket_use_expandable:n`, `\tag_-socket_use:n`, and `\tag_socket_use:nn`, `\tag_socket_use:nnn` 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} {2026-03-19} {0.99z}
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 43.*)

\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 43.*)

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 {  }
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 43.*)

\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 43.*)

10 Commands for the structure

\tagstructbegin
\tagstructend   These are structure related user commands. There are direct wrapper around the expl3
\tagstructuse   variants.

```
45  ⟨*base⟩
46  \NewDocumentCommand \tagstructbegin { m }
47    {
48      \tag_struct_begin:n {#1}
49    }
50
51  \NewDocumentCommand \tagstructend {  }
52    {
53      \tag_struct_end:
54    }
55
56  \NewDocumentCommand \tagstructuse { m }
57    {
58      \tag_struct_use:n {#1}
59    }
60  ⟨/base⟩
```
(*End of definition for* \tagstructbegin, \tagstructend, *and* \tagstructuse. *These functions are documented on page 43.*)

48

11 Socket support

Until we can be sure that the kernel defines the commands we provide them before redefining them: The expandable version will only work correctly after the 2024-11-01 release.

```
61 ⟨*base⟩
62 \providecommand\tag_socket_use:n[1]{}
63 \providecommand\tag_socket_use:nn[2]{}
64 \providecommand\tag_socket_use:nnn[3]{#3}
65 \providecommand\tag_socket_use_expandable:n[1]{}
66 \providecommand\socket_use_expandable:nw [1] {
67   \use:c { __socket_#1_plug_ \str_use:c { l__socket_#1_plug_str } :w }
68 }
69 \providecommand\UseTaggingSocket[1]{}
70 \providecommand\UseExpandableTaggingSocket[1]{}
71 ⟨/base⟩
```

\tag_socket_use:n
\tag_socket_use:nn
\tag_socket_use:nnn
\UseTaggingSocket
\tag_socket_use_expandable:n
\UseExpandableTaggingSocket

```
72 ⟨*package⟩
73 \cs_set_protected:Npn \tag_socket_use:n #1
74   {
75     \bool_if:NT \l__tag_active_socket_bool
76       { \socket_use:n {tagsupport/#1} }
77   }
78 \cs_set_protected:Npn \tag_socket_use:nn #1#2
79   {
80     \bool_if:NT \l__tag_active_socket_bool
81       { \socket_use:nn {tagsupport/#1} {#2} }
82   }
83 \cs_set_protected:Npn \tag_socket_use:nnn #1#2#3
84   {
85     \bool_if:NTF \l__tag_active_socket_bool
86       { \socket_use:nnn {tagsupport/#1} {#2} {#3} }
87       { #3 }
88   }
89 \cs_set:Npn \tag_socket_use_expandable:n #1
90   {
91     \bool_if:NT \l__tag_active_socket_bool
92       { \socket_use_expandable:n {tagsupport/#1} }
93   }
94 \cs_set_protected:Npn \UseTaggingSocket #1
95   {
96     \bool_if:NTF \l__tag_active_socket_bool
97       { \socket_use:nw {tagsupport/#1} }
98       {
99         \int_case:nnF
100          { \int_use:c { c__socket_tagsupport/#1_args_int } }
101          {
102            0 \prg_do_nothing:
103            1 \use_none:n
104            2 \use_ii:nn
```

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

```
105                }
106                \ERRORusetaggingsocket
107        }
108    }
109 \cs_set:Npn \UseExpandableTaggingSocket #1
110    {
111      \bool_if:NTF \l__tag_active_socket_bool
112        { \socket_use_expandable:nw {tagsupport/#1} }
113        {
114          \int_case:nnF
115              { \int_use:c { c__socket_tagsupport/#1_args_int } }
116              {
117                0 \prg_do_nothing:
118                1 \use_none:n
119                2 \use_ii:nn
```

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

```
120                }
121                \ERRORusetaggingsocket
122        }
123    }
124 ⟨/package⟩
```

(*End of definition for* \tag_socket_use:n *and others. These functions are documented on page 46.*)

12 Debugging

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

```
125 ⟨*package⟩
126 \NewDocumentCommand\ShowTagging { m }
127    {
128      \keys_set:nn { __tag / show }{ #1}
129
130    }
```

(*End of definition for* \ShowTagging*. This function is documented on page 44.*)

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.

```
131 \keys_define:nn { __tag / show }
132    {
133      mc-data .code:n =
134        {
135          \bool_if:NT \g__tag_mode_lua_bool
136            {
137              \lua_now:e{ltx.__tag.trace.show_all_mc_data(#1,\__tag_get_mc_abs_cnt:,0)}
138            }
139        }
```

```
140        ,mc-data .default:n = 1
141    }
142
```
(End of definition for `mc-data` *(show-key). This function is documented on page* *44.)*

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

```
143 \keys_define:nn { __tag / show }
144   { mc-current .code:n =
145      {
146         \bool_if:NTF \g__tag_mode_lua_bool
147           {
148              \int_compare:nNnTF
149                { -2147483647 }
150                 =
151                {
152                   \lua_now:e
153                     {
154                        tex.print
155                        (\int_use:N\c_document_cctab,
156                         tex.getattribute
157                         (luatexbase.attributes.g__tag_mc_cnt_attr))
158                     }
159                }
160                {
161                   \lua_now:e
162                     {
163                       ltx.__tag.trace.log
164                        (
165                         "mc-current:~no~MC~open,~current~abscnt
166                          =\__tag_get_mc_abs_cnt:"
167                         ,0
168                        )
169                       texio.write_nl("")
170                     }
171                }
172                {
173                   \lua_now:e
174                     {
175                       ltx.__tag.trace.log
176                        (
177                         "mc-current:~abscnt=\__tag_get_mc_abs_cnt:=="
178                          ..
179                          tex.getattribute(luatexbase.attributes.g__tag_mc_cnt_attr)
180                          ..
181                          "~=>tag="
182                          ..
183                          tostring
184                            (ltx.__tag.func.get_tag_from
185                              (tex.getattribute
186                                (luatexbase.attributes.g__tag_mc_type_attr)))
187                          ..
188                          "="
189                          ..
190                          tex.getattribute
```

51

```
191                          (luatexbase.attributes.g__tag_mc_type_attr)
192                        ,0
193                      )
194                    texio.write_nl("")
195                  }
196                }
197              }
198              {
199                \msg_note:nn{ tag }{ mc-current }
200              }
201          }
202    }
```

(*End of definition for* `mc-current` (*show-key*). *This function is documented on page 44.*)

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).

```
203 \keys_define:nn { __tag / show }
204   {
205     mc-marks .choice: ,
206     mc-marks / show .code:n =
207       {
208         \__tag_mc_get_marks:
209         \__tag_check_if_mc_in_galley:TF
210         {
211          \iow_term:n {Marks~from~this~page:~}
212         }
213         {
214            \iow_term:n {Marks~from~a~previous~page:~}
215         }
216         \seq_show:N \l__tag_mc_firstmarks_seq
217         \seq_show:N \l__tag_mc_botmarks_seq
218         \__tag_check_if_mc_tmb_missing:T
219         {
220            \iow_term:n {BDC~missing~on~this~page!}
221         }
222         \__tag_check_if_mc_tme_missing:T
223         {
224            \iow_term:n {EMC~missing~on~this~page!}
225         }
226       },
227     mc-marks / use .code:n =
228       {
229         \__tag_mc_get_marks:
230         \__tag_check_if_mc_in_galley:TF
231         { Marks~from~this~page:~}
232         { Marks~from~a~previous~page:~}
233         \seq_use:Nn \l__tag_mc_firstmarks_seq {,~}\quad
234         \seq_use:Nn \l__tag_mc_botmarks_seq {,~}\quad
235         \__tag_check_if_mc_tmb_missing:T
236         {
237            BDC~missing~
238         }
239         \__tag_check_if_mc_tme_missing:T
240         {
```

```
241              EMC~missing
242           }
243        },
244      mc-marks .default:n = show
245    }
```
(*End of definition for* `mc-marks` *(show-key). This function is documented on page 44.*)

```
246 \keys_define:nn { __tag / show }
247   {
248      struct-stack .choice:
249      ,struct-stack / log .code:n = \seq_log:N \g__tag_struct_tag_stack_seq
250      ,struct-stack / show .code:n = \seq_show:N \g__tag_struct_tag_stack_seq
251      ,struct-stack .default:n = show
252    }
253 ⟨/package⟩
```
(*End of definition for* `struct-stack` *(show-key). This function is documented on page 44.*)

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.

```
254 ⟨∗debug⟩
255 \keys_define:nn { __tag / show }
256   {
257      ,debug/structures .code:n =
258        {
259          \int_step_inline:nnn{#1}{\c@g__tag_struct_abs_int}
260            {
261              \msg_term:nneeee
262                   { tag/debug } { show-struct }
263                   { ##1 }
264                   {
265                      \prop_map_function:cN
266                        {g__tag_struct_debug_##1_prop}
267                        \msg_show_item_unbraced:nn
268                   }
269                   { } { }
270              \msg_term:nneeee
271                   { tag/debug } { show-kids }
272                   {  ##1 }
273                   {
274                      \seq_map_function:cN
275                        {g__tag_struct_debug_kids_##1_seq}
276                        \msg_show_item_unbraced:n
277                   }
278                   { } { }
279            }
280        }
281      ,debug/structures .default:n = 1
282    }
283 ⟨/debug⟩
```
(*End of definition for* `debug/structures` *(show-key). This function is documented on page 44.*)

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.

284 ⟨*package⟩

13.1 Document structure

```
285 \tl_new:N\g__tag_root_default_tl
286 \tl_gset:Nn\g__tag_root_default_tl {Document}
287
288 \hook_gput_code:nnn{begindocument}{tagpdf}{\tagstructbegin{tag=\g__tag_root_default_tl}}
289 \hook_gput_code:nnn{tagpdf/finish/before}{tagpdf}{\tagstructend}
290
291 \keys_define:nn { __tag / setup}
292  {
293    activate/socket .bool_set:N  = \l__tag_active_socket_bool,
294    activate  .code:n =
295     {
296       \keys_set:nn { __tag / setup }
297         { activate/mc,activate/tree,activate/struct,activate/socket }
298       \tl_gset:Nn\g__tag_root_default_tl {#1}
299     },
300    activate .default:n = Document
301  }
302
```

(*End of definition for* \g__tag_root_default_tl, activate (setup-key), *and* activate/socket (setup-key). *These functions are documented on page 43.*)

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.

```
303 \AddToHook{begindocument/before}
304  {
305    \bool_lazy_and:nnT
306      { \g__tag_active_struct_dest_bool }
307      { \g__tag_active_struct_bool }
308      {
309        \tl_set:Nn \l_pdf_current_structure_destination_tl
310          { {__tag/struct}{\g__tag_struct_stack_current_tl }}
311        \pdf_activate_indexed_structure_destination:
312      }
313  }
```

54

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

```
314 \bool_if:NT \g__tag_mode_lua_bool
315   {
316     \NewDocumentCommand\pdffakespace { }
317       {
318         \__tag_fakespace:
319       }
320   }
321 \providecommand\pdffakespace{}
```

(*End of definition for* `\pdffakespace`*. This function is documented on page 44.*)

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
\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

At first some variables.

```
322 ⟨/package⟩
323 ⟨base⟩\bool_new:N \l__tag_para_flattened_bool
324 ⟨base⟩\bool_new:N \l__tag_para_bool
325 ⟨*package⟩
326 \int_new:N  \g__tag_para_begin_int
327 \int_new:N  \g__tag_para_end_int
328 \int_new:N  \g__tag_para_main_begin_int
329 \int_new:N  \g__tag_para_main_end_int
```

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

```
330 \tl_new:N   \g__tag_para_main_struct_tl
331 \tl_gset:Nn  \g__tag_para_main_struct_tl {1}
332 \tl_new:N   \l__tag_para_tag_default_tl
333 \tl_new:N   \l__tag_para_tag_tl
334 \tl_set:Nn  \l__tag_para_tag_tl { \l__tag_para_tag_default_tl }
335 \tl_new:N   \l__tag_para_main_tag_tl
336 \IfFormatAtLeastTF{2025-11-01}
337   {
338     \tl_set:Nn  \l__tag_para_tag_default_tl { \UseStructureName {para/textblock} }
339     \tl_set:Nn  \l__tag_para_main_tag_tl    { \UseStructureName {para/semantic} }
340   }
341   {
342     \tl_set:Nn  \l__tag_para_tag_default_tl { text }
343     \tl_set:Nn  \l__tag_para_main_tag_tl    {text-unit}
344   }
345
```

this is perhaps already defined by the block code

```
346 \tl_if_exist:NF \l__tag_para_attr_class_tl
347   {\tl_new:N \l__tag_para_attr_class_tl }
348 \tl_new:N \l__tag_para_main_attr_class_tl
```

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

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

```
\_tag_gincr_para_main_begin_int:
\_tag_gincr_para_main_end_int:
\__tag_gincr_para_begin_int:
\__tag_gincr_para_end_int:
```

```
349 \cs_new_protected:Npn \__tag_gincr_para_main_begin_int:
350   {
351     \int_gincr:N \g__tag_para_main_begin_int
352   }
353 \cs_new_protected:Npn \__tag_gincr_para_begin_int:
354   {
355     \int_gincr:N \g__tag_para_begin_int
356   }
357 \cs_new_protected:Npn \__tag_gincr_para_main_end_int:
358   {
359     \int_gincr:N \g__tag_para_main_end_int
360   }
361 \cs_new_protected:Npn \__tag_gincr_para_end_int:
362   {
363     \int_gincr:N \g__tag_para_end_int
364   }
```

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

```
\__tag_start_para_ints:
\__tag_stop_para_ints:
```

```
365 \cs_new_protected:Npn \__tag_start_para_ints:
366   {
367     \cs_set_protected:Npn \__tag_gincr_para_main_begin_int:
368       {
369         \int_gincr:N \g__tag_para_main_begin_int
370       }
371     \cs_set_protected:Npn \__tag_gincr_para_begin_int:
372       {
373         \int_gincr:N \g__tag_para_begin_int
374       }
375     \cs_set_protected:Npn \__tag_gincr_para_main_end_int:
376       {
377         \int_gincr:N \g__tag_para_main_end_int
378       }
379     \cs_set_protected:Npn \__tag_gincr_para_end_int:
380       {
381         \int_gincr:N \g__tag_para_end_int
382       }
383   }
384 \cs_new_protected:Npn \__tag_stop_para_ints:
385   {
386     \cs_set_eq:NN \__tag_gincr_para_main_begin_int:\prg_do_nothing:
387     \cs_set_eq:NN \__tag_gincr_para_begin_int: \prg_do_nothing:
388     \cs_set_eq:NN \__tag_gincr_para_main_end_int: \prg_do_nothing:
389     \cs_set_eq:NN \__tag_gincr_para_end_int: \prg_do_nothing:
390   }
```

(*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:
```

```
391 \cs_new:Npn \__tag_para_main_store_struct:
```

```
392    {
393      \tl_gset:Ne \g__tag_para_main_struct_tl {\int_use:N \c@g__tag_struct_abs_int }
394    }
```
(*End of definition for* __tag_para_main_store_struct:.)

temporary adaption for the block module:

```
395 \AddToHook{package/latex-lab-testphase-block/after}
396  {
397    \tl_if_exist:NT \l_tag_para_attr_class_tl
398      {
399        \tl_set:Nn \l__tag_para_attr_class_tl { \l_tag_para_attr_class_tl }
400      }
401 }
```

These keys enable/disable locally paratagging. Paragraphs are typically tagged with two

para/tagging (setup-key)
para/tag (setup-key)
para/maintag (setup-key)
para/tagging (tool-key)
para/tag (tool-key)
para/maintag (tool-key)
para/flattened (tool-key)
unittag (deprecated)
para-flattened (deprecated)
paratagging (deprecated)
paratagging-show (deprecated)
paratag (deprecated)

These keys enable/disable locally paratagging. Paragraphs are typically tagged with two structure: A main structure around the whole paragraph, and inner structures around the various chunks. Debugging can be activated locally with debug/show=para, this can affect the typesetting as the small numbers are boxes and they have a (small) height. Debugging can be deactivated with debug/show=paraOff The para/tag key sets the tag used by the inner structure, para/maintag the tag of the outer structure, both can also be changed with \tag_tool:n

```
402 \keys_define:nn { __tag / setup }
403  {
404    para/tagging      .bool_set:N = \l__tag_para_bool,
405    debug/show/para   .code:n = {\bool_set_true:N \l__tag_para_show_bool},
406    debug/show/paraOff .code:n = {\bool_set_false:N \l__tag_para_show_bool},
407    para/tag          .tl_set:N  = \l__tag_para_tag_tl,
408    para/maintag      .tl_set:N  = \l__tag_para_main_tag_tl,
409    para/flattened    .bool_set:N = \l__tag_para_flattened_bool
410  }
411 \keys_define:nn { tag / tool}
412  {
413    para/tagging   .bool_set:N = \l__tag_para_bool,
414    para/tag       .tl_set:N = \l__tag_para_tag_tl,
415    para/maintag   .tl_set:N = \l__tag_para_main_tag_tl,
416    para/flattened .bool_set:N = \l__tag_para_flattened_bool
417  }
```

the deprecated names

```
418 \keys_define:nn { __tag / setup }
419  {
420    paratagging      .bool_set:N = \l__tag_para_bool,
421    paratagging-show .bool_set:N = \l__tag_para_show_bool,
422    paratag          .tl_set:N  = \l__tag_para_tag_tl
423  }
424 \keys_define:nn { tag / tool}
425  {
426    para    .bool_set:N = \l__tag_para_bool,
427    paratag .tl_set:N = \l__tag_para_tag_tl,
428    unittag .tl_set:N = \l__tag_para_main_tag_tl,
429    para-flattened .bool_set:N = \l__tag_para_flattened_bool
430  }
```
(*End of definition for* para/tagging (setup-key) *and others. These functions are documented on page 45.*)

Helper command for debugging:

```
431 \cs_new_protected:Npn \__tag_check_para_begin_show:nn #1 #2
432 %#1 color, #2 prefix
433   {
434     \bool_if:NT \l__tag_para_show_bool
435       {
436         \tag_mc_begin:n{artifact}
437         \llap{\color_select:n{#1}\tiny#2\int_use:N\g__tag_para_begin_int\ }
438         \tag_mc_end:
439       }
440   }
441
442 \cs_new_protected:Npn \__tag_check_para_end_show:nn #1 #2
443 %#1  color, #2 prefix
444   {
445     \bool_if:NT \l__tag_para_show_bool
446       {
447         \tag_mc_begin:n{artifact}
448         \rlap{\color_select:n{#1}\tiny\ #2\int_use:N\g__tag_para_end_int}
449         \tag_mc_end:
450       }
451   }
```

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. This code moves into lttagging (2025/11), so we add a test for the transition.

```
452 \socket_if_exist:nF { tagsupport/para/begin }
453   {
454     \NewTaggingSocket  {para/begin}{0}
455     \NewTaggingSocket  {para/end}{0}
456
457     \NewTaggingSocketPlug{para/begin}{plain}
458      {
459        \bool_if:NT \l__tag_para_bool
460          {
461            \bool_if:NF \l__tag_para_flattened_bool
462              {
463                \__tag_gincr_para_main_begin_int:
464                \tag_struct_begin:n
465                  {
466                    tag=\l__tag_para_main_tag_tl,
467                  }
468                \__tag_para_main_store_struct:
469              }
470            \__tag_gincr_para_begin_int:
471            \tag_struct_begin:n {tag=\l__tag_para_tag_tl}
472            \__tag_check_para_begin_show:nn {green}{}
473            \tag_mc_begin:n {}
474          }
475      }
476    \NewTaggingSocketPlug{para/begin}{block}
477      {
```

```
478        \bool_if:NT \l__tag_para_bool
479          {
480            \legacy_if:nF { @inlabel }
481              {
482                \__tag_check_typeout_v:n
483                  {==>~ @endpe = \legacy_if:nTF { @endpe }{true}{false} \on@line }
484                \legacy_if:nF { @endpe }
485                 {
486                   \bool_if:NF \l__tag_para_flattened_bool
487                     {
488                       \__tag_gincr_para_main_begin_int:
489                       \tag_struct_begin:n
490                        {
491                          tag=\l__tag_para_main_tag_tl,
492                          attribute-class=\l__tag_para_main_attr_class_tl,
493                        }
494                       \__tag_para_main_store_struct:
495                     }
496                 }
497                \__tag_gincr_para_begin_int:
498                \__tag_check_typeout_v:n {==>~increment~ P \on@line }
499                \tag_struct_begin:n
500                  {
501                    tag=\l__tag_para_tag_tl
502                    ,attribute-class=\l__tag_para_attr_class_tl
503                  }
504                \__tag_check_para_begin_show:nn {green}{\PARALABEL}
505                \tag_mc_begin:n {}
506              }
507          }
508      }
```
there was no real difference between the original and in the block variant, only a debug message. We therefore define only a plain variant.
```
509      \NewTaggingSocketPlug {para/end}{plain}
510        {
511          \bool_if:NT \l__tag_para_bool
512            {
513              \__tag_gincr_para_end_int:
514              \__tag_check_typeout_v:n {==>~increment~ /P \on@line }
515              \tag_mc_end:
516              \__tag_check_para_end_show:nn {red}{}
517              \tag_struct_end:
518              \bool_if:NF \l__tag_para_flattened_bool
519                {
520                  \__tag_gincr_para_main_end_int:
521                  \tag_struct_end:
522                }
523            }
524        }
525    }
```
By default we assign the plain plug:
```
526 \AssignTaggingSocketPlug { para/begin}{plain}
527 \AssignTaggingSocketPlug { para/end}{plain}
```

59

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

```
528 \AddToHook{para/begin}{ \tag_socket_use:n { para/begin } }
529 \AddToHook{para/end}  { \tag_socket_use:n { 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.

```
530 \AddToHook{package/latex-lab-testphase-block/after}
531 {
532   \RemoveFromHook{para/begin}[tagpdf]
533   \RemoveFromHook{para/end}[latex-lab-testphase-block]
534   \AddToHook{para/begin}[tagpdf]
535     {
536       \socket_use:n { tagsupport/para/begin }
537     }
538   \AddToHook{para/end}[tagpdf]
539     {
540       \socket_use:n { tagsupport/para/end }
541     }
542   \socket_assign_plug:nn { tagsupport/para/begin}{block}
543 }
544
```

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.

```
545 \AddToHook{enddocument/info}
546   {
547     \tag_if_active:F
548       {
549         \msg_redirect_name:nnn { tag } { para-hook-count-wrong } { warning }
550       }
551     \int_compare:nNnF {\g__tag_para_main_begin_int}={\g__tag_para_main_end_int}
552         {
553           \msg_error:nneee
554             {tag}
555             {para-hook-count-wrong}
556             {\int_use:N\g__tag_para_main_begin_int}
557             {\int_use:N\g__tag_para_main_end_int}
558             {text-unit}
559         }
560     \int_compare:nNnF {\g__tag_para_begin_int}={\g__tag_para_end_int}
561         {
562           \msg_error:nneee
563             {tag}
564             {para-hook-count-wrong}
565             {\int_use:N\g__tag_para_begin_int}
566             {\int_use:N\g__tag_para_end_int}
567             {text}
568         }
569   }
```

</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/tagging=false}

```
570 ⟨base⟩\newcommand\tagpdfparaOn {}
571 ⟨base⟩\newcommand\tagpdfparaOff{}
572 ⟨*package⟩
573 \renewcommand\tagpdfparaOn {\bool_set_true:N \l__tag_para_bool}
574 \renewcommand\tagpdfparaOff{\bool_set_false:N \l__tag_para_bool}
```

(*End of definition for* \tagpdfparaOn *and* \tagpdfparaOff*. These functions are documented on page 45.*)

\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}%
```

```
575 \NewDocumentCommand\tagpdfsuppressmarks{m}
576   {{\use:c{__tag_mc_disable_marks:} #1}}
```

(*End of definition for* \tagpdfsuppressmarks*. This function is documented on page 45.*)

13.5 Language support

With the following key the lang variable is set. All structures in the current group will then set this lang variable.

test/lang (setup-key)

```
577 \keys_define:nn { __tag / setup }
578   {
579     text / lang .tl_set:N = \l__tag_struct_lang_tl
580   }
```

(*End of definition for* test/lang (setup-key)*. This function is documented on page* **??***.*)

13.6 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.

```
581 \cs_new_protected:Npn\__tag_hook_kernel_before_head:{}
582 \cs_new_protected:Npn\__tag_hook_kernel_after_head:{}
583 \cs_new_protected:Npn\__tag_hook_kernel_before_foot:{}
584 \cs_new_protected:Npn\__tag_hook_kernel_after_foot:{}
```

We use the page sockets.

```
585  \NewTaggingSocketPlug{build/page/header}{tagpdf}
586      {
587        \__tag_hook_kernel_before_head:
588        #2
589        \__tag_hook_kernel_after_head:
590      }
591
592  \AssignTaggingSocketPlug{build/page/header}{tagpdf}
593  \NewTaggingSocketPlug{build/page/footer}{tagpdf}
594      {
595        \__tag_hook_kernel_before_foot:
596        #2
597        \__tag_hook_kernel_after_foot:
598      }
599  \AssignTaggingSocketPlug{build/page/footer}{tagpdf}
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_suspend:n{headfoot}
614  }
615  \cs_new_protected:Npn \__tag_exclude_headfoot_end:
616  {
617      \tag_resume: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_suspend:n{headfoot}
642  }
643
644 \cs_new_protected:Npn \__tag_exclude_struct_headfoot_end:
645  {
646    \tag_resume: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     page/exclude-header-footer .choice:,
660     page/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     page/exclude-header-footer / pagination .code:n =
668       {
669         \cs_set:Nn \__tag_hook_kernel_before_head: { \__tag_exclude_struct_headfoot_begin:n {p
670         \cs_set:Nn \__tag_hook_kernel_before_foot: { \__tag_exclude_struct_headfoot_begin:n {p
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     page/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     page/exclude-header-footer .default:n = true,
682     page/exclude-header-footer .initial:n = true,
```

deprecated name

```
683     exclude-header-footer .meta:n = { page/exclude-header-footer = {#1} }
684   }
```

A special, experimental tagged version, which only works with fancyhdr or similar that uses parbox. The sockets aren't in lttagging yet, so until 2026-06-01 we have to declare the plugs at begin document:

```
685 \AtBeginDocument
686   {
687     \NewTaggingSocketPlug{parbox/before}{tag/footer}
688       {
689         \tag_struct_begin:n{tag=Span}
690         \tag_mc_begin:n{}
691       }
692
693     \NewTaggingSocketPlug{parbox/after}{tag/footer}
694       {
695         \tag_mc_end:
696         \tag_struct_end:
697       }
698   }
699 \cs_new_protected:Npn \__tag_headfoot_tagged_begin:n #1
700   {
701     \AssignTaggingSocketPlug{parbox/before}{tag/footer}
702     \AssignTaggingSocketPlug{parbox/after}{tag/footer}
703     \bool_set_false:N  \l__tag_para_bool
704     \bool_if:NTF \g__tag_mode_lua_bool
705       {
706         \tag_mc_end_push:
707       }
708       {
709         \bool_gset_eq:NN   \g__tag_saved_in_mc_bool \g__tag_in_mc_bool
710         \bool_gset_false:N \g__tag_in_mc_bool
711       }
712     \tag_struct_begin:n{tag=Artifact,attribute-class=__tag/attr/#1,parent=\tag_get:n{current_
713   }
714
715 \cs_new_protected:Npn \__tag_headfoot_tagged_end:
716   {
717     \tag_struct_end:
718     \bool_if:NTF \g__tag_mode_lua_bool
719       {
720         \tag_mc_begin_pop:n{}
721       }
722       {
723         \bool_gset_eq:NN \g__tag_in_mc_bool\g__tag_saved_in_mc_bool
724       }
725   }
726 \keys_define:nn { __tag / setup }
727   {
728     page/tag-header-footer .choice:,
729     page/tag-header-footer/artifact .code:n =
730       {
731         \cs_set:Nn \__tag_hook_kernel_before_head: { \__tag_headfoot_tagged_begin:n {pagination
732         \cs_set:Nn \__tag_hook_kernel_before_foot: { \__tag_headfoot_tagged_begin:n {pagination
```

```
733        \cs_set_eq:NN \__tag_hook_kernel_after_head:  \__tag_headfoot_tagged_end:
734        \cs_set_eq:NN \__tag_hook_kernel_after_foot:  \__tag_headfoot_tagged_end:
735      },
736    page/tag-header-footer .default:n = artifact,
737  }
```

13.7 Links

We need to close and reopen mc-chunks around links. We handle URI, GoTo (internal) links, GoToR, Launch and Named links. Links should have an alternative text in the Contents key; this is added for normal links by the generic hyperref driver. With luatex we make use of the lualinksplit package to get OBJR of all annotations into the Link structure, so the hook code should not contain the command to insert the OBJR into the structure.

At first we provide the link name that will be in the next LaTeX release (06/2026)

```
738 \AddToHookNext{class/before}
739 {
740   \cs_if_exist:NF \l__tag_name_link_tl
741     {
742       \tl_new:N  \l__tag_name_link_tl
743       \tl_set:Nn \l__tag_name_link_tl{Link}
744     }
745 }
```

Tagging sockets for links

```
746 \socket_if_exist:nF {tagsupport/link/before}
747 {
748   \NewTaggingSocket{link/before}{1}
749   \NewTaggingSocket{link/after}{1}
750 }
751 \NewTaggingSocketPlug{link/before}{kernel}
752 {
753   \mode_leave_vertical:
754   \tag_mc_end_push:
755   \tag_struct_begin:n { tag=\UseStructureName{link} }
756   \tag_mc_begin:n {}
757   #1
758 }
759 \AssignTaggingSocketPlug{link/before}{kernel}
760
761 \NewTaggingSocketPlug{link/after}{kernel}
762 {
763   #1
764   \tag_mc_end:
765   \tag_struct_end:
766   \tag_mc_begin_pop:n{}
767 }
768 \AssignTaggingSocketPlug{link/after}{kernel}
769
770
771 \bool_lazy_and:nnTF
772 { \sys_if_engine_luatex_p: }
773 {
```

```
774    \tl_if_empty_p:e
775     {
776       \lua_now:e
777        { if~ luatexbase.in_callback('pre_shipout_filter','linksplit')~
778          then~else~tex.print('1')~end
779        }
780     }
781  }
782  {
783     \hook_gput_code:nnn
784       {pdfannot/link/URI/before}
785       {tagpdf}
786       {
787         \UseTaggingSocket{link/before}{}
788       }
789
790     \hook_gput_code:nnn
791       {pdfannot/link/URI/after}
792       {tagpdf}
793       {
794         \UseTaggingSocket{link/after}{}
795       }
796
797     \hook_gput_code:nnn
798       {pdfannot/link/GoTo/before}
799       {tagpdf}
800       {
801         \UseTaggingSocket{link/before}{}
802       }
803
804     \hook_gput_code:nnn
805       {pdfannot/link/GoTo/after}
806       {tagpdf}
807       {
808         \UseTaggingSocket{link/after}{}
809       }
810
811     \hook_gput_code:nnn
812       {pdfannot/link/GoToR/before}
813       {tagpdf}
814       {
815         \UseTaggingSocket{link/before}
816       }
817
818     \hook_gput_code:nnn
819       {pdfannot/link/GoToR/after}
820       {tagpdf}
821       {
822         \UseTaggingSocket{link/after}{}
823       }
824     \hook_gput_code:nnn
825       {pdfannot/link/Launch/before}
826       {tagpdf}
827       {
```

```
828        \UseTaggingSocket{link/before}{}
829      }

830

831    \hook_gput_code:nnn
832      {pdfannot/link/Launch/after}
833      {tagpdf}
834      {
835        \UseTaggingSocket{link/after}{}
836      }
837    \hook_gput_code:nnn
838      {pdfannot/link/Named/before}
839      {tagpdf}
840      {
841        \UseTaggingSocket{link/before}{}
842      }

843

844    \hook_gput_code:nnn
845      {pdfannot/link/Named/after}
846      {tagpdf}
847      {
848        \UseTaggingSocket{link/after}{}
849      }
850  }
851  {
852    \hook_gput_code:nnn
853      {pdfannot/link/URI/before}
854      {tagpdf}
855      {
856        \UseTaggingSocket{link/before}
857          {
858           \pdfannot_dict_put:nne
859             { link/URI }
860             { StructParent }
861             { \tag_struct_parent_int: }
862          }
863      }

864

865    \hook_gput_code:nnn
866      {pdfannot/link/URI/after}
867      {tagpdf}
868      {
869        \UseTaggingSocket{link/after}
870          {
871           \tag_struct_insert_annot:ee
872             {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
873          }
874      }

875

876    \hook_gput_code:nnn
877      {pdfannot/link/GoTo/before}
878      {tagpdf}
879      {
880        \UseTaggingSocket{link/before}
881          {
```

```
882        \pdfannot_dict_put:nne
883          { link/GoTo }
884          { StructParent }
885          { \tag_struct_parent_int: }
886      }
887    }
888
889  \hook_gput_code:nnn
890    {pdfannot/link/GoTo/after}
891    {tagpdf}
892    {
893      \UseTaggingSocket{link/after}
894      {
895       \tag_struct_insert_annot:ee
896        {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
897      }
898    }
899
900  \hook_gput_code:nnn
901    {pdfannot/link/GoToR/before}
902    {tagpdf}
903    {
904      \UseTaggingSocket{link/before}
905      {
906       \pdfannot_dict_put:nne
907          { link/GoToR }
908          { StructParent }
909          { \tag_struct_parent_int: }
910      }
911    }
912
913  \hook_gput_code:nnn
914    {pdfannot/link/GoToR/after}
915    {tagpdf}
916    {
917      \UseTaggingSocket{link/after}
918      {
919       \tag_struct_insert_annot:ee
920        {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
921      }
922    }
923
924  \hook_gput_code:nnn
925    {pdfannot/link/Named/before}
926    {tagpdf}
927    {
928      \UseTaggingSocket{link/before}
929      {
930       \pdfannot_dict_put:nne
931          { link/Named }
932          { StructParent }
933          { \tag_struct_parent_int: }
934      }
935    }
```

68

```
936
937    \hook_gput_code:nnn
938      {pdfannot/link/Named/after}
939      {tagpdf}
940      {
941        \UseTaggingSocket{link/after}
942          {
943           \tag_struct_insert_annot:ee
944             {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
945          }
946      }
947    \hook_gput_code:nnn
948      {pdfannot/link/Launch/before}
949      {tagpdf}
950      {
951        \UseTaggingSocket{link/before}
952          {
953            \pdfannot_dict_put:nne
954              { link/Launch }
955              { StructParent }
956              { \tag_struct_parent_int: }
957          }
958      }
959
960    \hook_gput_code:nnn
961      {pdfannot/link/Launch/after}
962      {tagpdf}
963      {
964        \UseTaggingSocket{link/after}
965          {
966           \tag_struct_insert_annot:ee
967             {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
968          }
969      }
970    }
```

13.8 Attaching css-files for derivation

Derivation to html (https://pdfa.org/wp-content/uploads/2019/06/Deriving_HTML_-
from_PDF.pdf, implemented by, e.g., ngpdf) can be improved by attaching CSS style
definitions in associated files with relationship supplement to the Catalog[1].

Such CSS style definitions can be given in two ways:

- In files with the extension `.css`. Such files should contain only CSS style definitions.
  ngpdf will store these files and include them with an `<link rel=stylesheet href=...>`
  in the head of the html.

- In files with the extension `.html`. Such files should contain CSS style definitions
  inside one (or more) `<style>...</style>` html tags. The content of these files are
  copied by ngpdf directly into the head of the derived html.

---

[1] Previously they suggested the StructTreeRoot, but this is not compatible with pdf/A-3

By default (if tagging is active) tagpdf embeds now such CSS style definitions. Currently the list of files is rather short and consists of two files (with extension .html and `<style>...</style>` html tags) which are provided by the tagpdf package:

- latex-align-css.html which improves the styling of amsmath alignments tagged with MathML.

- latex-list-css.html which improves the style of list environments.

It is possible to suppress the embedding of these files by setting the `\tagpdfsetup` key attach-css to `false`, attach-css=true or attach-css reverts this again.

For developers, `\tagpdfsetup` some keys to manipulate the list exist: With css-list={file1,file2} the list can be overwritten. css-list= clears the list (and so suppresses the embedding too). To remove a file from the list, use css-list-remove=file, e.g. css-list-remove=latex-list-css.html. To add your own file use css-list-add=my-fancy-align-css.html. It is also possible to attach a .css-file in this way.

These keys do not affect files added directly with root-supplemental-file or catalog-supplemental-file.

The files in this list are attached at the end of the compilation but you shouldn't rely on a specific order of the embedding in the html.

We want to avoid to embed files twice, so we use a prop.

```
971 \prop_new:N \g__tag_css_prop
972 \prop_gset_from_keyval:Nn \g__tag_css_prop
973   {
974     latex-list-css.html=true,
975     latex-align-css.html=true
976   }
977
978
979 \bool_new:N \g__tag_css_bool
980 \bool_gset_true:N \g__tag_css_bool
```

The files for the catalog must be added before the catalog is pushed.

```
981 \tl_gput_left:Nn \g__kernel_pdfmanagement_end_run_code_tl
982   {
983    \bool_lazy_and:nnT  { \g__tag_css_bool }{ \tag_if_active_p: }
984     {
985      \prop_map_inline:Nn \g__tag_css_prop
986       {
987         \keys_set:nn { __tag / setup }{ catalog-supplemental-file= {#1} }
988       }
989     }
990   }
991
992 \keys_define:nn { __tag / setup }
993   {
994     attach-css .bool_gset:N = \g__tag_css_bool,
995     css-list .code:n =
996       {
997         \tl_if_empty:nTF{#1}
998           {\prop_gclear:N \g__tag_css_prop }
999           {\prop_gput:Nnn \g__tag_css_prop { #1 }{true}}
1000      },
1001     css-list-add .code:n    = { \prop_gput:Nnn \g__tag_css_prop { #1 }{true} },
```

```
1002    css-list-remove .code:n = { \prop_gremove:Nn \g__tag_css_prop { #1 } },
1003  }
```

</package>   The tagpdf-tree module
Commands trees and main dictionaries
Part of the tagpdf package
    Ulrike Fischer
    Version 0.99z, released 2026-03-19

Part IV

1 ⟨@@=tag⟩
2 ⟨∗header⟩
3 \ProvidesExplPackage {tagpdf-tree-code} {2026-03-19} {0.99z}
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   \socket_use:n { tag/check/parent-child-end }
29   \msg_note:nn {tag}{tree-statistic}
30 }

(*End of definition for* __tag_tree_final_checks:.)

1.2   Catalog: MarkInfo and StructTreeRoot and OpenAction

The StructTreeRoot and the MarkInfo entry must be added to the catalog. If there is an OpenAction entry we must update it, so that it contains also a structure destination. We do it late so that we can win, but before the pdfmanagement hook.

__tag/struct/1   This is the object for the root object, the StructTreeRoot

```
31 \pdf_object_new_indexed:nn { __tag/struct }{ 1 }
```

(*End of definition for* __tag/struct/1.)

\g__tag_tree_openaction_struct_tl   We need a variable that indicates which structure is wanted in the OpenAction. By default we use 2 (the Document structure).

```
32 \tl_new:N   \g__tag_tree_openaction_struct_tl
33 \tl_gset:Nn \g__tag_tree_openaction_struct_tl { 2 }
```

(*End of definition for* \g__tag_tree_openaction_struct_tl.)

viewer/startstructure (setup-key)   We also need an option to setup the start structure. So we setup a key which sets the variable to the current structure. This still requires hyperref to do most of the job, this should perhaps be changed.

```
34 \keys_define:nn { __tag / setup }
35   {
36     viewer/startstructure .code:n =
37       {
38           \tl_gset:Ne \g__tag_tree_openaction_struct_tl {#1}
39       }
40     ,viewer/startstructure .default:n =  { \int_use:N \c@g__tag_struct_abs_int }
41   }
```

(*End of definition for* viewer/startstructure (setup-key). *This function is documented on page* **??**.)

The OpenAction should only be updated if it is there. So we inspect the Catalog-prop:

```
42 \cs_new_protected:Npn \__tag_tree_update_openaction:
43   {
44     \prop_get:cnNT
45     { \__kernel_pdfdict_name:n { g__pdf_Core/Catalog } }
46     {OpenAction}
47     \l__tag_tmpa_tl
48       {
```

we only do something if the OpenAction is an array (as set by hyperref) in other cases we hope that the author knows what they did.

```
49         \tl_if_head_eq_charcode:eNT { \tl_trim_spaces:o { \l__tag_tmpa_tl } } [ %]
50           {
51           \seq_set_split:Nno\l__tag_tmpa_seq {/} {\l__tag_tmpa_tl}
52           \pdfmanagement_add:nne {Catalog} { OpenAction }
53             {
54               <<
55                 /S/GoTo \c_space_tl
56                 /D~\l__tag_tmpa_tl\c_space_tl
57                 /SD~[\pdf_object_ref_indexed:nn{__tag/struct}{\g__tag_tree_openaction_struct
```

there should be always a /Fit etc in the array but better play safe here ...

```
58                       \int_compare:nNnTF{ \seq_count:N \l__tag_tmpa_seq } > {1}
59                       { /\seq_item:Nn\l__tag_tmpa_seq{2} }
60                       { ] }
61                 >>
62             }
63         }
64     }
65   }
```

```
66   \hook_gput_code:nnn{shipout/lastpage}{tagpdf}
67     {
68        \bool_if:NT \g__tag_active_tree_bool
69          {
70             \pdfmanagement_add:nnn { Catalog / MarkInfo } { Marked } { true }
71             \pdfmanagement_add:nne
72               { Catalog }
73               { StructTreeRoot }
74               { \pdf_object_ref_indexed:nn { __tag/struct } { 1 } }
75             \__tag_tree_update_openaction:
76          }
77     }
```

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

```
78   \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

```
79   \cs_generate_variant:Nn \tl_count:n {e}
80   \hook_gput_code:nnn{begindocument}{tagpdf}
81     {
82        \int_gset:Nn\g__tag_tree_id_pad_int
83          {\tl_count:e { \__tag_property_ref_lastpage:nn{tagstruct}{1000}}+1}
84     }
85
```

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.

```
86   \cs_new_protected:Npn \__tag_tree_write_idtree:
87     {
88        \tl_clear:N \l__tag_tmpa_tl
89        \tl_clear:N \l__tag_tmpb_tl
90        \int_zero:N \l__tag_tmpa_int
91        \int_step_inline:nnn {2} {\c@g__tag_struct_abs_int}
92          {
93             \int_incr:N\l__tag_tmpa_int
94             \tl_put_right:Ne \l__tag_tmpa_tl
95               {
96                  \__tag_struct_get_id:n{##1}~\pdf_object_ref_indexed:nn {__tag/struct}{##1}~
97               }
98             \int_compare:nNnF {\l__tag_tmpa_int}<{50} %
99               {
100                 \pdf_object_unnamed_write:ne {dict}
101                   { /Limits~[\__tag_struct_get_id:n{##1-\l__tag_tmpa_int+1}~\__tag_struct_get_id:
102                     /Names~[\l__tag_tmpa_tl]
103                   }
104                 \tl_put_right:Ne\l__tag_tmpb_tl {\pdf_object_ref_last:\c_space_tl}
```

```
105        \int_zero:N \l__tag_tmpa_int
106        \tl_clear:N \l__tag_tmpa_tl
107      }
108    }
109  \tl_if_empty:NF \l__tag_tmpa_tl
110    {
111      \pdf_object_unnamed_write:ne {dict}
112        {
113         /Limits~
114           [\__tag_struct_get_id:n{\c@g__tag_struct_abs_int-\l__tag_tmpa_int+1}~
115             \__tag_struct_get_id:n{\c@g__tag_struct_abs_int}]
116         /Names~[\l__tag_tmpa_tl]
117        }
118      \tl_put_right:Ne\l__tag_tmpb_tl {\pdf_object_ref_last:}
119    }
120  \pdf_object_unnamed_write:ne {dict}{/Kids~[\l__tag_tmpb_tl]}
121  \__tag_prop_gput:cne
122      { g__tag_struct_1_prop }
123      { IDTree }
124      { \pdf_object_ref_last: }
125  }
```

1.4 Writing structure elements

The following commands are needed to write out the structure.

`\__tag_tree_write_structtreeroot:` This writes out the root object.

```
126 \cs_new_protected:Npn \__tag_tree_write_structtreeroot:
127  {
128    \__tag_prop_gput:cne
129      { g__tag_struct_1_prop }
130      { ParentTree }
131      { \pdf_object_ref:n { __tag/tree/parenttree } }
132    \__tag_prop_gput:cne
133      { g__tag_struct_1_prop }
134      { RoleMap }
135      { \pdf_object_ref:n { __tag/tree/rolemap } }
136    \__tag_struct_fill_kid_key:n { 1 }
137    \prop_gremove:cn { g__tag_struct_1_prop } {S}
138    \__tag_struct_get_dict_content:nN { 1 } \l__tag_tmpa_tl
139    \pdf_object_write_indexed:nnne
140        { __tag/struct } { 1 }
141        {dict}
142        {
143         \l__tag_tmpa_tl
144        }
```

Better put S back, see https://github.com/latex3/tagging-project/issues/86

```
145        \prop_gput:cnn { g__tag_struct_1_prop } {S}{ /StructTreeRoot }
146  }
```

(*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.

```
147 \cs_new_protected:Npn \__tag_tree_write_structelements:
```

```
148     {
149       \int_step_inline:nnnn {2}{1}{\c@g__tag_struct_abs_int}
150         {
151           \__tag_struct_write_obj:n { ##1 }
152         }
153     }
```

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

1.5 ParentTree

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

```
154 \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 distinct 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.

```
155 \int_new:N  \c@g__tag_parenttree_obj_int
156 \hook_gput_code:nnn{begindocument}{tagpdf}
157   {
158     \int_gset:Nn
159       \c@g__tag_parenttree_obj_int
160       { \__tag_property_ref_lastpage:nn{abspage}{100}  }
161   }
```

(*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

```
162 \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.

```
163 \cs_new_protected:Npn \__tag_parenttree_add_objr:nn #1 #2 %#1 StructParent number, #2 objref
164   {
165     \tl_gput_right:Ne \g__tag_parenttree_objr_tl
166       {
167         #1 \c_space_tl #2 ^^J
168       }
169   }
```

(*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.

```
170 \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.

```
171 \cs_new_protected:Npn \__tag_tree_parenttree_rerun_msg: {}
172 \cs_new_protected:Npn \__tag_tree_fill_parenttree:
173   {
174     \int_step_inline:nnnn{1}{1}{\__tag_property_ref_lastpage:nn{abspage}{-1}} %not quite clea
175       { %page ##1
176         \prop_clear:N \l__tag_tmpa_prop
177         \int_step_inline:nnnn{1}{1}{\__tag_property_ref_lastpage:nn{tagmcabs}{-
   1}}
178           {
179             %mcid####1
180             \int_compare:nT
181               {\property_ref:enn{mcid-####1}{tagabspage}{-1}=##1} %mcid is on current page
182               {% yes
183                 \prop_get:NnNT
184                   \g__tag_mc_parenttree_prop
185                   {####1}
186                   \l__tag_tmpa_tl
187                   {
188                     \prop_put:Nee
189                       \l__tag_tmpa_prop
190                       {\property_ref:enn{mcid-####1}{tagmcid}{-1}}
191                       {\l__tag_tmpa_tl}
192                   }
193               }
194           }
195         \tl_put_right:Ne\l__tag_parenttree_content_tl
196           {
197             \int_eval:n {##1-1}\c_space_tl
198             [\c_space_tl %]
199           }
200         \int_step_inline:nnnn %####1
201           {0}
202           {1}
203           { \prop_count:N \l__tag_tmpa_prop -1 }
204           {
205             \prop_get:NnNTF \l__tag_tmpa_prop {####1} \l__tag_tmpa_tl
206               {% page#1:mcid##1:\l__tag_tmpa_tl :content
207                 \tl_put_right:Ne \l__tag_parenttree_content_tl
208                   {
209                     \prop_if_exist:cTF  { g__tag_struct_ \l__tag_tmpa_tl _prop  }
210                       {
211                         \pdf_object_ref_indexed:nn { __tag/struct }{ \l__tag_tmpa_tl }
212                       }
213                       {
214                         null
215                       }
216                     \c_space_tl
217                   }
218               }
219               {
220                 \cs_set_protected:Npn \__tag_tree_parenttree_rerun_msg:
221                   {
```

```
222                    \msg_warning:nn { tag } {tree-mcid-index-wrong}
223                  }
224                }
225              }
226          \tl_put_right:Nn
227            \l__tag_parenttree_content_tl
228            {%[
229              ]^^J
230            }
231        }
232    }
```
(*End of definition for* __tag_tree_fill_parenttree:.)

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

```
233 \cs_new_protected:Npn \__tag_tree_lua_fill_parenttree:
234   {
235      \tl_set:Nn \l__tag_parenttree_content_tl
236        {
237          \lua_now:e
238            {
239              ltx.__tag.func.output_parenttree
240                (
241                   \int_use:N\g_shipout_readonly_int
242                )
243            }
244        }
245   }
```
(*End of definition for* __tag_tree_lua_fill_parenttree:.)

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

```
246 \cs_new_protected:Npn \__tag_tree_write_parenttree:
247   {
248      \bool_if:NTF \g__tag_mode_lua_bool
249        {
250           \__tag_tree_lua_fill_parenttree:
251        }
252        {
253           \__tag_tree_fill_parenttree:
254        }
255      \__tag_tree_parenttree_rerun_msg:
256      \tl_put_right:No \l__tag_parenttree_content_tl { \g__tag_parenttree_objr_tl }
257      \pdf_object_write:nne  { __tag/tree/parenttree }{dict}
258        {
259          /Nums\c_space_tl [\l__tag_parenttree_content_tl]
260        }
261   }
```
(*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.

```
262 \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.

```
263 \cs_new_protected:Npn \__tag_tree_write_rolemap:
264 {
265   \bool_if:NT \g__tag_role_add_mathml_bool
266     {
267       \prop_map_inline:Nn \g__tag_role_NS_mathml_prop
268         {
269           \prop_gput:Nnn \g__tag_role_rolemap_prop {##1}{Span}
270         }
271     }
272   \prop_map_inline:Nn\g__tag_role_rolemap_prop
273     {
274       \tl_if_eq:nnF {##1}{##2}
275         {
276           \pdfdict_gput:nne {g__tag_role/RoleMap_dict}
277           {##1}
278           {\pdf_name_from_unicode_e:n{##2}}
279         }
280     }
281   \pdf_object_write:nne  { __tag/tree/rolemap }{dict}
282     {
283       \pdfdict_use:n{g__tag_role/RoleMap_dict}
284     }
285 }
```

(*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:

```
286 \cs_new_protected:Npn \__tag_tree_write_classmap:
287   {
288     \tl_clear:N \l__tag_tmpa_tl
```

We process the older sec for compatibility with the table code. TODO: check if still needed

```
289     \seq_map_inline:Nn \g__tag_attr_class_used_seq
290       {
291         \prop_gput:Nnn \g__tag_attr_class_used_prop {##1}{}
292       }
293     \prop_map_inline:Nn \g__tag_attr_class_used_prop
294       {
295         \prop_get:NnNT \g__tag_attr_entries_prop {##1} \l__tag_tmpb_tl
296           {
297             \tl_put_right:Ne \l__tag_tmpa_tl
298               {
```

```
299              ##1\c_space_tl
300              <<
301               \l__tag_tmpb_tl
302              >>
303             \iow_newline:
304           }
305        }
306     }
307   \tl_if_empty:NF
308     \l__tag_tmpa_tl
309     {
310       \pdf_object_new:n { __tag/tree/classmap }
311       \pdf_object_write:nne
312         { __tag/tree/classmap }
313         {dict}
314         { \l__tag_tmpa_tl }
315       \__tag_prop_gput:cne
316         { g__tag_struct_1_prop }
317         { ClassMap }
318         { \pdf_object_ref:n { __tag/tree/classmap }  }
319     }
320   }
```

(*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.

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

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

```
322 \cs_new_protected:Npn \__tag_tree_write_namespaces:
323   {
324     \pdf_version_compare:NnF < {2.0}
325       {
326         \prop_map_inline:Nn \g__tag_role_NS_prop
327           {
328             \pdfdict_if_empty:nF {g__tag_role/RoleMapNS_##1_dict}
329               {
330                 \pdf_object_write:nne {__tag/RoleMapNS/##1}{dict}
331                   {
332                     \pdfdict_use:n {g__tag_role/RoleMapNS_##1_dict}
333                   }
334                 \pdfdict_gput:nne{g__tag_role/Namespace_##1_dict}
335                   {RoleMapNS}{\pdf_object_ref:n {__tag/RoleMapNS/##1}}
336               }
337             \pdf_object_write:nne{tag/NS/##1}{dict}
338               {
339                 \pdfdict_use:n {g__tag_role/Namespace_##1_dict}
340               }
```

```
341          }
342      \pdf_object_write:nne {__tag/tree/namespaces}{array}
343        {
344            \prop_map_tokens:Nn \g__tag_role_NS_prop{\use_ii:nn}
345        }
346    }
347  }
```
(*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:

```
348  \hook_new:n {tagpdf/finish/before}
349  \cs_new_protected:Npn \__tag_finish_structure:
350    {
351      \bool_if:NT\g__tag_active_tree_bool
352        {
353          \hook_use:n {tagpdf/finish/before}
354          \__tag_tree_final_checks:
355          \iow_term:n{Package~tagpdf~Info:~writing~ParentTree}
356          \__tag_check_benchmark_tic:
357          \__tag_tree_write_parenttree:
358          \__tag_check_benchmark_toc:
359          \iow_term:n{Package~tagpdf~Info:~writing~IDTree}
360          \__tag_check_benchmark_tic:
361          \__tag_tree_write_idtree:
362          \__tag_check_benchmark_toc:
363          \iow_term:n{Package~tagpdf~Info:~writing~RoleMap}
364          \__tag_check_benchmark_tic:
365          \__tag_tree_write_rolemap:
366          \__tag_check_benchmark_toc:
367          \iow_term:n{Package~tagpdf~Info:~writing~ClassMap}
368          \__tag_check_benchmark_tic:
369          \__tag_tree_write_classmap:
370          \__tag_check_benchmark_toc:
371          \iow_term:n{Package~tagpdf~Info:~writing~NameSpaces}
372          \__tag_check_benchmark_tic:
373          \__tag_tree_write_namespaces:
374          \__tag_check_benchmark_toc:
375          \iow_term:n{Package~tagpdf~Info:~writing~StructElems}
376          \__tag_check_benchmark_tic:
377          \__tag_tree_write_structelements: %this is rather slow!!
378          \__tag_check_benchmark_toc:
379          \iow_term:n{Package~tagpdf~Info:~writing~Root}
380          \__tag_check_benchmark_tic:
381          \__tag_tree_write_structtreeroot:
382          \__tag_check_benchmark_toc:
383        }
384    }
385  ⟨/package⟩
```
(*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.

```
386 ⟨*package⟩
387 \hook_gput_code:nnn{begindocument}{tagpdf}
388   {
389     \bool_if:NT\g__tag_active_tree_bool
390       {
391         \hook_gput_code:nnn{shipout/before} { tagpdf/structparents }
392           {
393             \pdfmanagement_add:nne
394               { Page }
395               { StructParents }
396               { \int_eval:n { \g_shipout_readonly_int} }
397           }
398       }
399   }
400 ⟨/package⟩
```

The `tagpdf-mc-shared` module
Code related to Marked Content (mc-chunks), code shared by all modes
Part of the tagpdf package
    Ulrike Fischer
    Version 0.99z, released 2026-03-19

Part V

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:

New: 2019-11-20

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

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

New: 2021-04-22

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

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 *

New: 2023-06-11

\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.

\tag_mc_add_missing_to_stream:Nn \tag_mc_add_missing_to_stream:Nn ⟨*box*⟩ {⟨*stream name*⟩}

This command is only needed in generic mode, in lua mode it gobbles its arguments. In generic mode it adds MC literals to the stream that are missing because of page breaks. The first argument is the box with the stream, the second a string representing the stream. Predeclared are the names `main`, `footnote` and `multicol`. If more streams should be handle the underlying interface must be enabled with `\tag_mc_new_stream:n` The command is only for packages doing deep manipulations of the output routine! Example of use are in the multicol package and in tagpdf itself.

\tag_mc_new_stream:n \tag_mc_new_stream:n {⟨*stream name*⟩}

This declares the interface needed to handle a new stream with `\tag_mc_add_missing_-to_stream:Nn`. Predeclared are the names `main`, `footnote` and `multicol`.

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 split 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} {2026-03-19} {0.99z}
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_abs_int and \tl_put_right:Nn\cl@@ckpt{\@elt{g_@@_MCID_abs_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_linked: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 \l__tag_mc_artifact_bool	This booleans store the stash and artifact status of the mc-chunk.

```
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_lang_tl	a variable to set a Lang on the mc. This is not conforming to the spec! But it seems to work in acrobat.

```
22  \tl_new:N \l__tag_mc_lang_tl
```
(*End of definition for* \l__tag_mc_lang_tl.)

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

```
23  \tl_new:N \l__tag_mc_key_tag_tl
24  \tl_new:N \g__tag_mc_key_tag_tl
25  \tl_new:N \l__tag_mc_key_label_tl
26  \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.

```
27  \cs_new:Npn \__tag_mc_handle_mc_label:e #1
28    {
29      \__tag_property_record:en{tagpdf-#1}{tagabspage,tagmcabs}
30    }
```
(*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

```
31 \cs_new_protected:Npn \__tag_mc_set_label_used:n #1 %#1 labelname
32   {
33     \tl_new:c { g__tag_mc_label_\tl_to_str:n{#1}_used_tl }
34   }
35 ⟨/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?

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

(*End of definition for* `\tag_mc_use:n`. *This function is documented on page 83.*)

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

```
60 ⟨base⟩\cs_new_protected:Npn \tag_mc_artifact_group_begin:n #1 {}
61 ⟨base⟩\cs_new_protected:Npn \tag_mc_artifact_group_end:{}
62 ⟨*shared⟩
63 \cs_set_protected:Npn \tag_mc_artifact_group_begin:n #1
64   {
65     \tag_mc_end_push:
66     \tag_mc_begin:n {artifact=#1}
67     \group_begin:
68     \tag_suspend:n{artifact-group}
69   }
70
```

```
71  \cs_set_protected:Npn \tag_mc_artifact_group_end:
72    {
73      \tag_resume:n{artifact-group}
74      \group_end:
75      \tag_mc_end:
76      \tag_mc_begin_pop:n{}
77    }
78  ⟨/shared⟩
```

(*End of definition for* `\tag_mc_artifact_group_begin:n` *and* `\tag_mc_artifact_group_end:`. *These functions are documented on page 83.*)

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

```
79  ⟨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 83.*)

`\tag_mc_end_push:`
`\tag_mc_begin_pop:n`

```
80  ⟨base⟩\cs_new_protected:Npn \tag_mc_end_push: {}
81  ⟨base⟩\cs_new_protected:Npn \tag_mc_begin_pop:n #1 {}
82  ⟨∗shared⟩
83  \cs_set_protected:Npn \tag_mc_end_push:
84    {
85      \__tag_check_if_active_mc:T
86        {
87          \__tag_mc_if_in:TF
88            {
89              \seq_gpush:Ne \g__tag_mc_stack_seq { \tag_get:n {mc_tag} }
90              \__tag_check_mc_pushed_popped:nn
91                { pushed }
92                { \tag_get:n {mc_tag} }
93              \tag_mc_end:
94            }
95            {
96              \seq_gpush:Nn \g__tag_mc_stack_seq {-1}
97              \__tag_check_mc_pushed_popped:nn { pushed }{-1}
98            }
99        }
100   }
101
102 \cs_set_protected:Npn \tag_mc_begin_pop:n #1
103   {
104     \__tag_check_if_active_mc:T
105       {
106         \seq_gpop:NNTF \g__tag_mc_stack_seq \l__tag_tmpa_tl
107           {
108             \tl_if_eq:NnTF \l__tag_tmpa_tl {-1}
109               {
110                 \__tag_check_mc_pushed_popped:nn {popped}{-1}
111               }
112               {
113                 \__tag_check_mc_pushed_popped:nn {popped}{\l__tag_tmpa_tl}
114                 \tag_mc_begin:n {tag=\l__tag_tmpa_tl,#1}
115               }
```

```
116                    }
117                    {
118                        \__tag_check_mc_pushed_popped:nn {popped}{empty~stack,~nothing}
119                    }
120                }
121            }
```

(*End of definition for* \tag_mc_end_push: *and* \tag_mc_begin_pop:n. *These functions are documented on page* 83.)

__tag_mc_check_parent_child:n   This checks if an MC can be used in a structure.

```
122    \cs_new_protected:Npn \__tag_mc_check_parent_child:n #1
123    % #1 structure number of parent
124        {
```

This records if logging is on

```
125        \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
126            {
127                \prop_get:cnN{g__tag_struct_#1_prop}{tag}\l__tag_get_parent_tmpa_tl
128                \msg_note:nnee
129                { tag }
130                { role-parent-child-check }
131                {
132                    \quark_if_no_value:NTF \l__tag_get_parent_tmpa_tl
133                    {??}
134                    {
135                        \exp_last_unbraced:No\use_ii:nn
136                        { \l__tag_get_parent_tmpa_tl }
137                        :
138                        \exp_last_unbraced:No\use_i:nn
139                        { \l__tag_get_parent_tmpa_tl }
140                    }
141                }
142                {
143                    MC~(real~content)
144                }
145            }
146        \__tag_struct_get_role:nnNN
147            {#1}
148            {rolemap}
149            \l__tag_get_parent_tmpa_tl
150            \l__tag_get_parent_tmpb_tl
151        \__tag_role_check_parent_child:ooooN
152            { \l__tag_get_parent_tmpa_tl }
153            { \l__tag_get_parent_tmpb_tl }
154            { MC } %
155            {  } %
156            \l__tag_parent_child_check_tl
```

if the return value is 7 we have to check against the parentrole field. TODO ruby and warichu use 7 too, that should be changed!

```
157        \int_compare:nNnT {\l__tag_parent_child_check_tl} = { \c__tag_role_rule_checkparent_tl }
158            {
159                \__tag_struct_get_role:nnNN
160                    {#1}
```

89

```
161        {parentrole}
162        \l__tag_get_parent_tmpa_tl
163        \l__tag_get_parent_tmpb_tl
164      \__tag_role_check_parent_child:ooooN
165        { \l__tag_get_parent_tmpa_tl }
166        { \l__tag_get_parent_tmpb_tl }
167        { MC } %
168        {  } %
169        \l__tag_parent_child_check_tl
170      }
171    \__tag_check_forbidden_parent_child:nnee
172    {\l__tag_parent_child_check_tl}
173    {#1}
174    {
175      \l__tag_get_parent_tmpb_tl : \l__tag_get_parent_tmpa_tl
176    }
177    {
178      MC~(real content)
179    }
180  }
181 \cs_generate_variant:Nn \__tag_mc_check_parent_child:n {o}
```
(*End of definition for* `\__tag_mc_check_parent_child:n`.)

3.3 Keys

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

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

```
182 \keys_define:nn { __tag / mc }
183  {
184    stash                     .bool_set:N    = \l__tag_mc_key_stash_bool,
185    __artifact-bool           .bool_set:N    = \l__tag_mc_artifact_bool,
186    __artifact-type           .choice:,
187    __artifact-type / pagination .code:n    =
188      {
189        \tl_set:Nn \l__tag_mc_artifact_type_tl { Pagination }
190      },
191    __artifact-type / pagination/header .code:n    =
192      {
193        \tl_set:Nn \l__tag_mc_artifact_type_tl { Pagination/Subtype/Header }
194      },
195    __artifact-type / pagination/footer .code:n    =
196      {
197        \tl_set:Nn \l__tag_mc_artifact_type_tl { Pagination/Subtype/Footer }
198      },
199    __artifact-type / layout      .code:n    =
200      {
201        \tl_set:Nn \l__tag_mc_artifact_type_tl { Layout }
202      },
203    __artifact-type / page        .code:n    =
```

```
  204        {
  205          \tl_set:Nn \l__tag_mc_artifact_type_tl { Page }
  206        },
  207      __artifact-type / background .code:n    =
  208        {
  209          \tl_set:Nn \l__tag_mc_artifact_type_tl { Background }
  210        },
  211      __artifact-type / notype      .code:n    =
  212        {
  213          \tl_set:Nn \l__tag_mc_artifact_type_tl {}
  214        },
  215      __artifact-type /        .code:n    =
  216        {
  217          \tl_set:Nn \l__tag_mc_artifact_type_tl {}
  218        },
  219    }
```

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

```
  220  ⟨/shared⟩
```

The `tagpdf-mc-generic` module
Code related to Marked Content (mc-chunks), generic mode
Part of the tagpdf package
    Ulrike Fischer
    Version 0.99z, released 2026-03-19

Part VI

1 Marked content code – generic mode

```
1  ⟨@@=tag⟩
2  ⟨*generic⟩
3  \ProvidesExplPackage {tagpdf-mc-code-generic} {2026-03-19} {0.99z}
4    {part of tagpdf - code related to marking chunks - generic mode}
5  ⟨/generic⟩
6  ⟨*debug⟩
7  \ProvidesExplPackage {tagpdf-debug-generic} {2026-03-19} {0.99z}
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
\g_tag_mc_footnote_marks_seq
\g_tag_mc_multicol_marks_seq   Each stream has an associated global seq variable holding the bottom marks from the/a previous chunk in the stream. We provide three by default: main, footnote and multicol. TODO: perhaps an interface for more streams will be needed.

```
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.)

\tag_mc_new_stream:n

```
17  \cs_new_protected:Npn  \tag_mc_new_stream:n #1
18    {
19      \seq_new:c { g__tag_mc_multicol_#1_seq }
20    }
```

(*End of definition for* \tag_mc_new_stream:n. *This function is documented on page 84.*)

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

```
21 \seq_new:N  \l__tag_mc_firstmarks_seq
22 \seq_new:N  \l__tag_mc_botmarks_seq
```

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

1.2 Functions

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

```
23 \cs_new_protected:Npn \__tag_mc_begin_marks:nn #1 #2 %#1 tag, #2 label
24   {
25     \tex_marks:D \g__tag_mc_marks
26       {
27         b-, %first of begin pair
28         \int_use:N\c@g__tag_MCID_abs_int, %mc-num
29         \g__tag_struct_stack_current_tl,  %structure num
30         #1, %tag
31         \bool_if:NT \l__tag_mc_key_stash_bool{stash}, % stash info
32         #2, %label
33       }
34     \tex_marks:D \g__tag_mc_marks
35       {
36         b+, % second of begin pair
37         \int_use:N\c@g__tag_MCID_abs_int, %mc-num
38         \g__tag_struct_stack_current_tl,  %structure num
39         #1, %tag
40         \bool_if:NT \l__tag_mc_key_stash_bool{stash}, % stash info
41         #2, %label
42       }
43   }
44 \cs_generate_variant:Nn \__tag_mc_begin_marks:nn {oo}
45 \cs_new_protected:Npn \__tag_mc_artifact_begin_marks:n #1 %#1 type
46   {
47     \tex_marks:D \g__tag_mc_marks
48       {
49         b-, %first of begin pair
50         \int_use:N\c@g__tag_MCID_abs_int, %mc-num
51         -1, %structure num
52         #1 %type
53       }
54     \tex_marks:D \g__tag_mc_marks
55       {
56         b+, %first of begin pair
57         \int_use:N\c@g__tag_MCID_abs_int, %mc-num
58         -1,  %structure num
59         #1  %Type
60       }
61   }
```

```
62
63  \cs_new_protected:Npn \__tag_mc_end_marks:
64    {
65      \tex_marks:D \g__tag_mc_marks
66        {
67          e-, %first of end pair
68          \int_use:N\c@g__tag_MCID_abs_int, %mc-num
69          \g__tag_struct_stack_current_tl,  %structure num
70        }
71      \tex_marks:D \g__tag_mc_marks
72        {
73          e+, %second of end pair
74          \int_use:N\c@g__tag_MCID_abs_int, %mc-num
75          \g__tag_struct_stack_current_tl,  %structure num
76        }
77    }
```

(*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.

```
78  \cs_new_protected:Npn \__tag_mc_disable_marks:
79    {
80      \cs_set_eq:NN \__tag_mc_begin_marks:nn \use_none:nn
81      \cs_set_eq:NN \__tag_mc_artifact_begin_marks:n \use_none:n
82      \cs_set_eq:NN \__tag_mc_end_marks: \prg_do_nothing:
83    }
```

(*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.

```
84  \cs_new_protected:Npn \__tag_mc_get_marks:
85    {
86      \exp_args:NNe
87      \seq_set_from_clist:Nn \l__tag_mc_firstmarks_seq
88        { \tex_firstmarks:D \g__tag_mc_marks }
89      \exp_args:NNe
90      \seq_set_from_clist:Nn \l__tag_mc_botmarks_seq
91        { \tex_botmarks:D   \g__tag_mc_marks }
92    }
```

(*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.

```
93  \cs_new_protected:Npn \__tag_mc_store:nnn #1 #2 #3 %#1 mc-prev, #2 mc-num #3 structure-
    num
94    {
95      %\prop_show:N \g__tag_struct_cont_mc_prop
96      \prop_get:NnNTF \g__tag_struct_cont_mc_prop {#1} \l__tag_tmpa_tl
97        {
98          \prop_gput:Nne \g__tag_struct_cont_mc_prop {#1}{ \l__tag_tmpa_tl \__tag_struct_mcid_o
99        }
```

94

```
100            {
101                \prop_gput:Nne \g__tag_struct_cont_mc_prop {#1}{ \__tag_struct_mcid_dict:n {#2}}
102            }
103        \prop_gput:Nee \g__tag_mc_parenttree_prop
104          {#2}
105          {#3}
106    }
107 \cs_generate_variant:Nn \__tag_mc_store:nnn {eee}
```

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

`\__tag_mc_insert_extra_tmb:n`
`\__tag_mc_insert_extra_tme:n`

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

```
108 \cs_new_protected:Npn \__tag_mc_insert_extra_tmb:n #1 % #1 stream: e.g. main or footnote
109    {
110        \__tag_check_typeout_v:n {=>~ first~ \seq_use:Nn \l__tag_mc_firstmarks_seq {,~}}
111        \__tag_check_typeout_v:n {=>~ bot~ \seq_use:Nn \l__tag_mc_botmarks_seq {,~}}
112        \__tag_check_if_mc_tmb_missing:TF
113          {
114            \__tag_check_typeout_v:n {=>~ TMB~ ~ missing~ --~ inserted}
115            %test if artifact
116            \int_compare:nNnTF { \seq_item:cn { g__tag_mc_#1_marks_seq } {3} } = {-
    1}
117              {
118                \tl_set:Ne \l__tag_tmpa_tl { \seq_item:cn { g__tag_mc_#1_marks_seq } {4} }
119                \__tag_mc_handle_artifact:N \l__tag_tmpa_tl
120              }
121              {
122                \exp_args:Ne
123                \__tag_mc_bdc_mcid:n
124                  {
125                    \seq_item:cn { g__tag_mc_#1_marks_seq } {4}
126                  }
127                \str_if_eq:eeTF
128                  {
129                    \seq_item:cn { g__tag_mc_#1_marks_seq } {5}
130                  }
131                  {}
132                  {
133                    %store
134                    \__tag_mc_store:eee
135                      {
136                        \seq_item:cn { g__tag_mc_#1_marks_seq } {2}
137                      }
138                      { \int_eval:n{\c@g__tag_MCID_abs_int} }
139                      {
140                        \seq_item:cn { g__tag_mc_#1_marks_seq } {3}
141                      }
```

95

```
142                    }
143                  {
144                        %stashed -> warning!!
145                  }
146              }
147          }
148          {
149            \__tag_check_typeout_v:n {=>~ TMB~ not~ missing}
150          }
151    }
152
153 \cs_new_protected:Npn \__tag_mc_insert_extra_tme:n #1 % #1 stream, eg. main or footnote
154  {
155    \__tag_check_if_mc_tme_missing:TF
156      {
157        \__tag_check_typeout_v:n {=>~ TME~ ~ missing~ --~ inserted}
158        \__tag_mc_emc:
159        \seq_gset_eq:cN
160          { g__tag_mc_#1_marks_seq }
161          \l__tag_mc_botmarks_seq
162      }
163      {
164        \__tag_check_typeout_v:n {=>~ TME~ not~ missing}
165      }
166  }
```

(*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.

```
167 \cs_new_protected:Npn\__tag_add_missing_mcs:Nn #1 #2 {
168    \vbadness \@M
169    \vfuzz     \c_max_dim
170    \vbox_set_to_ht:Nnn #1 { \box_ht:N #1 } {
171      \hbox_set:Nn \l__tag_tmpa_box { \__tag_mc_insert_extra_tmb:n {#2} }
172      \hbox_set:Nn \l__tag_tmpb_box { \__tag_mc_insert_extra_tme:n {#2} }
173      \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
174          {
175            \seq_log:c { g__tag_mc_#2_marks_seq}
176          }
```

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

```
177        \box_set_ht:Nn \l__tag_tmpa_box \c_zero_dim
178        \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.

```
179        \box_set_ht:Nn \l__tag_tmpb_box \c_zero_dim
180        \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.

```
181        \boxmaxdepth \@maxdepth
182        \box_use_drop:N        \l__tag_tmpa_box
183        \vbox_unpack_drop:N      #1
```

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

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

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

```
185        \nointerlineskip
186        \box_use_drop:N \l__tag_tmpb_box
187      }
188    }
```

(*End of definition for* __tag_add_missing_mcs:Nn.)

\tag_mc_add_missing_to_stream:Nn
__tag_add_missing_mcs_to_stream:Nn
This is the main command to add mc to the stream. It is therefore 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 artificially 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.

```
189  \cs_new_protected:Npn \__tag_add_missing_mcs_to_stream:Nn #1#2
190    {
191      \__tag_check_if_active_mc:T {
```

First set up a temp box for trial splitting.

```
192        \vbadness\maxdimen
193        \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).

```
194        \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

```
195        \exp_args:NNe
196        \seq_set_from_clist:Nn \l__tag_mc_firstmarks_seq
197          { \tex_splitfirstmarks:D \g__tag_mc_marks }
```

Some debugging info:

```
198  %      \iow_term:n { First~ mark~ from~ this~ box: }
199  %      \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!

```
200      \seq_if_empty:NTF \l__tag_mc_firstmarks_seq
201        {
202          \__tag_check_typeout_v:n
203            {
204              No~ marks~ so~ use~ saved~ bot~ mark:~
205              \seq_use:cn {g__tag_mc_#2_marks_seq} {,~} \iow_newline:
206            }
207          \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).

```
208          \seq_set_eq:NN \l__tag_mc_botmarks_seq \l__tag_mc_firstmarks_seq
209        }
```

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.

```
210        {
211          \__tag_check_typeout_v:n
212            {
213              Pick~ up~ new~ bot~ mark!
214            }
215          \exp_args:NNe
216          \seq_set_from_clist:Nn \l__tag_mc_botmarks_seq
217            { \tex_splitbotmarks:D   \g__tag_mc_marks }
218        }
```

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

```
219      \__tag_add_missing_mcs:Nn #1 {#2}
220  %%
221      \seq_gset_eq:cN  {g__tag_mc_#2_marks_seq} \l__tag_mc_botmarks_seq
222  %%
223    }
224  }
225  \cs_set_eq:NN \tag_mc_add_missing_to_stream:Nn \__tag_add_missing_mcs_to_stream:Nn
```

(*End of definition for* \tag_mc_add_missing_to_stream:Nn *and* __tag_add_missing_mcs_to_stream:Nn. *This function is documented on page 84.*)

__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.

```
226  \prg_new_conditional:Nnn \__tag_mc_if_in: {p,T,F,TF}
227    {
228      \bool_if:NTF \g__tag_in_mc_bool
```

```
229        { \prg_return_true:  }
230        { \prg_return_false: }
231    }
232
233  \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* *83*.)

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

```
234  % #1 tag, #2 properties
235  \cs_set_eq:NN \__tag_mc_bmc:n  \pdf_bmc:n
236  \cs_set_eq:NN \__tag_mc_emc:    \pdf_emc:
237  \cs_set_eq:NN \__tag_mc_bdc:nn \pdf_bdc:nn
238  \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   This create a BDC mark with an /MCID key. Most of the work here is to get the current
__tag_mc_handle_mcid:nn number value for the MCID: they must be numbered by page starting with 0 and then
__tag_mc_handle_mcid:oo 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.

```
239  \hook_gput_code:nnn {shipout/before}{tagpdf}{ \flag_clear:n { __tag/mcid } }
240  \cs_set_protected:Npn \__tag_mc_bdc_mcid:nn #1 #2
241    {
242      \int_gincr:N \c@g__tag_MCID_abs_int
243      \__tag_property_record:eo
244        {
245          mcid-\int_use:N \c@g__tag_MCID_abs_int
246        }
247      { \c__tag_property_mc_clist }
248      \__tag_mc_bdc_shipout:ee
249        {#1}
250        {
251          /MCID~\flag_height:n { __tag/mcid }
252          \flag_raise:n { __tag/mcid }~  #2
253        }
254    }
255  \cs_new_protected:Npn \__tag_mc_bdc_mcid:n #1
256    {
257      \__tag_mc_bdc_mcid:nn {#1} {}
258    }
259
260  \cs_new_protected:Npn \__tag_mc_handle_mcid:nn #1 #2 %#1 tag, #2 properties
261    {
262      \__tag_mc_bdc_mcid:nn {#1} {#2}
263    }
264
```

99

₂₆₅ `\cs_generate_variant:Nn \__tag_mc_handle_mcid:nn {oo}`

(*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?

₂₆₆ `\cs_new_protected:Npn \__tag_mc_handle_stash:n #1 %1 mcidnum`
₂₆₇ `  {`
₂₆₈ `    \__tag_check_mc_used:n {#1}`
₂₆₉ `    \__tag_struct_kid_mc_gput_right:nn`
₂₇₀ `      { \g__tag_struct_stack_current_tl }`
₂₇₁ `      {#1}`
₂₇₂ `    \prop_gput:Nee \g__tag_mc_parenttree_prop`
₂₇₃ `      {#1}`
₂₇₄ `      { \g__tag_struct_stack_current_tl }`
₂₇₅ `  }`
₂₇₆ `\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

₂₇₇ `\cs_new_protected:Npn  \__tag_mc_bmc_artifact:`
₂₇₈ `  {`
₂₇₉ `    \__tag_mc_bmc:n {Artifact}`
₂₈₀ `  }`
₂₈₁ `\cs_new_protected:Npn \__tag_mc_bmc_artifact:n #1`
₂₈₂ `  {`
₂₈₃ `    \__tag_mc_bdc:nn {Artifact}{/Type/#1}`
₂₈₄ `  }`
₂₈₅ `\cs_new_protected:Npn \__tag_mc_handle_artifact:N #1`
₂₈₆ `  % #1 is a var containing the artifact type`
₂₈₇ `  {`
₂₈₈ `    \int_gincr:N \c@g__tag_MCID_abs_int`
₂₈₉ `    \tl_if_empty:NTF #1`
₂₉₀ `      { \__tag_mc_bmc_artifact: }`
₂₉₁ `      { \exp_args:No\__tag_mc_bmc_artifact:n {#1} }`
₂₉₂ `  }`

(*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.

₂₉₃ `\cs_new:Nn \__tag_get_data_mc_tag: { \g__tag_mc_key_tag_tl }`
₂₉₄ `⟨/generic⟩`

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

`\tag_mc_begin:n`
`\tag_mc_end:`

These are the core public commands to open and close an mc. They don't need to be 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.

```
295 ⟨base⟩\cs_new_protected:Npn \tag_mc_begin:n #1 { \__tag_whatsits: \int_gincr:N \c@g__tag_MCID_
296 ⟨base⟩\cs_new_protected:Nn \tag_mc_end:{ \__tag_whatsits: }
297 ⟨∗generic | debug⟩
298 ⟨∗generic⟩
299 \cs_set_protected:Npn \tag_mc_begin:n #1 %#1 keyval
300   {
301     \__tag_check_if_active_mc:T
302       {
303 ⟨/generic⟩
304 ⟨∗debug⟩
305 \cs_set_protected:Npn \tag_mc_begin:n #1 %#1 keyval
306   {
307     \__tag_check_if_active_mc:TF
308       {
309         \__tag_debug_mc_begin_insert:n { #1 }
310 ⟨/debug⟩
311         \group_begin: %hm
312         \__tag_check_mc_if_nested:
313         \bool_gset_true:N \g__tag_in_mc_bool
```

set default MC tags to structure:

```
314         \tl_set_eq:NN \l__tag_mc_key_tag_tl \g__tag_struct_tag_tl
315         \tl_gset_eq:NN\g__tag_mc_key_tag_tl \g__tag_struct_tag_tl
316         \tl_if_empty:NTF\l__tag_mc_lang_tl
317           {
318             \keys_set:nn { __tag / mc }{ #1 }
319           }
320           {
321             \keys_set:nn { __tag / mc }{ lang=\l__tag_mc_lang_tl, #1 }
322           }
323         \bool_if:NTF \l__tag_mc_artifact_bool
324           { %handle artifact
325             \__tag_mc_handle_artifact:N \l__tag_mc_artifact_type_tl
326             \exp_args:No
327             \__tag_mc_artifact_begin_marks:n { \l__tag_mc_artifact_type_tl }
328           }
329           { %handle mcid type
330             \__tag_check_mc_tag:N   \l__tag_mc_key_tag_tl
331             \__tag_mc_handle_mcid:oo
332               { \l__tag_mc_key_tag_tl }
333               { \l__tag_mc_key_properties_tl }
334             \__tag_mc_begin_marks:oo{\l__tag_mc_key_tag_tl}{\l__tag_mc_key_label_tl}
335             \tl_if_empty:NF {\l__tag_mc_key_label_tl}
336               {
337                 \__tag_mc_handle_mc_label:e { \l__tag_mc_key_label_tl }
338               }
```

check if the MC can be used here. This is guarded by the stash boolean.

```
339               \bool_if:NF \l__tag_mc_key_stash_bool
340                 {
341                   \socket_use:nn{tag/check/parent-child}
342                     {
343                       \__tag_mc_check_parent_child:o
```

101

```
344                        { \g__tag_struct_stack_current_tl }
345                      }
346                  \__tag_mc_handle_stash:e { \int_use:N \c@g__tag_MCID_abs_int }
347
348              }
349            }
350          \group_end:
351        }
352 ⟨∗debug⟩
353        {
354          \__tag_debug_mc_begin_ignore:n { #1 }
355        }
356 ⟨/debug⟩
357    }
358 ⟨∗generic⟩
359 \cs_set_protected:Nn \tag_mc_end:
360   {
361     \__tag_check_if_active_mc:T
362       {
363 ⟨/generic⟩
364 ⟨∗debug⟩
365 \cs_set_protected:Nn \tag_mc_end:
366   {
367     \__tag_check_if_active_mc:TF
368       {
369         \__tag_debug_mc_end_insert:
370 ⟨/debug⟩
371         \__tag_check_mc_if_open:
372         \bool_gset_false:N \g__tag_in_mc_bool
373         \tl_gset:Nn  \g__tag_mc_key_tag_tl { }
374         \__tag_mc_emc:
375         \__tag_mc_end_marks:
376       }
377 ⟨∗debug⟩
378       {
379         \__tag_debug_mc_end_ignore:
380       }
381 ⟨/debug⟩
382    }
383 ⟨/generic | debug⟩
```

(*End of definition for* \tag_mc_begin:n *and* \tag_mc_end:. *These functions are documented on page 83.*)

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.

```
384 ⟨∗generic⟩
385 \keys_define:nn { __tag / mc }
386   {
387     tag .code:n = % the name (H,P,Span) etc
388       {
```

```
389          \tl_set:Ne    \l__tag_mc_key_tag_tl { #1 }
390          \tl_gset:Ne   \g__tag_mc_key_tag_tl { #1 }
391        },
392      raw  .code:n =
393        {
394          \tl_put_right:Ne \l__tag_mc_key_properties_tl { #1 }
395        },
396      alt .code:n      = % Alt property
397        {
398          \str_set_convert:Noon
399            \l__tag_tmpa_str
400            { #1 }
401            { default }
402            { utf16/hex }
403          \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
404          \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
405        },
406      alttext .meta:n = {alt=#1},
```

lang is not according to the spec, but it works in acrobat …. We assume that this are
simple strings that do not need escaping.

```
407      lang .code:n      = % Lang property
408        {
409          \tl_put_right:Ne \l__tag_mc_key_properties_tl { /Lang~(#1) }
410        },
411      actualtext .code:n      = % ActualText property
412        {
413          \tl_if_empty:oF{#1}
414          {
415            \str_set_convert:Noon
416              \l__tag_tmpa_str
417              { #1 }
418              { default }
419              { utf16/hex }
420            \tl_put_right:Nn \l__tag_mc_key_properties_tl { /ActualText~< }
421            \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
422          }
423        },
424      label .tl_set:N        = \l__tag_mc_key_label_tl,
425      artifact .code:n      =
426        {
427          \exp_args:Nne
428            \keys_set:nn
429              { __tag / mc }
430              { __artifact-bool, __artifact-type=#1 }
431        },
432      artifact .default:n    = {notype}
433    }
434  ⟨/generic⟩
```

(*End of definition for* tag (mc-key) *and others. These functions are documented on page 84.*)

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

Ulrike Fischer

Part VII

The code is split 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} {2026-03-19} {0.99z}
4    {tagpdf - mc code only for the luamode }
5  ⟨/luamode⟩
6  ⟨*debug⟩
7  \ProvidesExplPackage {tagpdf-debug-lua} {2026-03-19} {0.99z}
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", "aft
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 {}
66 \cs_set_eq:NN \tag_mc_add_missing_to_stream:Nn \__tag_add_missing_mcs_to_stream:Nn
```

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

`\tag_mc_new_stream:n`

```
67 \cs_new_protected:Npn \tag_mc_new_stream:n #1 {}
```

(*End of definition for* `\tag_mc_new_stream:n`. *This function is documented on page 84.*)

`\__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.

```
68 \prg_new_conditional:Nnn \__tag_mc_if_in: {p,T,F,TF}
69   {
70     \int_compare:nNnTF
71       { -2147483647 }
72       =
73       {\lua_now:e
74         {
75           tex.print(\int_use:N \c_document_cctab,tex.getattribute(luatexbase.attributes.g__t
76         }
77       }
78       { \prg_return_false:  }
79       { \prg_return_true: }
80   }
81
82 \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 83.*)

`\__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.

```
83 \cs_new:Nn \__tag_mc_lua_set_mc_type_attr:n % #1 is a tag name
84   {
85     %TODO ltx.__tag.func.get_num_from("#1") seems not to return a suitable number??
86     \tl_set:Ne\l__tag_tmpa_tl{\lua_now:e{ltx.__tag.func.output_num_from ("#1")} }
87     \lua_now:e
88       {
89         tex.setattribute
90           (
91           "global",
92           luatexbase.attributes.g__tag_mc_type_attr,
93           \l__tag_tmpa_tl
94           )
95       }
96     \lua_now:e
97       {
98         tex.setattribute
99           (
100          "global",
101          luatexbase.attributes.g__tag_mc_cnt_attr,
102          \__tag_get_mc_abs_cnt:
103          )
104      }
105   }
106
107 \cs_generate_variant:Nn\__tag_mc_lua_set_mc_type_attr:n { o }
108
109 \cs_new:Nn \__tag_mc_lua_unset_mc_type_attr:
110   {
```

```
111    \lua_now:e
112      {
113        tex.setattribute
114          (
115            "global",
116            luatexbase.attributes.g__tag_mc_type_attr,
117            -2147483647
118          )
119      }
120    \lua_now:e
121      {
122        tex.setattribute
123          (
124            "global",
125            luatexbase.attributes.g__tag_mc_cnt_attr,
126            -2147483647
127          )
128      }
129  }
130
```

(*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

```
131  \cs_new:Nn \__tag_mc_insert_mcid_kids:n
132    {
133      \lua_now:e { ltx.__tag.func.mc_insert_kids (#1,0) }
134    }
135
136  \cs_new:Nn \__tag_mc_insert_mcid_single_kids:n
137    {
138      \lua_now:e {ltx.__tag.func.mc_insert_kids (#1,1) }
139    }
```

(*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.

```
140  ⟨/luamode⟩
141  ⟨*luamode | debug⟩
142  ⟨luamode⟩\cs_new_protected:Npn \__tag_mc_handle_stash:n #1 %1 mcidnum
143  ⟨debug⟩\cs_set_protected:Npn \__tag_mc_handle_stash:n #1 %1 mcidnum
144    {
145      \__tag_check_mc_used:n { #1 }
146      \seq_gput_right:cn % Don't fill a lua table due to the command in the item,
147                         % so use the kernel command
148        { g__tag_struct_kids_\g__tag_struct_stack_current_tl _seq }
149        {
150          \__tag_mc_insert_mcid_kids:n {#1}%
151        }
152  ⟨debug⟩    \seq_gput_right:cn % Don't fill a lua table due to the command in the item,
153  ⟨debug⟩                       % so use the kernel command
154  ⟨debug⟩      { g__tag_struct_debug_kids_\g__tag_struct_stack_current_tl _seq }
155  ⟨debug⟩      {
```

108

```
156 ⟨debug⟩          MC~#1%
157 ⟨debug⟩        }
158     \lua_now:e
159        {
160          ltx.__tag.func.store_struct_mcabs
161             (
162                \g__tag_struct_stack_current_tl,#1
163             )
164        }
165    }
166 ⟨/luamode | debug⟩
167 ⟨∗luamode⟩
168 \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.

```
169 \cs_set_protected:Nn \tag_mc_begin:n
170    {
171       \__tag_check_if_active_mc:T
172          {
173             \group_begin:
174             %\__tag_check_mc_if_nested:
175             \bool_gset_true:N \g__tag_in_mc_bool
176             \bool_set_false:N\l__tag_mc_artifact_bool
177             \tl_clear:N \l__tag_mc_key_properties_tl
178             \int_gincr:N \c@g__tag_MCID_abs_int
```
set the default tag to the structure:
```
179             \tl_set_eq:NN \l__tag_mc_key_tag_tl \g__tag_struct_tag_tl
180             \tl_gset_eq:NN\g__tag_mc_key_tag_tl \g__tag_struct_tag_tl
181             \lua_now:e
182                {
183                   ltx.__tag.func.store_mc_data(\__tag_get_mc_abs_cnt:,"tag","\g__tag_struct_tag_tl"
184                }
```
2025-05-23 allow lang on the MC (not really spec conform, but does work in acrobat).
```
185             \tl_if_empty:NTF\l__tag_mc_lang_tl
186                {
187                   \keys_set:nn { __tag / mc }{ label={}, #1 }
188                }
189                {
190                   \keys_set:nn { __tag / mc }{ label={},lang=\l__tag_mc_lang_tl, #1 }
191                }
192             %check that a tag or artifact has been used
193             \__tag_check_mc_tag:N \l__tag_mc_key_tag_tl
194             %set the attributes:
195             \__tag_mc_lua_set_mc_type_attr:o  { \l__tag_mc_key_tag_tl }
196             \bool_if:NF \l__tag_mc_artifact_bool
197                { % store the absolute num name in a label:
198                   \tl_if_empty:NF {\l__tag_mc_key_label_tl}
199                      {
200                         \__tag_mc_handle_mc_label:e { \l__tag_mc_key_label_tl }
201                      }
202                   % if not stashed record the absolute number
```

109

```
203         \bool_if:NF \l__tag_mc_key_stash_bool
204           {
205             \socket_use:nn{tag/check/parent-child}
206               {
207                 \__tag_mc_check_parent_child:o
208                   { \g__tag_struct_stack_current_tl }
209               }
210             \__tag_mc_handle_stash:e { \__tag_get_mc_abs_cnt: }
211           }
212       }
213     \group_end:
214   }
215 }
```

(*End of definition for* \tag_mc_begin:n. *This function is documented on page 83.*)

\tag_mc_end:

TODO: check how the use command must be guarded.

```
216 \cs_set_protected:Nn \tag_mc_end:
217   {
218     \__tag_check_if_active_mc:T
219       {
220         %\__tag_check_mc_if_open:
221         \bool_gset_false:N \g__tag_in_mc_bool
222         \bool_set_false:N\l__tag_mc_artifact_bool
223         \__tag_mc_lua_unset_mc_type_attr:
224         \tl_set:Nn  \l__tag_mc_key_tag_tl { }
225         \tl_gset:Nn \g__tag_mc_key_tag_tl { }
226       }
227   }
```

(*End of definition for* \tag_mc_end:. *This function is documented on page 83.*)

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

```
228 \cs_set_protected:Npn \tag_mc_reset_box:N #1
229   {
230     \lua_now:e
231       {
232         local~type=tex.getattribute(luatexbase.attributes.g__tag_mc_type_attr)
233         local~mc=tex.getattribute(luatexbase.attributes.g__tag_mc_cnt_attr)
234         ltx.__tag.func.update_mc_attributes(tex.getbox(\int_use:N #1),mc,type)
235       }
236   }
```

(*End of definition for* \tag_mc_reset_box:N. *This function is documented on page 83.*)

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

```
237 \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

TODO: check conversion, check if local/global setting is right.

```
238  \keys_define:nn { __tag / mc }
239    {
240      tag .code:n = %
241        {
242          \tl_set:Ne   \l__tag_mc_key_tag_tl { #1 }
243          \tl_gset:Ne  \g__tag_mc_key_tag_tl { #1 }
244          \lua_now:e
245            {
246              ltx.__tag.func.store_mc_data(\__tag_get_mc_abs_cnt:,"tag","#1")
247            }
248        },
249      raw .code:n =
250        {
251          \tl_put_right:Ne \l__tag_mc_key_properties_tl { #1 }
252          \lua_now:e
253            {
254              ltx.__tag.func.store_mc_data(\__tag_get_mc_abs_cnt:,"raw","#1")
255            }
256        },
257      alt .code:n      = % Alt property
258        {
259          \tl_if_empty:oF{#1}
260            {
261              \str_set_convert:Noon
262                \l__tag_tmpa_str
263                { #1 }
264                { default }
265                { utf16/hex }
266              \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
267              \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
268              \lua_now:e
269                {
270                  ltx.__tag.func.store_mc_data
271                    (
272                      \__tag_get_mc_abs_cnt:,"alt","/Alt~<\str_use:N \l__tag_tmpa_str>"
273                    )
274                }
275            }
276        },
277      lang .code:n      = % Lang property
278          {
279            \tl_if_empty:oF{#1}
280              {
281                \tl_put_right:Ne \l__tag_mc_key_properties_tl { /Lang~(#1) }
282                \lua_now:e
283                  {
284                    ltx.__tag.func.store_mc_data
285                      (
286                        \__tag_get_mc_abs_cnt:,"lang","/Lang~(#1)"
287                      )
288                  }
```

111

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

```
343            { __tag / mc }
344            { __artifact-store=\l__tag_mc_artifact_type_tl }
345        },
346     artifact .default:n    = { notype }
347   }
348
349 ⟨/luamode⟩
```

(*End of definition for* `tag` `(mc-key)` *and others. These functions are documented on page* *84.*)

Part VIII

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 {⟨structure 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 annotation. 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)

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

This is an user interface to add a Ref key to an existing structure. The target structure doesn't have to exist yet but can be addressed by label, destname or even num. ⟨*keyword*⟩ is currently either `label`, `dest` or `num`. The value is then either a label name, the name of a destination or a structure number.

2 Public keys

2.1 Keys for the structure commands

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

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

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

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

firstkid (*struct key*) If this key is used the structure is added at the left of the kids of the parent structure (if the structure is not stashed). This means that it will be the first kid of the structure (unless some later structure uses the key too).

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

alt (*struct key*) This key inserts an `/Alt` value in the dictionary of structure object. The value is handled as verbatim string and hex encoded. The value will be expanded first once. 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*) These keys handle associated files in the structure element.
AFref (*struct key*)
AFinline (*struct key*)
AFinline-o (*struct key*)
texsource (*struct key*)
mathml (*struct key*)

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

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 content 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 content 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

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

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

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

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

116

2.2 Setup keys

role/new-attribute (setup-key) role/new-attribute = {⟨name⟩}{⟨Content⟩}
newattribute (deprecated)

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
 {
 role/new-attribute =
  {TH-col}{/O /Table /Scope /Column},
 role/new-attribute =
  {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} {2026-03-19} {0.99z}
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.

```
₆ ⟨base⟩\int_new:N  \c@g__tag_struct_abs_int
₇ ⟨base⟩\int_gset:Nn \c@g__tag_struct_abs_int { 1 }
```

(*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 absolute 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 {1}
```
(*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. We create one which contains also the real tags and one with only the rolemapped tags for faster testing.

```
14 \seq_new:N    \g__tag_struct_tag_stack_seq
15 \seq_new:N    \g__tag_struct_roletag_stack_seq
16 \seq_gpush:Nn \g__tag_struct_tag_stack_seq {{Root}{StructTreeRoot}}
17 \seq_gpush:Nn \g__tag_struct_roletag_stack_seq {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.

```
18 ⟨/package⟩
19 ⟨base⟩\tl_new:N  \g__tag_struct_stack_current_tl
20 ⟨base⟩\tl_gset:Nn \g__tag_struct_stack_current_tl {\int_use:N\c@g__tag_struct_abs_int}
21 ⟨*package⟩
22 \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`.)

In luatex we will store the structure number as attribute.

```
23 \sys_if_engine_luatex:TF
24  {
25    \cs_new:Npn \__tag_struct_set_attribute:
26      {
27        \lua_now:e
28        {
29          tex.setattribute
30          (
31            "global",
32            luatexbase.attributes.g__tag_structnum_attr,
33            \g__tag_struct_stack_current_tl
34          )
35        }
36      }
37  }
38  {
39    \cs_new_eq:NN \__tag_struct_set_attribute: \prg_do_nothing:
40  }
```

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_1_prop` for the root and `\g_@@_struct_N_prop`, $N \geq 2$ 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)

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.

```
41  \seq_const_from_clist:Nn \c__tag_struct_StructTreeRoot_entries_seq
42    {%p. 857/858
43      Type,               % always /StructTreeRoot
44      K,                  % kid, dictionary or array of dictionaries
45      IDTree,             % currently unused
46      ParentTree,         % required,obj ref to the parent tree
47      ParentTreeNextKey,  % optional
48      RoleMap,
49      ClassMap,
50      Namespaces,
51      AF                  %pdf 2.0
52    }
53
54  \seq_const_from_clist:Nn \c__tag_struct_StructElem_entries_seq
55    {%p 858 f
56      Type,               %always /StructElem
57      S,                  %tag/type
58      P,                  %parent
59      ID,                 %optional
60      Ref,                %optional, pdf 2.0 Use?
61      Pg,                 %obj num of starting page, optional
62      K,                  %kids
63      A,                  %attributes, probably unused
64      C,                  %class ""
65      %R,                 %attribute revision number, irrelevant for us as we
66                          % don't update/change existing PDF and (probably)
67                          % deprecated in PDF 2.0
68      T,                  %title, value in () or <>
69      Lang,               %language
70      Alt,                % value in () or <>
71      E,                  % abbreviation
72      ActualText,
73      AF,                 %pdf 2.0, array of dict, associated files
74      NS,                 %pdf 2.0, dict, namespace
75      PhoneticAlphabet,   %pdf 2.0
76      Phoneme             %pdf 2.0
77    }
```

119

(*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`
`\l__tag_struct_parenttag_tl`
`\l__tag_struct_parenttag_NS_tl`

Use by the tag key to store the tag and the namespace. The `roletag` variables will hold locally rolemapping info needed for the parent-child checks. The `parenttag` variables allow to set the target role of the parent of stashed structures.

```
78 \tl_new:N \g__tag_struct_tag_tl
79 \tl_new:N \g__tag_struct_tag_NS_tl
80 \tl_new:N \l__tag_struct_roletag_tl
81 \tl_new:N \l__tag_struct_roletag_NS_tl
82 \tl_new:N \l__tag_struct_parenttag_tl
83 \tl_set:Nn \l__tag_struct_parenttag_tl {STASHED}
84 \tl_new:N \l__tag_struct_parenttag_NS_tl
85 \tl_set:Nn \l__tag_struct_parenttag_NS_tl {latex}
```

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

`\g__tag_struct_label_num_prop`

This will hold for every structure label the associated structure number. The prop will allow to fill the /Ref key directly at the first compilation if the ref key is used.

```
86 \prop_new_linked:N \g__tag_struct_label_num_prop
```

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

`\l__tag_struct_elem_stash_bool`

This will keep track of the stash status

```
87 \bool_new:N \l__tag_struct_elem_stash_bool
```

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

`\l__tag_struct_addkid_tl`

This decides if a structure kid is added at the left or right of the parent. The default is `right`.

```
88 \tl_new:N  \l__tag_struct_addkid_tl
89 \tl_set:Nn \l__tag_struct_addkid_tl {right}
```

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

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.

```
90 ⟨/package⟩
91 ⟨base⟩\prop_new_linked:N \g__tag_struct_dest_num_prop
92 ⟨*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. TODO: remove after 11/2024 release.

```
93 \prop_new_linked:N \g__tag_struct_ref_by_dest_prop
94 ⟨/package⟩
```

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

4 Commands

__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:

```
95  ⟨*package | debug⟩
96  ⟨package⟩\cs_new_protected:Npn \__tag_struct_prop_gput:nnn #1 #2 #3
97  ⟨debug⟩\cs_set_protected:Npn \__tag_struct_prop_gput:nnn #1 #2 #3
98    {
99      \__tag_prop_gput:cnn
100          { g__tag_struct_#1_prop }{#2}{#3}
101  ⟨debug⟩\prop_gput:cnn { g__tag_struct_debug_#1_prop } {#2} {#3}
102    }
103  \cs_generate_variant:Nn \__tag_struct_prop_gput:nnn {onn,nne,nee,nno}
104  ⟨/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/1 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.

```
105  ⟨*package⟩
106  \tl_gset:Nn \g__tag_struct_stack_current_tl {1}
```

__tag_pdf_name_e:n

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

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

g__tag_struct_1_prop
g__tag_struct_kids_1_seq

```
109  ⟨*package⟩
110  \__tag_prop_new:c { g__tag_struct_1_prop }
111  \__tag_seq_new:c  { g__tag_struct_kids_1_seq }
112
113  \__tag_struct_prop_gput:nne
114    { 1 }
115    { Type }
116    { \pdf_name_from_unicode_e:n {StructTreeRoot} }
117
118  \__tag_struct_prop_gput:nne
119    { 1 }
120    { S }
121    { \pdf_name_from_unicode_e:n {StructTreeRoot} }
122
123  \__tag_struct_prop_gput:nne
124    { 1 }
125    { tag }
126    { {StructTreeRoot}{pdf} }
127
128  \__tag_struct_prop_gput:nne
```

```
129    { 1 }
130    { rolemap }
131    { {StructTreeRoot}{pdf} }

132

133  \__tag_struct_prop_gput:nne
134    { 1 }
135    { parentrole }
136    { {StructTreeRoot}{pdf} }

137
```

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

```
138  \pdf_version_compare:NnF < {2.0}
139  {
140    \__tag_struct_prop_gput:nne
141    { 1 }
142    { Namespaces }
143    { \pdf_object_ref:n { __tag/tree/namespaces } }
144  }
145  ⟨/package⟩
```

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

```
146  ⟨debug⟩\prop_new:c { g__tag_struct_debug_1_prop }
147  ⟨debug⟩\seq_new:c  { g__tag_struct_debug_kids_1_seq }
148  ⟨debug⟩\prop_gset_eq:cc { g__tag_struct_debug_1_prop }{ g__tag_struct_1_prop }
149  ⟨debug⟩\prop_gremove:cn { g__tag_struct_debug_1_prop }{Namespaces}
```
(*End of definition for* g__tag_struct_1_prop *and* g__tag_struct_kids_1_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

```
150  ⟨*package⟩
151  \cs_new:Npn \__tag_struct_get_id:n #1 %#1=struct num
152    {
153      (
154        ID.
155        \prg_replicate:nn
156        { \int_abs:n{\g__tag_tree_id_pad_int - \tl_count:e { \int_to_arabic:n { #1 } }} }
157        { 0 }
158        \int_to_arabic:n { #1 }
159      )
160    }
```
(*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.

```
161  \pdf_version_compare:NnTF < {2.0}
162  {
163    \cs_new_protected:Npn \__tag_struct_set_tag_info:nnn #1 #2 #3
```

```
164    %#1 structure number, #2 tag, #3 NS
165    {
166      \__tag_struct_prop_gput:nne
167        { #1 }
168        { S }
169        { \pdf_name_from_unicode_e:n {#2}  } %
170      \__tag_struct_prop_gput:nnn
171        { #1 }
172        { tag }
173        { {#2} {} }
174    }
175  }
176  {
177    \cs_new_protected:Npn \__tag_struct_set_tag_info:nnn #1 #2 #3
178      {
179        \__tag_struct_prop_gput:nne
180          { #1 }
181          { S }
182          { \pdf_name_from_unicode_e:n {#2} } %
183        \prop_get:NnNT \g__tag_role_NS_prop {#3} \l__tag_get_tmpc_tl
184          {
185            \__tag_struct_prop_gput:nne
186              { #1 }
187              { NS }
188              { \l__tag_get_tmpc_tl } %
189          }
190        \__tag_struct_prop_gput:nnn
191          { #1 }
192          { tag }
193          { {#2} {#3} }
194      }
195  }
196  \cs_generate_variant:Nn \__tag_struct_set_tag_info:nnn {eoo}
```
(*End of definition for* __tag_struct_set_tag_info:nnn.)

__tag_struct_get_role:nnNN We also need a way to get the tag info needed for parent child check from parent structures. The tag info is stored as the value of the rolemap key, but for "transparent" structures we also have to look into parentrole key.

```
197  \cs_new_protected:Npn \__tag_struct_get_role:nnNN #1 #2 #3 #4
198    %#1 :struct num,
199    %#2 :rolemap or parentrole
200    %#3 :tlvar for tag (rolemapped)
201    %#4 :tlvar for NS  (rolemapped, so standard or empty or UNKNOWN)
202    {
203      \prop_get:cnNTF
204        { g__tag_struct_#1_prop }
205        { #2 }
206        \l__tag_get_tmpc_tl
207        {
208          \tl_set:Ne #3{\exp_last_unbraced:No\use_i:nn  { \l__tag_get_tmpc_tl }}
209          \tl_set:Ne #4{\exp_last_unbraced:No\use_ii:nn { \l__tag_get_tmpc_tl }}
210        }
211        {
212          \tl_clear:N#3
```

```
213              \tl_clear:N#4
214            }
215        }
216  \cs_generate_variant:Nn\__tag_struct_get_role:nnNN {enNN}
```

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

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!

```
217  \cs_new:Npn \__tag_struct_mcid_dict:n #1 %#1 MCID absnum
218    {
219      <<
220      /Type \c_space_tl /MCR \c_space_tl
221      /Pg
222        \c_space_tl
223      \pdf_pageobject_ref:n { \property_ref:enn{mcid-#1}{tagabspage}{1} }
224       /MCID \c_space_tl \property_ref:enn{mcid-#1}{tagmcid}{1}
225      >>
226    }
227  ⟨/package⟩
228  ⟨*package | debug⟩
229  ⟨package⟩\cs_new_protected:Npn \__tag_struct_kid_mc_gput_right:nn #1 #2
230  ⟨debug⟩\cs_set_protected:Npn \__tag_struct_kid_mc_gput_right:nn #1 #2
231  %#1 structure num, #2 MCID absnum%
232    {
233      \__tag_seq_gput_right:ce
234        { g__tag_struct_kids_#1_seq }
235        {
236          \__tag_struct_mcid_dict:n {#2}
237        }
238  ⟨debug⟩    \seq_gput_right:cn
239  ⟨debug⟩      { g__tag_struct_debug_kids_#1_seq }
240  ⟨debug⟩      {
241  ⟨debug⟩        MC~#2
242  ⟨debug⟩      }
243      \__tag_seq_gput_right:cn
244        { g__tag_struct_kids_#1_seq }
245        {
246          \prop_item:Nn \g__tag_struct_cont_mc_prop {#2}
247        }
248    }
249  ⟨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.

```
250 ⟨package⟩\cs_new_protected:Npn\__tag_struct_kid_struct_gput_right:nn #1 #2
251 ⟨debug⟩\cs_set_protected:Npn\__tag_struct_kid_struct_gput_right:nn #1 #2
252 %%#1 num of parent struct, #2 kid struct
253   {
254     \__tag_seq_gput_right:ce
255       { g__tag_struct_kids_#1_seq }
256       {
257         \pdf_object_ref_indexed:nn { __tag/struct }{ #2 }
258       }
259 ⟨debug⟩    \seq_gput_right:cn
260 ⟨debug⟩      { g__tag_struct_debug_kids_#1_seq }
261 ⟨debug⟩      {
262 ⟨debug⟩        Struct~#2
263 ⟨debug⟩      }
264   }
265 ⟨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_struct_gput_left:nn`
`\__tag_struct_kid_struct_gput_left:ee`

This commands adds a structure as kid one the left, so as first kid. We only need to record the object reference in the sequence.

```
266 ⟨package⟩\cs_new_protected:Npn\__tag_struct_kid_struct_gput_left:nn #1 #2
267 ⟨debug⟩\cs_set_protected:Npn\__tag_struct_kid_struct_gput_left:nn #1 #2
268 %%#1 num of parent struct, #2 kid struct
269   {
270     \__tag_seq_gput_left:ce
271       { g__tag_struct_kids_#1_seq }
272       {
273         \pdf_object_ref_indexed:nn { __tag/struct }{ #2 }
274       }
275 ⟨debug⟩    \seq_gput_left:cn
276 ⟨debug⟩      { g__tag_struct_debug_kids_#1_seq }
277 ⟨debug⟩      {
278 ⟨debug⟩        Struct~#2
279 ⟨debug⟩      }
280   }
281 ⟨package⟩\cs_generate_variant:Nn \__tag_struct_kid_struct_gput_left:nn {ee}
```
(*End of definition for* `\__tag_struct_kid_struct_gput_left:nn`.)

`\__tag_struct_kid_OBJR_gput_right:nnn`
`\__tag_struct_kid_OBJR_gput_right:eee`

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

```
282 ⟨package⟩\cs_new_protected:Npn\__tag_struct_kid_OBJR_gput_right:nnn #1 #2 #3
283 ⟨package⟩
284 ⟨package⟩
285 ⟨debug⟩\cs_set_protected:Npn\__tag_struct_kid_OBJR_gput_right:nnn #1 #2 #3
286 %%#1 num of parent struct,#2 obj reference,#3 page object reference
287   {
288     \pdf_object_unnamed_write:nn
289       { dict }
```

```
290          {
291            /Type/OBJR/Obj~#2/Pg~#3
292          }
293        \__tag_seq_gput_right:ce
294          { g__tag_struct_kids_#1_seq }
295          {
296            \pdf_object_ref_last:
297          }
298 ⟨debug⟩    \seq_gput_right:ce
299 ⟨debug⟩      { g__tag_struct_debug_kids_#1_seq }
300 ⟨debug⟩      {
301 ⟨debug⟩        OBJR~reference
302 ⟨debug⟩      }
303      }
304 ⟨/package | debug⟩
305 ⟨*package⟩
306 \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
__tag_struct_exchange_kid_command:c

In luamode it can happen that a single kid in a structure is split at a page break into two or more mcid. In this case the lua code has to convert put the dictionary of the kid into an array. See issue 13 at tagpdf repo. We exchange the dummy command for the kids to mark this case. Change 2024-03-19: don't use a regex - that is slow.

```
307 \cs_new_protected:Npn\__tag_struct_exchange_kid_command:N #1 %#1 = seq var
308   {
309     \seq_gpop_left:NN #1 \l__tag_tmpa_tl
310     \tl_replace_once:Nnn \l__tag_tmpa_tl
311      {\__tag_mc_insert_mcid_kids:n}
312      {\__tag_mc_insert_mcid_single_kids:n}
313     \seq_gput_left:No #1 { \l__tag_tmpa_tl }
314   }
315
316 \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.

```
317 \cs_new_protected:Npn \__tag_struct_fill_kid_key:n #1 %#1 is the struct num
318   {
319     \bool_if:NF \g__tag_mode_lua_bool
320       {
321         \seq_clear:N \l__tag_tmpa_seq
322         \seq_map_inline:cn { g__tag_struct_kids_#1_seq }
323          { \seq_put_right:Ne \l__tag_tmpa_seq { ##1 } }
324         %\seq_show:c { g__tag_struct_kids_#1_seq }
325         %\seq_show:N \l__tag_tmpa_seq
326         \seq_remove_all:Nn \l__tag_tmpa_seq {}
327         %\seq_show:N \l__tag_tmpa_seq
328         \seq_gset_eq:cN { g__tag_struct_kids_#1_seq } \l__tag_tmpa_seq
329       }
330
331     \int_case:nnF
332       {
```

```
333        \seq_count:c
334          {
335            g__tag_struct_kids_#1_seq
336          }
337      }
338      {
339        { 0 }
340        { } %no kids, do nothing
341        { 1 } % 1 kid, insert
342        {
343          % in this case we need a special command in
344          % luamode to get the array right. See issue #13
345          \sys_if_engine_luatex:TF
346            {
347              \__tag_struct_exchange_kid_command:c
348              {g__tag_struct_kids_#1_seq}
```

check if we get null

```
349              \tl_set:Ne\l__tag_tmpa_tl
350                {\use:e{\seq_item:cn {g__tag_struct_kids_#1_seq} {1}}}
351              \tl_if_eq:NNF\l__tag_tmpa_tl \c__tag_struct_null_tl
352                {
353                  \__tag_struct_prop_gput:nne
354                    {#1}
355                    {K}
356                    {
357                      \seq_item:cn
358                        {
359                          g__tag_struct_kids_#1_seq
360                        }
361                        {1}
362                    }
363                }
364            }
365            {
366              \__tag_struct_prop_gput:nne
367                {#1}
368                {K}
369                {
370                  \seq_item:cn
371                    {
372                      g__tag_struct_kids_#1_seq
373                    }
374                    {1}
375                }
376            }
377          } %
378      }
379      { %many kids, use an array
380        \__tag_struct_prop_gput:nne
381          {#1}
382          {K}
383          {
384            [
```

127

```
385                \seq_use:cn
386                  {
387                    g__tag_struct_kids_#1_seq
388                  }
389                  {
390                    \c_space_tl
391                  }
392              ]
393            }
394        }
395    }
396
```

(*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. This is used a lot so should be rather fast.

```
397 \cs_new_protected:Npn \__tag_struct_get_dict_content:nN #1 #2 %#1: structure num
398    {
399      \tl_clear:N #2
400      \prop_map_inline:cn { g__tag_struct_#1_prop }
401        {
```

Some keys needs the option to format the value, e.g. add brackets for an array, we also
need the option to ignore some entries in the properties.

```
402        \cs_if_exist_use:cTF {__tag_struct_format_##1:nnN}
403          {
404            {##1}{##2}#2
405          }
406          {
407            \tl_put_right:Ne #2 { \c_space_tl/##1~##2 }
408          }
409      }
410    }
```

(*End of definition for __tag_struct_get_dict_content:nN.*)

__tag_struct_format_rolemap:nnN   This three entries should not end in the PDF. Todo: check if the S/NS keys can be
__tag_struct_format_parentrole:nnN   dropped and replaced by a processing of the tag key.
__tag_struct_format_P:nnN
__tag_struct_format_tag:nnN
```
411 \cs_new:Nn\__tag_struct_format_rolemap:nnN{}
412 \cs_new:Nn\__tag_struct_format_parentrole:nnN{}
413 \cs_new:Nn\__tag_struct_format_tag:nnN{}
```
(*End of definition for __tag_struct_format_rolemap:nnN and others.*)

__tag_struct_format_parentnum:nnN   parent is a structure number and should expand to the object reference.

```
414 \cs_new_protected:Nn\__tag_struct_format_parentnum:nnN
415    {
416      \tl_put_right:Ne #3 { ~/P~\pdf_object_ref_indexed:nn { __tag/struct} { #2 } }
417    }
```

(*End of definition for __tag_struct_format_parentnum:nnN.*)

`\__tag_struct_format_Ref:nnN` Ref is an array, we store values as a clist of commands that must be executed here, the formatting has to add also brackets.

```
418 \cs_new_protected:Nn\__tag_struct_format_Ref:nnN
419   {
420     \tl_put_right:Nn #3 { ~/#1~[ } %]
421     \clist_map_inline:nn{ #2 }
422       {
423         ##1 #3
424       }
425     \tl_put_right:Nn #3
426       { %[
427         \c_space_tl]
428       }
429   }
```

(*End of definition for* `\__tag_struct_format_Ref:nnN`.)

`\__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.

```
430 \cs_new_protected:Npn \__tag_struct_write_obj:n #1 % #1 is the struct num
431   {
432     \prop_if_exist:cTF { g__tag_struct_#1_prop }
433       {
```

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.

```
434         \prop_get:cnNF { g__tag_struct_#1_prop } {parentnum}\l__tag_tmpb_tl
435           {
436 %            \prop_gput:cne { g__tag_struct_#1_prop } {P}
437 %              {\pdf_object_ref_indexed:nn { __tag/struct }{1}}
438             \prop_gput:cne { g__tag_struct_#1_prop } {parentnum}{2}
439             \prop_gput:cne { g__tag_struct_#1_prop } {S}{/Artifact}
440             \seq_if_empty:cF {g__tag_struct_kids_#1_seq}
441               {
442                 \msg_warning:nnee
443                   {tag}
444                   {struct-orphan}
445                   { #1 }
446                   {\seq_count:c{g__tag_struct_kids_#1_seq}}
447               }
448           }
449         \__tag_struct_fill_kid_key:n { #1 }
450         \__tag_struct_get_dict_content:nN { #1 } \l__tag_tmpa_tl
451         \pdf_object_write_indexed:nnne
452           { __tag/struct }{ #1 }
453           {dict}
454           {
455             \l__tag_tmpa_tl\c_space_tl
456             /ID~\__tag_struct_get_id:n{#1}
457           }
458
459       }
460       {
461         \msg_error:nnn { tag } { struct-no-objnum } { #1}
```

129

```
462              }
463          }
```
(*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:
```

```
464 \cs_new_protected:Npn \__tag_struct_insert_annot:nn #1 #2
465 %#1 object reference to the annotation/xform
466 %#2 structparent number
467  {
468     \bool_if:NT \g__tag_active_struct_bool
469       {
470         %get the number of the parent structure:
471         \seq_get:NNF
472           \g__tag_struct_stack_seq
473           \l__tag_struct_stack_parent_tmpa_tl
474           {
475             \msg_error:nn { tag } { struct-faulty-nesting }
476           }
477         %put the obj number of the annot in the kid entry, this also creates
478         %the OBJR object
479         \__tag_property_record:nn {@tag@objr@page@#2 }{ tagabspage }
480         \__tag_struct_kid_OBJR_gput_right:eee
481           {
482             \l__tag_struct_stack_parent_tmpa_tl
483           }
484           {
485             #1 %
486           }
487           {
488             \pdf_pageobject_ref:n
489               { \property_ref:nnn {@tag@objr@page@#2 }{ tagabspage }{1} }
490           }
```

130

```
491      % add the parent obj number to the parent tree:
492      % the command always expands its arguments!
493      \__tag_parenttree_add_objr:nn
494        {
495          #2
496        }
497        {
498          \pdf_object_ref_indexed:nn
499            { __tag/struct }{ \l__tag_struct_stack_parent_tmpa_tl }
500        }
501      % increase the int:
502      \int_gincr:N \c@g__tag_parenttree_obj_int
503    }
504  }
```

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

__tag_struct_insert_annot_shipout:nnn This command is similar to the previous one but is meant to be used at shipout (currently only sensible for luatex). To move the OBJR into the right structure it has to get the structure number additionally as argument. But as it is used at shipout it doesn't need a label to get the page reference but can use \g_shipout_readonly_int. It does *not* increase the parenttree integer (timing is wrong in lua), instead code using the command has to do it. See the lua code.

```
505  \cs_new_protected:Npn\__tag_struct_insert_annot_shipout:nnn #1#2#3
506  % #1 structnum, #2 object reference, #3 StructParentNum
507  {
508    \__tag_struct_kid_OBJR_gput_right:eee
509      {
510        #1
511      }
512      {
513        #2
514      }
515      {
516        \pdf_pageobject_ref:n
517          { \int_use:N \g_shipout_readonly_int } %
518      }
519    % add the parent obj number to the parent tree:
520    % the command always expands its arguments!
521    \__tag_parenttree_add_objr:nn
522      {
523        #3
524      }
525      {
526        \pdf_object_ref_indexed:nn
527          { __tag/struct }{ #1 }
528      }
529  }
```

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

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

```
530  \cs_new:Npn \__tag_get_data_struct_tag:
531    {
```

```
532     \exp_args:Ne
533     \tl_tail:n
534       {
535         \prop_item:cn {g__tag_struct_\g__tag_struct_stack_current_tl _prop}{S}
536       }
537   }
```

(*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.

```
538 \cs_new:Npn \__tag_get_data_struct_id:
539   {
540     \__tag_struct_get_id:n {\g__tag_struct_stack_current_tl}
541   }
542 ⟨/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

```
543 ⟨*base⟩
544 \cs_new:Npn \__tag_get_data_struct_num:
545   {
546     \g__tag_struct_stack_current_tl
547   }
548 ⟨/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.

```
549 ⟨*base⟩
550 \cs_new:Npn \__tag_get_data_struct_counter:
551   {
552     \int_use:N \c@g__tag_struct_abs_int
553   }
554 ⟨/base⟩
```

(*End of definition for* __tag_get_data_struct_counter:.)

4.6 Commands for the parent-child checks

__tag_struct_check_parent_child_aux:nnnnN

```
555 ⟨*package⟩
556 \cs_new_protected:Npn \__tag_struct_check_parent_child_aux:nnnnN #1#2#3#4#5
557   {
558 % #1 structure number of parent
559 % #2 key to use to retrieve role of parent (either rolemap or parentrole field)
560 % #3 structure number of parent
561 % #4 key to use to retrieve role of child (either rolemap or parentrole field)
562 % #5 tl for return value
```

132

get parent rolemap

```
563    \__tag_struct_get_role:nnNN
564      {#1}
565      {#2}
566      \l__tag_get_parent_tmpa_tl
567      \l__tag_get_parent_tmpb_tl
```

get child rolemap

```
568    \__tag_struct_get_role:nnNN
569      {#3}
570      {#4}
571      \l__tag_get_child_tmpa_tl
572      \l__tag_get_child_tmpb_tl
```

check

```
573    \__tag_role_check_parent_child:ooooN
574      { \l__tag_get_parent_tmpa_tl } % rolemapped from above
575      { \l__tag_get_parent_tmpb_tl } % rolemapped from above
576      { \l__tag_get_child_tmpa_tl } %
577      { \l__tag_get_child_tmpb_tl } %
578      #5
579    }
```

(*End of definition for* __tag_struct_check_parent_child_aux:nnnnN.)

__tag_struct_check_parent_child:nn When comparing the relation between structures we use the structure numbers.

```
580  \cs_new_protected:Npn \__tag_struct_check_parent_child:nn #1 #2
581  % #1 structure number of parent
582  % #2 structure number of child. %
583  % This assumes that the fields rolemap/parentrole has already been filled.
584    {
```

This records if logging is on

```
585      \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
586        {
587          \prop_get:cnN{g__tag_struct_#1_prop}{tag}\l__tag_get_parent_tmpa_tl
588          \prop_get:cnN{g__tag_struct_#2_prop}{tag}\l__tag_get_parent_tmpb_tl
589          \msg_note:nnee
590          { tag }
591          { role-parent-child-check }
592          {
593            \quark_if_no_value:NTF \l__tag_get_parent_tmpa_tl
594            {??}
595            {
596              \exp_last_unbraced:No\use_ii:nn
597              { \l__tag_get_parent_tmpa_tl }
598              :
599              \exp_last_unbraced:No\use_i:nn
600              { \l__tag_get_parent_tmpa_tl }
601            }
602          }
603          {
604            \quark_if_no_value:NTF \l__tag_get_parent_tmpb_tl
605            {??}
606            {
607              \exp_last_unbraced:No\use_ii:nn
```

```
608            { \l__tag_get_parent_tmpb_tl }
609            :
610            \exp_last_unbraced:No\use_i:nn
611            { \l__tag_get_parent_tmpb_tl }
612          }
613        }
614      }
615    \__tag_struct_check_parent_child_aux:nnnnN
616      {#1}
617      {rolemap}
618      {#2}
619      {rolemap}
620      \l__tag_parent_child_check_tl
```

if the return value is 7 we have to check against the parentrole field.

```
621    \int_compare:nNnT {\l__tag_parent_child_check_tl} = { \c__tag_role_rule_checkparent_tl }
622      {
623        \__tag_struct_check_parent_child_aux:nnnnN
624          {#1}
625          {parentrole}
626          {#2}
627          {rolemap}
628          \l__tag_parent_child_check_tl
629      }
630    \__tag_check_struct_forbidden_parent_child:onn
631      {\l__tag_parent_child_check_tl}
632      {#1}
633      {#2}
634  }
635 \cs_generate_variant:Nn \__tag_struct_check_parent_child:nn {oo}
```

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

__tag_struct_use_check_parent_child:nn A similar command is needed if a structure is stashed and used. The child can be - a normal tag (e.g. H1) then rolemap = parentrole = H1pdf2 and we should test rolemap (parent) and rolemap (child) if = 7 parentrole (parent) and rolemap (child) That is the normal check above.
- Part/Div/Nonstruct then rolemap = Partpdf2 and parentrole = STASHEDlatex or target parentNS
If parentrole =STASHED we can't test if the child fits here. If parentrole is not STASHED, then would should test if target parent= rolemap (parent) or parentrole (parent) and if yet then test rolemap (child) against rolemap (parent) and if =7 rolemap(child) against parentrole(parent). that is again the normal check.

```
636 \cs_new_protected:Npn \__tag_struct_use_check_parent_child:nn #1 #2
637 % #1 structure number of parent
638 % #2 structure number of child. %
639  {
640    \__tag_struct_get_role:enNN
641      {#2}
642      {rolemap}
643      \l__tag_get_child_tmpa_tl
644      \l__tag_get_child_tmpb_tl
645    \str_case:onTF { \l__tag_get_child_tmpa_tl }
646      {
```

134

```
647        {Part} {}
648        {Div}  {}
649        {NonStruct} {}
650      }
651    { %child=Part etc
652      \__tag_struct_get_role:enNN
653       {#2}
654       {parentrole}
655       \l__tag_get_child_tmpa_tl
656       \l__tag_get_child_tmpb_tl
657      \str_if_eq:noTF
658       {STASHED}{\l__tag_get_child_tmpa_tl}
659       {
660          % warn about unknown relationship
661       }
662       {
663          % test if
664          \__tag_struct_get_role:enNN
665           {#1}
666           {parentrole}
667           \l__tag_get_parent_tmpa_tl
668           \l__tag_get_parent_tmpb_tl
669          \tl_if_eq:NNTF\l__tag_get_parent_tmpa_tl \l__tag_get_child_tmpa_tl
670          {
671             \__tag_struct_check_parent_child:nn {#1}{#2}
672          }
673          {
674             %warn that parent-tag was misused.
675          }
676       }
677    }
678    {
679       %child not Part etc, normal parent child test.
680       \__tag_struct_check_parent_child:nn {#1}{#2}
681    }
682  }
683 \cs_generate_variant:Nn  { \__tag_struct_use_check_parent_child:nn }{oo}
```
(*End of definition for* `\__tag_struct_use_check_parent_child:nn`.)

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.

This socket is used by the tag key. It allows to switch between the latex-tabs and the standard tags.

```
684 \socket_new:nn { tag/struct/tag }{1}
685 \socket_new_plug:nnn { tag/struct/tag }{ latex-tags }
686 {
687    \prop_get:NeNTF \g__tag_role_tags_NS_prop {#1} \l__tag_tmp_unused_tl
688    {
689       \seq_set_split:Nne \l__tag_tmpa_seq { / }
690         {#1/\l__tag_tmp_unused_tl}
```

```
691        }
692      {
693        \seq_set_split:Nne \l__tag_tmpa_seq { / }
694          {#1/}
695      }
696    \tl_gset:Ne \g__tag_struct_tag_tl   { \seq_item:Nn\l__tag_tmpa_seq {1} }
697    \tl_gset:Ne \g__tag_struct_tag_NS_tl{ \seq_item:Nn\l__tag_tmpa_seq {2} }
698    \__tag_check_structure_tag:N \g__tag_struct_tag_tl
699  }
700
701  \socket_new_plug:nnn { tag/struct/tag }{ pdf-tags }
702  {
703    \prop_get:NeNTF \g__tag_role_tags_NS_prop {#1} \l__tag_tmp_unused_tl
704      {
705        \seq_set_split:Nne \l__tag_tmpa_seq { / }
706          {#1/\l__tag_tmp_unused_tl}
707      }
708      {
709        \seq_set_split:Nne \l__tag_tmpa_seq { / }
710          {#1/}
711      }
712    \tl_gset:Ne \g__tag_struct_tag_tl   { \seq_item:Nn\l__tag_tmpa_seq {1} }
713    \tl_gset:Ne \g__tag_struct_tag_NS_tl{ \seq_item:Nn\l__tag_tmpa_seq {2} }
714    \__tag_role_get:ooNN
715      { \g__tag_struct_tag_tl }
716      { \g__tag_struct_tag_NS_tl}
717      \l__tag_tmpa_tl
718      \l__tag_tmpb_tl
719    \tl_gset:Ne \g__tag_struct_tag_tl {\l__tag_tmpa_tl}
720    \tl_gset:Ne \g__tag_struct_tag_NS_tl{\l__tag_tmpb_tl}
721    \__tag_check_structure_tag:N \g__tag_struct_tag_tl
722  }
723  \socket_assign_plug:nn { tag/struct/tag } {latex-tags}
```

label (*struct key*)
stash (*struct key*)
parent (*struct key*)
firstkid (*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*)
phoneme (*struct key*)

```
724  \keys_define:nn { __tag / struct }
725    {
726      label .code:n        =
727        {
728          \prop_gput:Nee\g__tag_struct_label_num_prop
729            {#1}{\int_use:N \c@g__tag_struct_abs_int}
730          \__tag_property_record:eo
731            {tagpdfstruct-#1}
732            { \c__tag_property_struct_clist }
733        },
734      stash .bool_set:N    = \l__tag_struct_elem_stash_bool,
735      parent .code:n       =
736        {
737          \bool_lazy_and:nnTF
738            {
739              \prop_if_exist_p:c { g__tag_struct_\int_eval:n {#1}_prop }
740            }
741            {
742              \int_compare_p:nNn {#1}<{\c@g__tag_struct_abs_int}
```

136

```
743              }
744              { \tl_set:Ne \l__tag_struct_stack_parent_tmpa_tl { \int_eval:n {#1} } }
745              {
746                \msg_warning:nnee { tag } { struct-unknown }
747                  { \int_eval:n {#1} }
748                  { parent~key~ignored }
749              }
750          },
751      parent .default:n    = {-1},
752      parent-tag .code:n =
753        {
754            \prop_get:NeNTF \g__tag_role_tags_NS_prop {#1} \l__tag_tmp_unused_tl
755              {
756               \seq_set_split:Nne \l__tag_tmpa_seq { / }
757                 {#1/\l__tag_tmp_unused_tl}
758              }
759              {
760                \seq_set_split:Nne \l__tag_tmpa_seq { / }
761                  {#1/}
762              }
763            \tl_set:Ne \l__tag_struct_parenttag_tl    { \seq_item:Nn\l__tag_tmpa_seq {1} }
764            \tl_set:Ne \l__tag_struct_parenttag_NS_tl{ \seq_item:Nn\l__tag_tmpa_seq {2} }
765            \__tag_role_get:ooNN
766              { \l__tag_struct_parenttag_tl }
767              { \l__tag_struct_parenttag_NS_tl}
768              \l__tag_tmpa_tl
769              \l__tag_tmpb_tl
770            \tl_set:No \l__tag_struct_parenttag_tl   {\l__tag_tmpa_tl}
771            \tl_set:No \l__tag_struct_parenttag_NS_tl{\l__tag_tmpb_tl}
772            \__tag_check_structure_tag:N \l__tag_struct_parenttag_tl
773        },
774  firstkid .code:n = { \tl_set:Nn \l__tag_struct_addkid_tl {left} },
775      tag    .code:n       = % S property
776        {
777            \socket_use:nn { tag/struct/tag }{#1}
778        },
779      title .code:n        = % T property
780        {
781          \str_set_convert:Nnnn
782            \l__tag_tmpa_str
783            { #1 }
784            { default }
785            { utf16/hex }
786          \__tag_struct_prop_gput:nne
787            { \int_use:N \c@g__tag_struct_abs_int }
788            { T }
789            { <\l__tag_tmpa_str> }
790        },
791      title-o .code:n       = % T property
792        {
793          \str_set_convert:Nonn
794            \l__tag_tmpa_str
795            { #1 }
796            { default }
```

```
797            { utf16/hex }
798          \__tag_struct_prop_gput:nne
799            { \int_use:N \c@g__tag_struct_abs_int }
800            { T }
801            { <\l__tag_tmpa_str> }
802        },
803      alt .code:n      = % Alt property
804        {
805          \tl_if_empty:oF{#1}
806            {
807            \str_set_convert:Noon
808              \l__tag_tmpa_str
809              { #1 }
810              { default }
811              { utf16/hex }
812            \__tag_struct_prop_gput:nne
813              { \int_use:N \c@g__tag_struct_abs_int }
814              { Alt }
815              { <\l__tag_tmpa_str> }
816            }
817        },
818      alttext .meta:n = {alt=#1},
819      actualtext .code:n  = % ActualText property
820        {
821          \tl_if_empty:oF{#1}
822            {
823            \str_set_convert:Noon
824              \l__tag_tmpa_str
825              { #1 }
826              { default }
827              { utf16/hex }
828            \__tag_struct_prop_gput:nne
829              { \int_use:N \c@g__tag_struct_abs_int }
830              { ActualText }
831              { <\l__tag_tmpa_str>}
832            }
833        },
834       phoneme .code:n  = % Phoneme property
835        {
836          \tl_if_empty:oF{#1}
837            {
838            \str_set_convert:Noon
839              \l__tag_tmpa_str
840              { #1 }
841              { default }
842              { utf16/hex }
843            \__tag_struct_prop_gput:nne
844              { \int_use:N \c@g__tag_struct_abs_int }
845              { Phoneme }
846              { <\l__tag_tmpa_str>}
847            }
848        },
849      lang .code:n        = % Lang property
850        {
```

```
851        \__tag_struct_prop_gput:nne
852          { \int_use:N \c@g__tag_struct_abs_int }
853          { Lang }
854          { (#1) }
855      },
856    }
```

Ref is rather special as it values are often known only at the end of the document. It therefore stores it values as clist of commands which are executed at the end of the document, when the structure elements are written.

this commands are helper commands that are stored as clist in the Ref key of a structure. They are executed when the structure elements are written in __tag_struct_write_-obj. They are used in __tag_struct_format_Ref. They allow to add a Ref by object reference, label, destname and structure number

```
857  \cs_new_protected:Npn \__tag_struct_Ref_obj:nN #1 #2 %#1 a object reference
858    {
859      \tl_put_right:Ne#2
860        {
861          \c_space_tl#1
862        }
863    }
864
865  \cs_new_protected:Npn \__tag_struct_Ref_label:nN #1 #2 %#1 a label
866    {
867      \prop_get:NnNTF \g__tag_struct_label_num_prop {#1} \l__tag_tmpb_tl
868        {
869          \tl_put_right:Ne#2
870            {
871              \c_space_tl\tag_struct_object_ref:e{ \l__tag_tmpb_tl }
872            }
873        }
874        {
875          \msg_warning:nnn {tag}{struct-Ref-unknown}{Label~'#1'}
876        }
877    }
878  \cs_new_protected:Npn \__tag_struct_Ref_dest:nN #1 #2 %#1 a dest name
879    {
880      \prop_get:NnNTF \g__tag_struct_dest_num_prop {#1} \l__tag_tmpb_tl
881        {
882          \tl_put_right:Ne#2
883            {
884              \c_space_tl\tag_struct_object_ref:e{ \l__tag_tmpb_tl }
885            }
886        }
887        {
888          \msg_warning:nnn {tag}{struct-Ref-unknown}{Destination~'#1'}
889        }
890    }
891  \cs_new_protected:Npn \__tag_struct_Ref_num:nN #1 #2 %#1 a structure number
892    {
893      \tl_put_right:Ne#2
894        {
895          \c_space_tl\tag_struct_object_ref:e{ #1 }
```

```
896          }
897      }
898
```

(*End of definition for* `\__tag_struct_Ref_obj:nN` *and others.*)

<div style="margin-left: 2em;">

ref (*struct key*)
E (*struct key*)

```
899  \keys_define:nn { __tag / struct }
900    {
901      ref .code:n        = % ref property
902        {
903          \clist_map_inline:on {#1}
904            {
905              \tag_struct_gput:nne
906                {\int_use:N \c@g__tag_struct_abs_int}{ref_label}{ ##1 }
907            }
908        },
909      E .code:n          = % E property
910        {
911          \str_set_convert:Nnon
912            \l__tag_tmpa_str
913            { #1 }
914            { default }
915            { utf16/hex }
916          \__tag_struct_prop_gput:nne
917            { \int_use:N \c@g__tag_struct_abs_int }
918            { E }
919            { <\l__tag_tmpa_str> }
920        },
921    }
```

</div>

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

`\g__tag_struct_AFobj_int` This variable is used to number the AF-object names

```
922  \int_new:N\g__tag_struct_AFobj_int
```

```
923  \cs_generate_variant:Nn \pdffile_embed_stream:nnN {neN}
924  \cs_new_protected:Npn \__tag_struct_add_inline_AF:nn #1 #2
925  % #1 content, #2 extension
926    {
927      \tl_if_empty:nF{#1}
928        {
929          \group_begin:
930          \int_gincr:N \g__tag_struct_AFobj_int
931          \pdffile_embed_stream:neN
932            {#1}
933            {tag-AFfile\int_use:N\g__tag_struct_AFobj_int.#2}
934            \l__tag_tmpa_tl
935          \__tag_struct_add_AF:ee
936            { \int_use:N \c@g__tag_struct_abs_int }
937            { \l__tag_tmpa_tl }
```

```
938        \__tag_struct_prop_gput:nne
939          { \int_use:N \c@g__tag_struct_abs_int }
940          { AF }
941          {
942            [
943              \tl_use:c
944                { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
945            ]
946          }
947        \group_end:
948      }
949  }
950
951  \cs_generate_variant:Nn \__tag_struct_add_inline_AF:nn {on}
952  \cs_new_protected:Npn \__tag_struct_add_AF:nn #1 #2
953  % #1 struct num #2 object reference
954    {
955      \tl_if_exist:cTF
956        {
957          g__tag_struct_#1_AF_tl
958        }
959        {
960          \tl_gput_right:ce
961            { g__tag_struct_#1_AF_tl }
962            {  \c_space_tl #2 }
963        }
964        {
965          \tl_new:c
966            { g__tag_struct_#1_AF_tl }
967          \tl_gset:ce
968            { g__tag_struct_#1_AF_tl }
969            { #2 }
970        }
971    }
972  \cs_generate_variant:Nn \__tag_struct_add_AF:nn {en,ee}
973  \keys_define:nn { __tag / struct }
974  {
975    AF .code:n         = % AF property
976      {
977        \pdf_object_if_exist:eTF {#1}
978          {
979            \__tag_struct_add_AF:ee
980            { \int_use:N \c@g__tag_struct_abs_int }{\pdf_object_ref:e {#1}}
981            \__tag_struct_prop_gput:nne
982            { \int_use:N \c@g__tag_struct_abs_int }
983            { AF }
984            {
985              [
986                \tl_use:c
987                  { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
988              ]
989            }
990          }
991          {
```

```
992                      % message?
993                  }
994             },
995      AFref .code:n        = % AF property
996          {
997          \tl_if_empty:eF {#1}
998            {
999              \__tag_struct_add_AF:ee { \int_use:N \c@g__tag_struct_abs_int }{#1}
1000             \__tag_struct_prop_gput:nne
1001               { \int_use:N \c@g__tag_struct_abs_int }
1002               { AF }
1003               {
1004                 [
1005                   \tl_use:c
1006                   { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
1007                 ]
1008               }
1009           }
1010         },
1011    ,AFinline .code:n =
1012       {
1013          \__tag_struct_add_inline_AF:nn {#1}{txt}
1014       }
1015    ,AFinline-o .code:n =
1016       {
1017          \__tag_struct_add_inline_AF:on {#1}{txt}
1018       }
1019    ,texsource .code:n =
1020       {
1021          \group_begin:
1022          \pdfdict_put:nnn { l_pdffile/Filespec } {Desc}{(TeX~source)}
1023          \pdfdict_put:nnn { l_pdffile/Filespec }{AFRelationship} { /Source }
1024          \__tag_struct_add_inline_AF:on {#1}{tex}
1025          \group_end:
1026       }
1027    ,mathml .code:n =
1028        {
1029          \group_begin:
1030          \pdfdict_put:nnn { l_pdffile/Filespec } {Desc}{(mathml~representation)}
1031          \pdfdict_put:nnn { l_pdffile/Filespec }{AFRelationship} { /Supplement }
1032          \pdfdict_put:nne { l_pdffile }{Subtype}
1033             { \pdf_name_from_unicode_e:n{application/mathml+xml} }
1034          \__tag_struct_add_inline_AF:on {#1}{xml}
1035          \group_end:
1036        }
1037  }
```

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

```
1038 \keys_define:nn { __tag / setup }
1039  {
1040     root-AF .code:n =
1041       {
1042           \pdf_object_if_exist:nTF {#1}
```

```
1043                {
1044                  \__tag_struct_add_AF:ee { 1 }{\pdf_object_ref:n {#1}}
1045                  \__tag_struct_prop_gput:nne
1046                  { 1 }
1047                  { AF }
1048                  {
1049                    [
1050                      \tl_use:c
1051                        { g__tag_struct_1_AF_tl }
1052                    ]
1053                  }
1054                }
1055                {

1057                }
1058          },
1059      }
```

ot-supplemental-file (*setup key*) This key allows to add a file as root-AF with relationship Supplement. This is typically need to add a css or an html

```
1060  \keys_define:nn { __tag / setup }
1061    {
1062      root-supplemental-file .code:n =
1063        {
1064          \group_begin:
1065          \pdfdict_put:nnn {l_pdffile/Filespec} {AFRelationship}{/Supplement}
1066          \int_gincr:N \g__tag_unique_cnt_int
1067          \pdffile_embed_file:eee
1068          {#1}
1069          {#1}
1070          {__tag_latex_css_\int_use:N\g__tag_unique_cnt_int}
1071          \keys_set:nn
1072          {__tag / setup}
1073          {root-AF={__tag_latex_css_\int_use:N\g__tag_unique_cnt_int}}
1074          \group_end:
1075        }
1076    }
```

og-supplemental-file (*setup key*) This key allows to add a file as AF with relationship Supplement to the Catalog. This is typically need to add a css or an html.

```
1077  \keys_define:nn { __tag / setup }
1078    {
1079      catalog-supplemental-file .code:n =
1080        {
1081          \group_begin:
1082          \pdfdict_put:nnn {l_pdffile/Filespec} {AFRelationship}{/Supplement}
1083          \int_gincr:N \g__tag_unique_cnt_int
1084          \pdffile_embed_file:eee
1085          {#1}
1086          {#1}
1087          {__tag_latex_css_\int_use:N\g__tag_unique_cnt_int}
1088          \pdfmanagement_add:nne
1089          {Catalog}
1090          {AF}
```

```
1091              {\pdf_object_ref:e{__tag_latex_css_\int_use:N\g__tag_unique_cnt_int }}
1092           \group_end:
1093        }
1094    }
```

6 User commands

We allow to set a language by default

```
1095  \tl_new:N \l__tag_struct_lang_tl
1096  ⟨/package⟩
```

(*End of definition for* \g__tag_struct_AFobj_int *and* \l__tag_struct_lang_tl.)

\tag_struct_begin:n
\tag_struct_end:

```
1097  ⟨base⟩\cs_new_protected:Npn \tag_struct_begin:n #1 {\int_gincr:N \c@g__tag_struct_abs_int}
1098  ⟨base⟩\cs_new_protected:Npn \tag_struct_end:{}
1099  ⟨base⟩\cs_new_protected:Npn \tag_struct_end:n{}
1100  ⟨*package | debug⟩
1101  ⟨package⟩\cs_set_protected:Npn \tag_struct_begin:n #1 %#1 key-val
1102  ⟨debug⟩\cs_set_protected:Npn \tag_struct_begin:n #1 %#1 key-val
1103      {
1104  ⟨package⟩\__tag_check_if_active_struct:T
1105  ⟨debug⟩\__tag_check_if_active_struct:TF
1106         {
1107           \group_begin:
1108           \int_gincr:N \c@g__tag_struct_abs_int
1109           \__tag_prop_new:c  { g__tag_struct_\int_eval:n { \c@g__tag_struct_abs_int }_prop }
1110  ⟨debug⟩         \prop_new:c { g__tag_struct_debug_\int_eval:n {\c@g__tag_struct_abs_int}_prop
1111           \__tag_seq_new:c  { g__tag_struct_kids_\int_eval:n { \c@g__tag_struct_abs_int }_seq}
1112  ⟨debug⟩         \seq_new:c { g__tag_struct_debug_kids_\int_eval:n {\c@g__tag_struct_abs_int}_
1113            \pdf_object_new_indexed:nn { __tag/struct }
1114             { \c@g__tag_struct_abs_int }
1115           \__tag_struct_prop_gput:nnn
1116             { \int_use:N \c@g__tag_struct_abs_int }
1117             { Type }
1118             { /StructElem }
1119           \tl_if_empty:NF \l__tag_struct_lang_tl
1120             {
1121               \__tag_struct_prop_gput:nne
1122                 { \int_use:N \c@g__tag_struct_abs_int }
1123                 { Lang }
1124                 { (\l__tag_struct_lang_tl) }
1125             }
1126           \__tag_struct_prop_gput:nnn
1127             { \int_use:N \c@g__tag_struct_abs_int }
1128             { Type }
1129             { /StructElem }
1130
1131           \tl_set:Nn \l__tag_struct_stack_parent_tmpa_tl {-1}
1132           \keys_set:nn { __tag / struct} { #1 }
```

144

```
1133            \__tag_struct_set_tag_info:eoo
1134              { \int_use:N \c@g__tag_struct_abs_int }
1135              { \g__tag_struct_tag_tl }
1136              { \g__tag_struct_tag_NS_tl }
1137            \__tag_check_structure_has_tag:n { \int_use:N \c@g__tag_struct_abs_int }
```

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

```
1138            \int_compare:nNnT { \l__tag_struct_stack_parent_tmpa_tl } = { -1 }
1139              {
1140                \seq_get:NNF
1141                  \g__tag_struct_stack_seq
1142                  \l__tag_struct_stack_parent_tmpa_tl
1143                  {
1144                    \msg_error:nn { tag } { struct-faulty-nesting }
1145                  }
1146              }
1147            \seq_gpush:NV \g__tag_struct_stack_seq  \c@g__tag_struct_abs_int
1148            \__tag_role_get:ooNN
1149              { \g__tag_struct_tag_tl }
1150              { \g__tag_struct_tag_NS_tl }
1151              \l__tag_struct_roletag_tl
1152              \l__tag_struct_roletag_NS_tl
```

We push the role tag on the stack:

```
1153            \seq_gpush:Ne \g__tag_struct_tag_stack_seq
1154              {{\g__tag_struct_tag_tl}{\l__tag_struct_roletag_tl}}
1155            \seq_gpush:Ne \g__tag_struct_roletag_stack_seq
1156              {\l__tag_struct_roletag_tl}
1157            \tl_gset:NV   \g__tag_struct_stack_current_tl \c@g__tag_struct_abs_int
1158            \__tag_struct_set_attribute:
1159            %\seq_show:N   \g__tag_struct_stack_seq
```

the rolemapped role and its NS are stored in the rolemap key.

```
1160            \__tag_struct_prop_gput:nne
1161              { \int_use:N \c@g__tag_struct_abs_int }
1162              { rolemap }
1163              {
1164                {\l__tag_struct_roletag_tl}{\l__tag_struct_roletag_NS_tl}
1165              }
```

If the role is one of Part, Div, NonStruct we have to (sometimes) retrieve the "real" parent for the parent/child test. The role of this real parent is stored in the key parentrole. If the current structure is stashed we use UNKNOWN as real parent if the current structure is rolemapped to Part, Div or NonStruct so that the children can detect that no reliable check is possible. For structures that are not rolemapped to Part, Div, NonStruct, parentrole and rolemap are always equal.

```
1166            \str_case:onTF { \l__tag_struct_roletag_tl }
1167              {
1168                {Part} {}
1169                {Div}  {}
1170                {NonStruct} {}
1171              }
1172              {
1173                \bool_if:NTF \l__tag_struct_elem_stash_bool
```

145

```
1174                     {
1175                       \__tag_struct_prop_gput:nne
1176                         { \int_use:N \c@g__tag_struct_abs_int }
1177                         { parentrole }
1178                         {
1179                           {\l__tag_struct_parenttag_tl}{\l__tag_struct_parenttag_NS_tl}
1180                         }
1181                     }
1182                     {
1183                       \prop_get:cnNT
1184                       { g__tag_struct_ \l__tag_struct_stack_parent_tmpa_tl _prop }
1185                       { parentrole }
1186                       \l__tag_get_tmpc_tl
1187                       {
1188                         \__tag_struct_prop_gput:nno
1189                           { \int_use:N \c@g__tag_struct_abs_int }
1190                           { parentrole }
1191                           {
1192                             \l__tag_get_tmpc_tl
1193                           }
1194                       }
1195                     }
1196                   }
1197                   {
1198                     \__tag_struct_prop_gput:nne
1199                       { \int_use:N \c@g__tag_struct_abs_int }
1200                       { parentrole }
1201                       {
1202                         {\l__tag_struct_roletag_tl}{\l__tag_struct_roletag_NS_tl}
1203                       }
1204                   }
1205               \bool_if:NF
1206                 \l__tag_struct_elem_stash_bool
1207                   {
```

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.

```
1208                       \socket_use:nn{tag/check/parent-child}
1209                         {
1210                           \__tag_struct_check_parent_child:oo
1211                             { \l__tag_struct_stack_parent_tmpa_tl }
1212                             { \int_use:N \c@g__tag_struct_abs_int }
1213                         }
```

Set the Parent structure number.

```
1214                       \__tag_struct_prop_gput:nne
1215                         { \int_use:N \c@g__tag_struct_abs_int }
1216                         { parentnum }
1217                         {
1218                           \l__tag_struct_stack_parent_tmpa_tl
1219                         }
1220
1221               %record this structure as kid:
1222               %\tl_show:N \g__tag_struct_stack_current_tl
                  %\tl_show:N \l__tag_struct_stack_parent_tmpa_tl
```

146

```
1223            \use:c { __tag_struct_kid_struct_gput_ \l__tag_struct_addkid_tl :ee }
1224                { \l__tag_struct_stack_parent_tmpa_tl }
1225                { \g__tag_struct_stack_current_tl }
1226            %\prop_show:c { g__tag_struct_\g__tag_struct_stack_current_tl _prop }
1227            %\seq_show:c {g__tag_struct_kids_\l__tag_struct_stack_parent_tmpa_tl _seq}
1228          }
```

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.

```
1229 ⟨debug⟩          \prop_gset_eq:cc
1230 ⟨debug⟩            { g__tag_struct_debug_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
1231 ⟨debug⟩            { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
1232 ⟨debug⟩          \prop_gput:cne
1233 ⟨debug⟩            { g__tag_struct_debug_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
1234 ⟨debug⟩            { parentnum }
1235 ⟨debug⟩            {
1236 ⟨debug⟩               \bool_if:NTF \l__tag_struct_elem_stash_bool
1237 ⟨debug⟩                 {no~parent:~stashed}
1238 ⟨debug⟩                 {
1239 ⟨debug⟩                    \l__tag_struct_stack_parent_tmpa_tl\c_space_tl =~
1240 ⟨debug⟩                    \prop_item:cn{ g__tag_struct_\l__tag_struct_stack_parent_tmpa_tl _p.
1241 ⟨debug⟩                 }
1242 ⟨debug⟩            }
1243 ⟨debug⟩          \prop_gput:cne
1244 ⟨debug⟩            { g__tag_struct_debug_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
1245 ⟨debug⟩            { NS }
1246 ⟨debug⟩            { \g__tag_struct_tag_NS_tl }
1247        %\prop_show:c { g__tag_struct_\g__tag_struct_stack_current_tl _prop }
1248        %\seq_show:c {g__tag_struct_kids_\l__tag_struct_stack_parent_tmpa_tl _seq}
1249 ⟨debug⟩ \__tag_debug_struct_begin_insert:n { #1 }
1250        \group_end:
1251      }
1252 ⟨debug⟩{ \__tag_debug_struct_begin_ignore:n { #1 }}
1253   }
1254 ⟨package⟩\cs_set_protected:Nn \tag_struct_end:
1255 ⟨debug⟩\cs_set_protected:Nn \tag_struct_end:
1256   { %take the current structure num from the stack:
1257     %the objects are written later, lua mode hasn't all needed info yet
1258     %\seq_show:N \g__tag_struct_stack_seq
1259 ⟨package⟩\__tag_check_if_active_struct:T
1260 ⟨debug⟩\__tag_check_if_active_struct:TF
1261       {
1262         \seq_gpop:NN   \g__tag_struct_roletag_stack_seq \l__tag_tmpa_tl
1263         \seq_gpop:NN   \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
1264         \seq_gpop:NNTF \g__tag_struct_stack_seq \l__tag_tmpa_tl
1265           {
1266             \__tag_check_info_closing_struct:o { \g__tag_struct_stack_current_tl }
1267           }
1268           { \__tag_check_no_open_struct: }
1269         % get the previous one, shouldn't be empty as the root should be there
1270         \seq_get:NNTF \g__tag_struct_stack_seq \l__tag_tmpa_tl
1271           {
1272             \tl_gset:No  \g__tag_struct_stack_current_tl { \l__tag_tmpa_tl }
```

147

```
1273                    \__tag_struct_set_attribute:
1274                  }
1275                  {
1276                    \__tag_check_no_open_struct:
1277                  }
1278              \seq_get:NNT \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
1279                  {
1280                    \tl_gset:Ne \g__tag_struct_tag_tl
1281                      { \exp_last_unbraced:No\use_i:nn { \l__tag_tmpa_tl } }
1282                    \prop_get:NoNT\g__tag_role_tags_NS_prop { \g__tag_struct_tag_tl} \l__tag_tmpa_tl
1283                     {
1284                       \tl_gset:Ne \g__tag_struct_tag_NS_tl { \l__tag_tmpa_tl }
1285                     }
1286                  }
1287 ⟨debug⟩\__tag_debug_struct_end_insert:
1288          }
1289 ⟨debug⟩{\__tag_debug_struct_end_ignore:}
1290    }
1291
1292 \cs_set_protected:Npn \tag_struct_end:n #1
1293  {
1294 ⟨debug⟩      \__tag_check_if_active_struct:T{\__tag_debug_struct_end_check:n{#1}}
1295     \tag_struct_end:
1296  }
1297 ⟨/package | debug⟩
```

(*End of definition for* \tag_struct_begin:n *and* \tag_struct_end:. *These functions are documented on page 114.*)

\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.

```
1298 ⟨base⟩\cs_new_protected:Npn \tag_struct_use:n #1 {}
1299 ⟨*package | debug⟩
1300 \cs_set_protected:Npn \tag_struct_use:n #1 %#1 is the label
1301   {
1302     \__tag_check_if_active_struct:T
1303       {
1304         \prop_if_exist:cTF
1305           { g__tag_struct_\property_ref:enn{tagpdfstruct-#1}{tagstruct}{unknown}_prop } %
1306           {
1307             \__tag_check_struct_used:n {#1}
1308             \tl_set:Ne \l__tag_get_child_tmpa_tl
1309               { \property_ref:enn{tagpdfstruct-#1}{tagstruct}{1} }
```

add the label structure as kid to the current structure (can be the root)

```
1310                 \__tag_struct_kid_struct_gput_right:ee
1311                   { \g__tag_struct_stack_current_tl }
1312                   { \l__tag_get_child_tmpa_tl }
```

add the current structure to the labeled one as parents

```
1313                 \__tag_prop_gput:cne
1314                   { g__tag_struct_ \l__tag_get_child_tmpa_tl _prop }
1315                   { parentnum }
1316                   {
1317                     \g__tag_struct_stack_current_tl
1318                   }
```

148

debug code

```
1319 ⟨debug⟩                \prop_gput:cne
1320 ⟨debug⟩                  { g__tag_struct_debug_ \l__tag_get_child_tmpa_tl _prop }
1321 ⟨debug⟩                  { parentnum }
1322 ⟨debug⟩                  {
1323 ⟨debug⟩                     \g__tag_struct_stack_current_tl\c_space_tl=~
1324 ⟨debug⟩                     \g__tag_struct_tag_tl
1325 ⟨debug⟩                  }
```

check if the tag is allowed as child. If the tag of the child after rolemapping is *not* one of Part, Div, NonStruct, then the parentrole field will be identically to the rolemap field and can be used for a check. Otherwise the parentrole will contain latex:STASHED (if not changed with the `parent-tag` key when the structure was stashed) and will produce a warning.

```
1326                   \socket_use:nn{tag/check/parent-child}
1327                    {
1328                       \__tag_struct_use_check_parent_child:oo
1329                         { \g__tag_struct_stack_current_tl }
1330                         { \l__tag_get_child_tmpa_tl }
1331                    }
1332                  }
1333                  {
1334                     \msg_warning:nnn{ tag }{struct-label-unknown}{#1}
1335                  }
1336              }
1337       }
1338 ⟨/package | debug⟩
```

(*End of definition for* \tag_struct_use:n. *This function is documented on page 114.*)

\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.

```
1339 ⟨base⟩\cs_new_protected:Npn \tag_struct_use_num:n #1 {}
1340 ⟨*package | debug⟩
1341 \cs_set_protected:Npn \tag_struct_use_num:n #1 %#1 is structure number
1342   {
1343     \__tag_check_if_active_struct:T
1344       {
1345         \prop_if_exist:cTF
1346         { g__tag_struct_#1_prop } %
1347         {
1348           \prop_get:cnNT
1349             {g__tag_struct_#1_prop}
1350             {parentnum}
1351             \l__tag_tmpa_tl
1352             {
1353               \msg_warning:nnn { tag } {struct-used-twice} {#1}
1354             }
```

add the #1 structure as kid to the current structure (can be the root)

```
1355               \__tag_struct_kid_struct_gput_right:ee
1356                 { \g__tag_struct_stack_current_tl }
1357                 { #1 }
```

149

add the current structure to #1 as parent

```
1358                    \__tag_struct_prop_gput:nne
1359                    { #1 }
1360                    { parentnum }
1361                    {
1362                      \g__tag_struct_stack_current_tl
1363                    }
```
`1364 ⟨debug⟩`   `\prop_gput:cne`
`1365 ⟨debug⟩`   `{ g__tag_struct_debug_#1_prop }`
`1366 ⟨debug⟩`   `{ parentnum }`
`1367 ⟨debug⟩`   `{`
`1368 ⟨debug⟩`   `\g__tag_struct_stack_current_tl\c_space_tl=~`
`1369 ⟨debug⟩`   `\g__tag_struct_tag_tl`
`1370 ⟨debug⟩`   `}`

check if the tag is allowed as child.

```
1371                    \socket_use:nn{tag/check/parent-child}
1372                    {
1373                      \__tag_struct_use_check_parent_child:oo
1374                      {\g__tag_struct_stack_current_tl}
1375                      {#1}
1376                    }
1377                  }
1378                  {
1379                    \msg_warning:nnn{ tag }{struct-label-unknown}{#1}
1380                  }
1381              }
1382          }
1383 \cs_generate_variant:Nn\tag_struct_use_num:n {o}
```
`1384 ⟨/package | debug⟩`

(*End of definition for* \tag_struct_use_num:n. *This function is documented on page 114.*)

\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.

`1385 ⟨*package⟩`
```
1386 \cs_new:Npn \tag_struct_object_ref:n #1
1387   {
1388     \pdf_object_ref_indexed:nn {__tag/struct}{ #1 }
1389   }
1390 \cs_generate_variant:Nn \tag_struct_object_ref:n {e}
```
`1391 ⟨/package⟩`

(*End of definition for* \tag_struct_object_ref:n. *This function is documented on page 114.*)

\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 existing keywords are mostly related to the Ref key (an array). The keyword `ref` takes as value an explicit object reference to a structure. The keyword `ref_-label` expects as value a label name (from a label set in a \tagstructbegin command). The keyword `ref_dest` expects a destination name set with \MakeLinkTarget. It then

150

will refer to the structure in which this `\MakeLinkTarget` was used. The keyword `ref_-num` expects a structure number. At last there is the keyword `attribute` which allows to add or extend the `/A` key of the structure. The value is the content of one attribute dictionary, so for example `/O /Layout /BBox [10 10 50 50]`. The content is stored in an object and the object reference is than added to the `/A`.

```
1392 ⟨base⟩\cs_new_protected:Npn \tag_struct_gput:nnn #1 #2 #3{}
1393 ⟨*package⟩
1394 \cs_set_protected:Npn \tag_struct_gput:nnn #1 #2 #3
1395   {
1396     \cs_if_exist_use:cF {__tag_struct_gput_data_#2:nn}
1397     { %warning??
1398       \use_none:nn
1399     }
1400     {#1}{#3}
1401   }
1402 \cs_generate_variant:Nn \tag_struct_gput:nnn {ene,nne}
1403 ⟨/package⟩
```

(*End of definition for* `\tag_struct_gput:nnn`. *This function is documented on page 114.*)

`\__tag_struct_gput_data_ref_aux:nnn`

```
1404 ⟨*package⟩
1405 \cs_new_protected:Npn \__tag_struct_gput_data_ref_aux:nnn #1 #2 #3
1406   % #1 receiving struct num, #2 key word #3 value
1407   {
1408     \prop_get:cnNTF
1409       { g__tag_struct_#1_prop }
1410       {Ref}
1411       \l__tag_get_tmpc_tl
1412       {
1413         \tl_put_right:No \l__tag_get_tmpc_tl
1414           {\cs:w __tag_struct_Ref_#2:nN \cs_end: {#3},}
1415       }
1416       {
1417         \tl_set:No \l__tag_get_tmpc_tl
1418           {\cs:w __tag_struct_Ref_#2:nN \cs_end: {#3},}
1419       }
1420     \__tag_struct_prop_gput:nno
1421       { #1 }
1422       { Ref }
1423       { \l__tag_get_tmpc_tl }
1424   }
1425 \cs_new_protected:Npn \__tag_struct_gput_data_ref:nn #1 #2
1426   {
1427     \__tag_struct_gput_data_ref_aux:nnn {#1}{obj}{#2}
1428   }
1429 \cs_new_protected:Npn \__tag_struct_gput_data_ref_label:nn #1 #2
1430   {
1431     \__tag_struct_gput_data_ref_aux:nnn {#1}{label}{#2}
1432   }
1433 \cs_new_protected:Npn \__tag_struct_gput_data_ref_dest:nn #1 #2
1434   {
1435     \__tag_struct_gput_data_ref_aux:nnn {#1}{dest}{#2}
1436   }
```

151

```
1437  \cs_new_protected:Npn \__tag_struct_gput_data_ref_num:nn #1 #2
1438    {
1439      \__tag_struct_gput_data_ref_aux:nnn {#1}{num}{#2}
1440    }
1441
1442  \cs_generate_variant:Nn \__tag_struct_gput_data_ref:nn {ee,no}
```
(*End of definition for* __tag_struct_gput_data_ref_aux:nnn.)

__tag_struct_gput_data_attribute:nn

```
1443  \cs_new_protected:Npn \__tag_struct_gput_data_attribute:nn #1 #2
1444    {
1445      \pdf_object_unnamed_write:nn {dict} {#2}
1446      \prop_get:cnNTF { g__tag_struct_#1_prop }{A} \l__tag_tmpa_tl
1447        {
1448          \tl_remove_once:Nn\l__tag_tmpa_tl{[}
1449          \tl_remove_once:Nn\l__tag_tmpa_tl{]}
1450          \__tag_prop_gput:cne { g__tag_struct_#1_prop }
1451          { A }
1452          {
1453            [ \l__tag_tmpa_tl \c_space_tl \pdf_object_ref_last: ]
1454          }
1455        }
1456        {
1457          \__tag_prop_gput:cne { g__tag_struct_#1_prop }
1458          { A }
1459          { \pdf_object_ref_last: }
1460        }
1461    }
```
(*End of definition for* __tag_struct_gput_data_attribute: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.

```
1462  \cs_new_protected:Npn \tag_struct_insert_annot:nn #1 #2 %#1 should be an object reference
1463                                                          %#2 struct parent num
1464    {
1465      \__tag_check_if_active_struct:T
1466        {
1467          \__tag_struct_insert_annot:nn {#1}{#2}
1468        }
1469    }
1470
1471  \cs_generate_variant:Nn \tag_struct_insert_annot:nn {xx,ee}
1472  \cs_new:Npn \tag_struct_parent_int: {\int_use:c { c@g__tag_parenttree_obj_int }}
1473
1474  ⟨/package⟩
1475
```
(*End of definition for* \tag_struct_insert_annot:nn *and* \tag_struct_parent_int:. *These functions are documented on page* *114.*)

7 Attributes and attribute classes

1476 ⟨∗header⟩
1477 \ProvidesExplPackage {tagpdf-attr-code} {2026-03-19} {0.99z}
1478   {part of tagpdf - code related to attributes and attribute classes}
1479 ⟨/header⟩

7.1 Variables

\g__tag_attr_entries_prop
\g__tag_attr_class_used_prop
\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_prop will hold the attributes which have been used as class name. \l_@@_attr_value_tl is used to build the attribute array or key. Every time 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

1480 ⟨∗package⟩
1481 \prop_new:N \g__tag_attr_entries_prop
1482 \prop_new_linked:N \g__tag_attr_class_used_prop
1483 \tl_new:N   \l__tag_attr_value_tl
1484 \prop_new:N \g__tag_attr_objref_prop %will contain obj num of used attributes

This seq is currently kept for compatibility with the table code.

1485 \seq_new:N\g__tag_attr_class_used_seq

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

7.2 Commands and keys

__tag_attr_new_entry:nn
role/new-attribute (setup-key)
newattribute (deprecated)

This allows to define attributes. Defined attributes are stored in a global property. role/new-attribute 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
 {
  role/new-attribute =
   {TH-col}{/O /Table /Scope /Column},
  role/new-attribute =
   {TH-row}{/O /Table /Scope /Row},
 }
```

1486 \cs_new_protected:Npn __tag_attr_new_entry:nn #1 #2 %#1:name, #2: content
1487   {
1488   \prop_gput:Nen \g__tag_attr_entries_prop
1489   {\pdf_name_from_unicode_e:n{#1}}{#2}
1490   }
1491
1492 \cs_generate_variant:Nn __tag_attr_new_entry:nn {ee}
1493 \keys_define:nn { __tag / setup }
1494   {
1495   role/new-attribute .code:n =
1496   {
1497   __tag_attr_new_entry:nn #1
1498   }

deprecated name

```
1499     ,newattribute .code:n =
1500       {
1501         \__tag_attr_new_entry:nn #1
1502       },
1503   }
```

(*End of definition for* `\__tag_attr_new_entry:nn`, `role/new-attribute (setup-key)`, *and* `newattribute` *(deprecated). These functions are documented on page 117.*)

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

```
1504 \keys_define:nn { __tag / struct }
1505   {
1506     attribute-class .code:n =
1507       {
1508         \clist_set:Ne \l__tag_tmpa_clist { #1 }
1509         \seq_set_from_clist:NN \l__tag_tmpb_seq \l__tag_tmpa_clist
```

we convert the names into pdf names with slash

```
1510         \seq_set_map_e:NNn \l__tag_tmpa_seq \l__tag_tmpb_seq
1511           {
1512             \pdf_name_from_unicode_e:n {##1}
1513           }
1514         \seq_map_inline:Nn \l__tag_tmpa_seq
1515           {
1516             \prop_get:NnNF \g__tag_attr_entries_prop {##1}\l__tag_tmpa_tl
1517               {
1518                 \msg_error:nnn { tag } { attr-unknown } { ##1 }
1519               }
1520             \prop_gput:Nnn\g__tag_attr_class_used_prop { ##1} {}
1521           }
1522         \tl_set:Ne \l__tag_tmpa_tl
1523           {
1524             \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[}
1525             \seq_use:Nn \l__tag_tmpa_seq  { \c_space_tl  }
1526             \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{]}
1527           }
1528         \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 0 }
1529           {
1530             \__tag_struct_prop_gput:nne
1531               { \int_use:N \c@g__tag_struct_abs_int }
1532               { C }
1533               { \l__tag_tmpa_tl }
1534            %\prop_show:c  { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
1535           }
1536       }
1537   }
```

attribute (*struct key*)

```
1538 \keys_define:nn { __tag / struct }
1539   {
1540     attribute .code:n  = % A property (attribute, value currently a dictionary)
1541       {
1542         \clist_set:Ne           \l__tag_tmpa_clist { #1 }
```

154

```
1543        \clist_if_empty:NF \l__tag_tmpa_clist
1544          {
1545            \seq_set_from_clist:NN \l__tag_tmpb_seq \l__tag_tmpa_clist
```
we convert the names into pdf names with slash
```
1546            \seq_set_map_e:NNn \l__tag_tmpa_seq \l__tag_tmpb_seq
1547              {
1548                \pdf_name_from_unicode_e:n {##1}
1549              }
1550            \tl_set:Ne \l__tag_attr_value_tl
1551              {
1552                \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[}%]
1553              }
1554            \seq_map_inline:Nn \l__tag_tmpa_seq
1555              {
1556                \prop_get:NnNF \g__tag_attr_entries_prop {##1}\l__tag_tmp_unused_tl
1557                  {
1558                    \msg_error:nnn { tag } { attr-unknown } { ##1 }
1559                  }
1560                \prop_get:NnNF \g__tag_attr_objref_prop {##1}\l__tag_tmpa_tl
1561                  {%\prop_show:N \g__tag_attr_entries_prop
1562                    \pdf_object_unnamed_write:ne
1563                      { dict }
1564                      {
1565                        \prop_item:Nn\g__tag_attr_entries_prop {##1}
1566                      }
1567                    \prop_gput:Nne \g__tag_attr_objref_prop {##1} {\pdf_object_ref_last:}
1568                  }
1569                \tl_put_right:Ne \l__tag_attr_value_tl
1570                  {
1571                    \c_space_tl
1572                    \prop_item:Nn \g__tag_attr_objref_prop {##1}
1573                  }
1574        %      \tl_show:N \l__tag_attr_value_tl
1575              }
1576            \tl_put_right:Ne \l__tag_attr_value_tl
1577              { %[
1578                \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{]}%
1579              }
1580        %      \tl_show:N \l__tag_attr_value_tl
1581            \__tag_struct_prop_gput:nne
1582              { \int_use:N \c@g__tag_struct_abs_int }
1583              { A }
1584              { \l__tag_attr_value_tl }
1585          }
1586    },
1587  }
1588 ⟨/package⟩
```

The tagpdf-luatex.def
Driver for luatex
Part of the tagpdf package
　　Ulrike Fischer
　　Version 0.99z, released 2026-03-19

Part IX

1 ⟨@@=tag⟩

2 ⟨∗luatex⟩

3 \ProvidesExplFile {tagpdf-luatex.def} {2026-03-19} {0.99z}

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}\selectfon

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!

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

9 \cs_set_protected:Npn __tag_prop_new:N #1

10 {

11   \prop_new:N #1

12   \lua_now:e { ltx.__tag.tables['\cs_to_str:N#1'] = {} }

13 }

14

15 \cs_set_protected:Npn __tag_prop_new_linked:N #1

16 {

17   \prop_new_linked:N #1

18   \lua_now:e { ltx.__tag.tables['\cs_to_str:N#1'] = {} }

19 }

20

21

22 \cs_set_protected:Npn __tag_seq_new:N #1

23 {

24   \seq_new:N #1

25   \lua_now:e { ltx.__tag.tables['\cs_to_str:N#1'] = {} }

26 }

27

28

29 \cs_set_protected:Npn __tag_prop_gput:Nnn #1 #2 #3

30 {

31   \prop_gput:Nnn #1 { #2 } { #3 }

32   \lua_now:e { ltx.__tag.tables['\cs_to_str:N#1'] ["#2"] = "\lua_escape:n{#3}" }

33 }

34

35 \cs_set_protected:Npn __tag_seq_gput_right:Nn #1 #2

```
36    {
37      \seq_gput_right:Nn #1 { #2 }
38      \lua_now:e { table.insert(ltx.__tag.tables['\cs_to_str:N#1'], "#2") }
39    }
```

this inserts on the right of the lua table, but as the lua table is not used for kids this is ignored for now.

```
40 \cs_set_protected:Npn \__tag_seq_gput_left:Nn #1 #2
41    {
42      \seq_gput_left:Nn #1 { #2 }
43      \lua_now:e { table.insert(ltx.__tag.tables['\cs_to_str:N#1'], "#2") }
44    }
45
46 %Hm not quite sure about the naming
47 \cs_set:Npn \__tag_seq_item:cn #1 #2
48    {
49      \lua_now:e { tex.sprint(\int_use:N\c_document_cctab,ltx.__tag.tables['#1'][#2]) }
50    }
51
52 \cs_set:Npn \__tag_prop_item:cn #1 #2
53    {
54      \lua_now:e { tex.sprint(\int_use:N\c_document_cctab,ltx.__tag.tables['#1']["#2"]) }
55    }
56
57 %for debugging commands that show both the seq/prop and the lua tables
58 \cs_set_protected:Npn \__tag_seq_show:N #1
59    {
60      \seq_show:N #1
61      \lua_now:e { ltx.__tag.trace.log ("lua~sequence~array~\cs_to_str:N#1",1) }
62      \lua_now:e { ltx.__tag.trace.show_seq (ltx.__tag.tables['\cs_to_str:N#1']) }
63    }
64
65 \cs_set_protected:Npn \__tag_prop_show:N #1
66    {
67      \prop_show:N #1
68      \lua_now:e {ltx.__tag.trace.log  ("lua~property~table~\cs_to_str:N#1",1) }
69      \lua_now:e {ltx.__tag.trace.show_prop (ltx.__tag.tables['\cs_to_str:N#1']) }
70    }
```

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

```
71 ⟨/luatex⟩
```

The module declaration

```
72 ⟨*lua⟩
73 -- tagpdf.lua
74 -- Ulrike Fischer
75
76 local ProvidesLuaModule = {
77     name          = "tagpdf",
78     version       = "0.99z",      --TAGVERSION
79     date          = "2026-03-19", --TAGDATE
80     description   = "tagpdf lua code",
81     license       = "The LATEX Project Public License 1.3c"
82 }
83
```

157

```
84  if luatexbase and luatexbase.provides_module then
85    luatexbase.provides_module (ProvidesLuaModule)
86  end
87
88  --[[
89  The code has quite probably a number of problems
90   - more variables should be local instead of global
91   - the naming is not always consistent due to the development of the code
92   - the traversing of the shipout box must be tested with more complicated setups
93   - it should probably handle more node types
94   -
95  --]]
96
```

Some comments about the lua structure.

```
97   --[[
98   the main table is named ltx.__tag. It contains the functions and also the data
99   collected during the compilation.
100
101  ltx.__tag.mc    will contain mc connected data.
102  ltx.__tag.role   will contain data related to parent-child relations.
103  ltx.__tag.struct will contain structure related data.
104  ltx.__tag.page   will contain page data
105  ltx.__tag.tables contains also data from mc and struct (from older code). This needs cleaning
106            There are certainly dublettes, but I don't dare yet ...
107  ltx.__tag.func   will contain (public) functions.
108  ltx.__tag.trace  will contain tracing/logging functions.
109  local functions starts with __
110  functions meant for users will be in ltx.tag
111
112  functions
113   ltx.__tag.func.get_num_from (tag):    takes a tag (string) and returns the id number
114   ltx.__tag.func.output_num_from (tag): takes a tag (string) and prints (to tex) the id number
115   ltx.__tag.func.get_tag_from (num):    takes a num and returns the tag
116   ltx.__tag.func.output_tag_from (num): takes a num and prints (to tex) the tag
117   ltx.__tag.func.store_mc_data (num,key,data): stores key=data in ltx.__tag.mc[num]
118   ltx.__tag.func.store_mc_label (label,num): stores label=num in ltx.__tag.mc.labels
119   ltx.__tag.func.store_mc_kid (mcnum,kid,page): stores the mc-kids of mcnum on page page
120   ltx.__tag.func.store_mc_in_page(mcnum,mcpagecnt,page): stores in the page table the number o
121   ltx.__tag.func.store_struct_mcabs (structnum,mcnum): stores relations structnum<->mcnum (abs
122   ltx.__tag.func.mc_insert_kids (mcnum): inserts the /K entries for mcnum by wandering through
123   ltx.__tag.func.mark_page_elements(box,mcpagecnt,mccntprev,mcopen,name,mctypeprev) : the main
124   ltx.__tag.func.mark_shipout (): a wrapper around the core function which inserts the last EM
125   ltx.__tag.func.fill_parent_tree_line (page): outputs the entries of the parenttree for this
126   ltx.__tag.func.output_parenttree(): outputs the content of the parenttree
127   ltx.__tag.func.pdf_object_ref(name,index): outputs the object reference for the object name
128   ltx.__tag.func.markspaceon(), ltx.__tag.func.markspaceoff(): (de)activates the marking of po
129   ltx.__tag.func.linkbin() returns the number of OBJR stored in the linkbin structure
130   ltx.__tag.trace.show_mc_data (num,loglevel): shows ltx.__tag.mc[num] is the current log leve
131   ltx.__tag.trace.show_all_mc_data (max,loglevel): shows a maximum about mc's if the current l
132   ltx.__tag.trace.show_seq: shows a sequence (array)
133   ltx.__tag.trace.show_struct_data (num): shows data of structure num
134   ltx.__tag.trace.show_prop: shows a prop
135   ltx.__tag.trace.log
136   ltx.__tag.trace.showspaces : boolean
```

```
137
138  ltx.tag.get_structnum: number, shows the current structure number
139  ltx.tag.get_structnum_next: number, shows the next structure number
140  --]]
141
```

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 structnum attribute stores the structure number. 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. The interwordspaceOff attr allows to locally suppress the insertion of real space chars, e.g. when they are inserted by other means (e.g. with `\char`). The softhyphenattribute allows to decide how to handle a soft hyphen: if unset it is surrounded by an Artifact mc, if set to 1 the char is changed. Other values will perhaps be used later.

```
142  local mctypeattributeid   = luatexbase.new_attribute ("g__tag_mc_type_attr")
143  local mccntattributeid    = luatexbase.new_attribute ("g__tag_mc_cnt_attr")
144  local structnumattributeid   = luatexbase.new_attribute ("g__tag_structnum_attr")
145  local iwspaceOffattributeid = luatexbase.new_attribute ("g__tag_interwordspaceOff_attr")
146  local iwspaceattributeid = luatexbase.new_attribute ("g__tag_interwordspace_attr")
147  local iwfontattributeid   = luatexbase.new_attribute ("g__tag_interwordfont_attr")
148  local softhyphenattribute = luatexbase.new_attribute ("l__tag_softhyphen_attr")
```

with this token we can query the state of the boolean and so detect if unmarked nodes should be marked as attributes

```
149  local tagunmarkedbool= token.create("g__tag_tagunmarked_bool")
150  local truebool        = token.create("c_true_bool")
```

with this token we can query the state of the softhyphen boolean and so detect if hyphens from hyphenation should be replaced by soft-hyphens.

```
151  local softhyphenbool = token.create("g__tag_softhyphen_bool")
```

Now a number of local versions from global tables. Not all is perhaps needed, most node variants were copied from lua-debug.

```
152  local catlatex        = luatexbase.registernumber("catcodetable@latex")
153  local tableinsert    = table.insert
154  local nodeid          = node.id
155  local nodecopy        = node.copy
156  local nodegetattribute = node.get_attribute
157  local nodesetattribute = node.set_attribute
158  local nodehasattribute = node.has_attribute
159  local nodenew         = node.new
160  local nodetail        = node.tail
161  local nodeslide       = node.slide
162  local noderemove      = node.remove
163  local nodetraverseid  = node.traverse_id
164  local nodetraverse    = node.traverse
165  local nodeinsertafter  = node.insert_after
166  local nodeinsertbefore = node.insert_before
167  local pdfpageref       = pdf.pageref
168
169  local fonthashes      = fonts.hashes
```

```
170 local identifiers     = fonthashes.identifiers
171 local fontid          = font.id
172
173 local HLIST           = node.id("hlist")
174 local VLIST           = node.id("vlist")
175 local RULE            = node.id("rule")
176 local DISC            = node.id("disc")
177 local GLUE            = node.id("glue")
178 local GLYPH           = node.id("glyph")
179 local KERN            = node.id("kern")
180 local PENALTY         = node.id("penalty")
181 local LOCAL_PAR       = node.id("local_par")
182 local MATH            = node.id("math")
183
184 local NEXT = next
185 local explicit_disc = 1
186 local regular_disc = 3
```

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

```
187 ltx              = ltx          or { }
188 ltx.tag          = ltx.tag        or { } -- user commands
189 ltx.__tag        = ltx.__tag         or { }
190 ltx.__tag.mc     = ltx.__tag.mc    or { } -- mc data
191 ltx.__tag.role   = ltx.__tag.role  or { } -- parent-child data
192 ltx.__tag.role.states = ltx.__tag.role.states   or { } -- the states
193 ltx.__tag.role.index  = ltx.__tag.role.index    or { } -- standard types to index
194                                            --- numbers
195 ltx.__tag.role.matrix = ltx.__tag.role.matrix   or { } -- implements the matrix
196 ltx.__tag.struct   = ltx.__tag.struct or  { } -- struct data
197 ltx.__tag.tables   = ltx.__tag.tables or  { } -- tables created with new prop and new seq.
198                                        -- wasn't a so great idea ...
199                                        -- g__tag_role_tags_seq used by tag<-> is in this tab
200                                        -- used for pure lua tables too now!
201 ltx.__tag.page     = ltx.__tag.page   or  { } -- page data, currently only i->{0->mcnum,1->mc
202 ltx.__tag.trace    = ltx.__tag.trace  or { } -- show commands
203 ltx.__tag.func     = ltx.__tag.func   or { } -- functions
204 ltx.__tag.conf     = ltx.__tag.conf   or { } -- configuration variables
```

2   User commands to access data

Code like the one in luamml will have to access the current state in some places.

\

```
205 local __tag_get_struct_num =
206 function()
207  local a = token.get_macro("g__tag_struct_stack_current_tl")
208  return a
209 end
210
211 local __tag_get_struct_counter =
212 function()
213  local a = tex.getcount("c@g__tag_struct_abs_int")
214  return a
```

```
215  end
216
217  local __tag_get_struct_num_next =
218   function()
219    local a = tex.getcount("c@g__tag_struct_abs_int") + 1
220    return a
221   end
222
223  ltx.tag.get_struct_num = __tag_get_struct_num
224  ltx.tag.get_struct_counter = __tag_get_struct_counter
225  ltx.tag.get_struct_num_next = __tag_get_struct_num_next
```
(*End of definition for* \. *This function is documented on page* **??**.)

3 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.

```
226  local __tag_log =
227   function (message,loglevel)
228    if (loglevel or 3) <= tex.count["l__tag_loglevel_int"] then
229     texio.write_nl("tagpdf: ".. message)
230    end
231   end
232
233  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.

```
234  function ltx.__tag.trace.show_seq (seq)
235   if (type(seq) == "table") then
236    for i,v in ipairs(seq) do
237     __tag_log ("[" .. i .. "] => " .. tostring(v),1)
238    end
239   else
240    __tag_log ("sequence " .. tostring(seq) .. " not found",1)
241   end
242  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.

```
243  local __tag_pairs_prop =
244   function  (prop)
245       local a = {}
246       for n in pairs(prop) do tableinsert(a, n) end
247       table.sort(a)
248       local i = 0                -- iterator variable
249       local iter = function ()   -- iterator function
250         i = i + 1
```

161

```
251     if a[i] == nil then return nil
252     else return a[i], prop[a[i]]
253       end
254   end
255   return iter
256 end
257

258
259 function ltx.__tag.trace.show_prop (prop)
260   if (type(prop) == "table") then
261     for i,v in __tag_pairs_prop (prop) do
262       __tag_log ("[" .. i .. "] => " .. tostring(v),1)
263     end
264   else
265     __tag_log ("prop " .. tostring(prop) .. " not found or not a table",1)
266   end
267 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

```
268 function ltx.__tag.trace.show_mc_data (num,loglevel)
269   if ltx.__tag and ltx.__tag.mc and ltx.__tag.mc[num] then
270     for k,v in pairs(ltx.__tag.mc[num]) do
271       __tag_log ("mc"..num..": "..tostring(k).."=>"..tostring(v),loglevel)
272     end
273     if ltx.__tag.mc[num]["kids"] then
274       __tag_log ("mc" .. num .. " has " .. #ltx.__tag.mc[num]["kids"] .. " kids",loglevel)
275       for k,v in ipairs(ltx.__tag.mc[num]["kids"]) do
276         __tag_log ("mc ".. num .. " kid "..k.." =>" .. v.kid.." on page " ..v.page,loglevel)
277       end
278     end
279   else
280     __tag_log ("mc"..num.." not found",loglevel)
281   end
282 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.

```
283 function ltx.__tag.trace.show_all_mc_data (min,max,loglevel)
284   for i = min, max do
285     ltx.__tag.trace.show_mc_data (i,loglevel)
286   end
287   texio.write_nl("")
288 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.

```
289 function ltx.__tag.trace.show_struct_data (num)
290   if ltx.__tag and ltx.__tag.struct and ltx.__tag.struct[num] then
291     for k,v in ipairs(ltx.__tag.struct[num]) do
292       __tag_log ("struct "..num..": "..tostring(k).."=>"..tostring(v),1)
```

162

```
293    end
294  else
295    __tag_log   ("struct ".. num.." not found ",1)
296  end
297 end
```
(*End of definition for* ltx.__tag.trace.show_struct_data.)

4   Helper functions

4.1   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.

```
298 local __tag_get_mc_cnt_type_tag = function (n)
299   local mccnt      = nodegetattribute(n,mccntattributeid)  or -1
300   local mctype     = nodegetattribute(n,mctypeattributeid)  or -1
301   local tag        = ltx.__tag.func.get_tag_from(mctype)
302   return mccnt,mctype,tag
303 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.

```
304 local function __tag_get_mathsubtype  (mathnode)
305   if mathnode.subtype == 0 then
306     subtype = "beginmath"
307   else
308     subtype = "endmath"
309   end
310   return subtype
311 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.

```
312 ltx.__tag.tables.role_tag_attribute = {}
313 ltx.__tag.tables.role_attribute_tag = {}
```
(*End of definition for* ltx.__tag.tables.role_tag_attribute.)

ltx.__tag.func.alloctag

```
314 local __tag_alloctag =
315  function (tag)
316    if not ltx.__tag.tables.role_tag_attribute[tag] then
317      table.insert(ltx.__tag.tables.role_attribute_tag,tag)
318      ltx.__tag.tables.role_tag_attribute[tag]=#ltx.__tag.tables.role_attribute_tag
319      __tag_log  ("Add ".. tag.." "..ltx.__tag.tables.role_tag_attribute[tag],3)
320    end
321  end
322 ltx.__tag.func.alloctag = __tag_alloctag
```
(*End of definition for* ltx.__tag.func.alloctag.)

__tag_get_num_from
ltx.__tag.func.get_num_from
ltx.__tag.func.output_num_from

These functions take as argument a string `tag`, and return the number under which is it recorded (and so the attribute value). The first function outputs the number for lua, while the `output` function outputs to tex.

```
323 local __tag_get_num_from =
324  function (tag)
325   if ltx.__tag.tables.role_tag_attribute[tag] then
326     a= ltx.__tag.tables.role_tag_attribute[tag]
327   else
328     a= -1
329   end
330   return a
331  end
332
333 ltx.__tag.func.get_num_from = __tag_get_num_from
334
335 function ltx.__tag.func.output_num_from (tag)
336   local num = __tag_get_num_from (tag)
337   tex.sprint(catlatex,num)
338   if num == -1 then
339    __tag_log ("Unknown tag "..tag.." used")
340   end
341 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.

```
342 local __tag_get_tag_from =
343  function  (num)
344   if ltx.__tag.tables.role_attribute_tag[num] then
345     a = ltx.__tag.tables.role_attribute_tag[num]
346   else
347     a= "UNKNOWN"
348   end
349   return a
350 end
351
352 ltx.__tag.func.get_tag_from = __tag_get_tag_from
353
354 function ltx.__tag.func.output_tag_from (num)
355   tex.sprint(catlatex,__tag_get_tag_from (num))
356 end
```

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

ltx.__tag.func.store_mc_data

This function stores for `key=data` for mc-chunk `num`. It is used in the tagpdf-mc code, to store for example the tag string, and the raw options.

```
357 function ltx.__tag.func.store_mc_data (num,key,data)
358  ltx.__tag.mc[num] = ltx.__tag.mc[num] or { }
359  ltx.__tag.mc[num][key] = data
360  __tag_log  ("INFO TEX-STORE-MC-DATA: "..num.." => "..tostring(key).." => "..tostring(data),3
361 end
```

(*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.

```
362 function ltx.__tag.func.store_mc_label (label,num)
363   ltx.__tag.mc["labels"] = ltx.__tag.mc["labels"] or { }
364   ltx.__tag.mc.labels[label] = num
365 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.

```
366 function ltx.__tag.func.store_mc_kid (mcnum,kid,page)
367   __tag_log("INFO TAG-STORE-MC-KID: "..mcnum.." => " .. kid.." on page " .. page,3)
368   ltx.__tag.mc[mcnum]["kids"] = ltx.__tag.mc[mcnum]["kids"] or { }
369   local kidtable = {kid=kid,page=page}
370   tableinsert(ltx.__tag.mc[mcnum]["kids"], kidtable )
371 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.

```
372 function ltx.__tag.func.mc_num_of_kids (mcnum)
373   local num = 0
374   if ltx.__tag.mc[mcnum] and ltx.__tag.mc[mcnum]["kids"] then
375     num = #ltx.__tag.mc[mcnum]["kids"]
376   end
377   __tag_log ("INFO MC-KID-NUMBERS: " .. mcnum .. "has " .. num .. "KIDS",4)
378   return num
379 end
```

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

4.2 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

```
380 local __tag_backend_create_emc_node
381 if tex.outputmode == 0 then
382   if token.get_macro("c_sys_backend_str") == "dvipdfmx" then
383     function __tag_backend_create_emc_node ()
384       local emcnode = nodenew("whatsit","special")
385         emcnode.data = "pdf:code EMC"
386       return emcnode
387     end
388   else -- assume a dvips variant
389     function __tag_backend_create_emc_node ()
390       local emcnode = nodenew("whatsit","special")
391         emcnode.data = "ps:SDict begin mark /EMC pdfmark end"
392       return emcnode
393     end
394   end
395 else -- pdf mode
396   function __tag_backend_create_emc_node ()
```

```
397     local emcnode = nodenew("whatsit","pdf_literal")
398       emcnode.data = "EMC"
399       emcnode.mode=1
400     return emcnode
401   end
402 end
403
404 local function __tag_insert_emc_node (head,current)
405   local emcnode= __tag_backend_create_emc_node()
406   head = node.insert_before(head,current,emcnode)
407   return head
408 end
```

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

This inserts a simple bmc node

```
409 local __tag_backend_create_bmc_node
410 if tex.outputmode == 0 then
411  if token.get_macro("c_sys_backend_str") == "dvipdfmx" then
412   function __tag_backend_create_bmc_node (tag)
413     local bmcnode = nodenew("whatsit","special")
414     bmcnode.data = "pdf:code /"..tag.." BMC"
415     return bmcnode
416   end
417  else -- assume a dvips variant
418   function __tag_backend_create_bmc_node (tag)
419     local bmcnode = nodenew("whatsit","special")
420     bmcnode.data = "ps:SDict begin mark/"..tag.." /BMC pdfmark end"
421     return bmcnode
422   end
423  end
424 else -- pdf mode
425   function __tag_backend_create_bmc_node (tag)
426     local bmcnode = nodenew("whatsit","pdf_literal")
427     bmcnode.data = "/"..tag.." BMC"
428     bmcnode.mode=1
429     return bmcnode
430   end
431 end
432
433 local function __tag_insert_bmc_node (head,current,tag)
434  local bmcnode = __tag_backend_create_bmc_node (tag)
435  head = node.insert_before(head,current,bmcnode)
436  return head
437 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.

```
438 local __tag_backend_create_bdc_node
439
440 if tex.outputmode == 0 then
441  if token.get_macro("c_sys_backend_str") == "dvipdfmx" then
442   function __tag_backend_create_bdc_node (tag,dict)
```

```
443    local bdcnode = nodenew("whatsit","special")
444    bdcnode.data = "pdf:code /"..tag.."<<"..dict..">> BDC"
445    return bdcnode
446  end
447  else -- assume a dvips variant
448   function __tag_backend_create_bdc_node (tag,dict)
449    local bdcnode = nodenew("whatsit","special")
450    bdcnode.data = "ps:SDict begin mark/"..tag.."<<"..dict..">> /BDC pdfmark end"
451    return bdcnode
452   end
453  end
454 else -- pdf mode
455   function __tag_backend_create_bdc_node (tag,dict)
456    local bdcnode = nodenew("whatsit","pdf_literal")
457    bdcnode.data = "/"..tag.."<<"..dict..">> BDC"
458    bdcnode.mode=1
459    return bdcnode
460   end
461 end
462
463 local function __tag_insert_bdc_node (head,current,tag,dict)
464  bdcnode= __tag_backend_create_bdc_node (tag,dict)
465  head = node.insert_before(head,current,bdcnode)
466  return head
467 end
```
(*End of definition for* `__tag_backend_create_bdc_node` *and* `__tag_insert_bdc_node`.)

`__tag_pdf_object_ref` This allows to reference a pdf object reserved with the l3pdf command by name. The return value is `n 0 R`, if the object doesn't exist, n is 0.

```
468 local function __tag_pdf_object_ref (name,index)
469    local object
470    if ltx.pdf.object_id then
471      object = ltx.pdf.object_id (name,index) ..' 0 R'
472    else
473      local tokenname = 'c__pdf_object_'..name..'/'..index..'_int'
474      object = token.create(tokenname).mode ..' 0 R'
475    end
476    return object
477 end
478 ltx.__tag.func.pdf_object_ref = __tag_pdf_object_ref
```
(*End of definition for* `__tag_pdf_object_ref`.)

5   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.

```
479 local function __tag_show_spacemark (head,current,color,height)
480  local markcolor = color or "1 0 0"
481  local markheight = height or 10
482  local pdfstring
483  if tex.outputmode == 0 then
484   -- ignore dvi mode for now
```

```
485  else
486    pdfstring = node.new("whatsit","pdf_literal")
487        pdfstring.data =
488        string.format("q "..markcolor.." RG "..markcolor.." rg 0.4 w 0 %g m 0 %g l S Q",-
  3,markheight)
489        head = node.insert_after(head,current,pdfstring)
490    return head
491   end
492 end
```

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

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

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

```
493 local function __tag_fakespace()
494     tex.setattribute(iwspaceattributeid,1)
495     tex.setattribute(iwfontattributeid,font.current())
496 end
497 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.

```
498 --[[ a function to mark up places where real space chars should be inserted
499     it only sets an attribute.
500 --]]
501
502 local function __tag_mark_spaces (head)
503   local inside_math = false
504   for n in nodetraverse(head) do
505     local id = n.id
506     if id == GLYPH then
507       local glyph = n
508       default_currfontid = glyph.font
509       if glyph.next and (glyph.next.id == GLUE)
510         and not inside_math  and (glyph.next.width >0)
511       then
512         nodesetattribute(glyph.next,iwspaceattributeid,1)
513         nodesetattribute(glyph.next,iwfontattributeid,glyph.font)
514       -- for debugging
515        if ltx.__tag.trace.showspaces then
516         __tag_show_spacemark (head,glyph)
517        end
518       elseif glyph.next and (glyph.next.id==KERN) and not inside_math then
519        local kern = glyph.next
520        if kern.next and (kern.next.id== GLUE)  and (kern.next.width >0)
521        -- the attribute is also set on the kern in case the kern+glue is
522        -- discarded at a line break tagging issue #1102
523        -- TODO iterate back through all discardable nodes.
524        then
525         nodesetattribute(kern,iwspaceattributeid,1)
526         nodesetattribute(kern,iwfontattributeid,glyph.font)
527         nodesetattribute(kern.next,iwspaceattributeid,1)
528         nodesetattribute(kern.next,iwfontattributeid,glyph.font)
```

```
529          end
530        end
531      --  look also back
532      if glyph.prev and (glyph.prev.id == GLUE)
533          and not inside_math
534          and (glyph.prev.width >0)
535          and not nodehasattribute(glyph.prev,iwspaceattributeid)
536        then
537          nodesetattribute(glyph.prev,iwspaceattributeid,1)
538          nodesetattribute(glyph.prev,iwfontattributeid,glyph.font)
539        -- for debugging
540         if ltx.__tag.trace.showspaces then
541          __tag_show_spacemark (head,glyph)
542         end
543        end
544      elseif id == PENALTY then
545        local glyph = n
546        -- __tag_log ("PENALTY ".. n.subtype.."VALUE"..n.penalty,3)
547        if glyph.next and (glyph.next.id == GLUE)
548          and not inside_math  and (glyph.next.width >0) and n.subtype==0
549        then
550          nodesetattribute(glyph.next,iwspaceattributeid,1)
551          --  changed 2024-01-18, issue #72
552          nodesetattribute(glyph.next,iwfontattributeid,default_currfontid)
553        -- for debugging
554         if ltx.__tag.trace.showspaces then
555          __tag_show_spacemark (head,glyph)
556         end
557        end
558      elseif id == MATH then
559        inside_math = (n.subtype == 0)
560      end
561    end
562    return head
563 end
```

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

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

`__tag_activate_mark_space`
`ltx.__tag.func.markspaceon`
`ltx.__tag.func.markspaceoff`

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

```
564 local function __tag_activate_mark_space ()
565  if not luatexbase.in_callback ("pre_linebreak_filter","markspaces") then
566   luatexbase.add_to_callback("pre_linebreak_filter",__tag_mark_spaces,"markspaces")
567   luatexbase.add_to_callback("hpack_filter",__tag_mark_spaces,"markspaces")
568  end
569 end
570
571 ltx.__tag.func.markspaceon=__tag_activate_mark_space
572
573 local function __tag_deactivate_mark_space ()
574  if luatexbase.in_callback ("pre_linebreak_filter","markspaces") then
575  luatexbase.remove_from_callback("pre_linebreak_filter","markspaces")
576  luatexbase.remove_from_callback("hpack_filter","markspaces")
577  end
578 end
```

169

```
580 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.

```
581 local default_space_char = nodenew(GLYPH)
582 local default_fontid     = fontid("TU/lmr/m/n/10")
583 local default_currfontid = fontid("TU/lmr/m/n/10")
584 default_space_char.char  = 32
585 default_space_char.font  = default_fontid
```

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

```
586 local function __tag_font_has_space (fontid)
587  t= fonts.hashes.identifiers[fontid]
588  if luaotfload.aux.slot_of_name(fontid,"space")
589    or t and t.characters and t.characters[32] and t.characters[32]["unicode"]==32
590  then
591    return true
592  else
593    return false
594  end
595 end
```

`__tag_space_chars_shipout`

`ltx.__tag.func.space_chars_shipout`

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.

```
596 local function __tag_space_chars_shipout (box)
597  local head = box.head
598   if head then
599     for n in node.traverse(head) do
600       local spaceattr = -1
601       if not nodehasattribute(n,iwspaceOffattributeid) then
602         spaceattr = nodegetattribute(n,iwspaceattributeid)  or -1
603       end
604       if n.id == HLIST  then -- enter the hlist
605          __tag_space_chars_shipout (n)
606       elseif n.id == VLIST then -- enter the vlist
607          __tag_space_chars_shipout (n)
608       elseif n.id == GLUE then
609         if ltx.__tag.trace.showspaces and spaceattr==1  then
610           __tag_show_spacemark (head,n,"0 1 0")
611         end
612         if spaceattr==1  then
613           local space
614           local space_char = node.copy(default_space_char)
615           local curfont    = nodegetattribute(n,iwfontattributeid)
616           __tag_log ("INFO SPACE-FUNCTION-FONT: ".. tostring(curfont),3)
617           if curfont and
618             -- luaotfload.aux.slot_of_name(curfont,"space")
619             __tag_font_has_space (curfont)
620           then
621             space_char.font=curfont
622           end
623           head, space = node.insert_before(head, n, space_char) --
```

```
624            n.width    = n.width - space.width
625            space.attr  = n.attr
626          end
627        end
628      end
629      box.head = head
630    end
631 end
632
633 function ltx.__tag.func.space_chars_shipout (box)
634   __tag_space_chars_shipout (box)
635 end
```

(*End of definition for* `__tag_space_chars_shipout` *and* `ltx.__tag.func.space_chars_shipout`.)

6 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.

```
636 function ltx.__tag.func.mc_insert_kids (mcnum,single)
637   if ltx.__tag.mc[mcnum] then
638   __tag_log("INFO TEX-MC-INSERT-KID-TEST: " .. mcnum,4)
639    if ltx.__tag.mc[mcnum]["kids"] then
640     if #ltx.__tag.mc[mcnum]["kids"] > 1 and single==1 then
641      tex.sprint(catlatex,"[")
642     end
643     for i,kidstable in ipairs( ltx.__tag.mc[mcnum]["kids"] ) do
644      local kidnum  = kidstable["kid"]
645      local kidpage = kidstable["page"]
646      local kidpageobjnum = pdfpageref(kidpage)
647      __tag_log("INFO TEX-MC-INSERT-KID: " .. mcnum ..
648                    " insert KID " ..i..
649                    " with num " .. kidnum ..
650                    " on page " .. kidpage.."/"..kidpageobjnum,3)
651     tex.sprint(catlatex,"<</Type /MCR /Pg "..kidpageobjnum .. " 0 R /MCID "..kidnum.. ">> "
652     end
653     if #ltx.__tag.mc[mcnum]["kids"] > 1 and single==1 then
654      tex.sprint(catlatex,"]")
655     end
656    else
657     -- this is typically not a problem, e.g. empty hbox in footer/header can
658     -- trigger this warning.
659     __tag_log("WARN TEX-MC-INSERT-NO-KIDS: "..mcnum.." has no kids",2)
660     if single==1 then
661       tex.sprint(catlatex,"null")
662     end
663    end
664   else
665    __tag_log("WARN TEX-MC-INSERT-MISSING: "..mcnum.." doesn't exist",0)
666   end
667 end
```

171

(*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.

```
668 function ltx.__tag.func.store_struct_mcabs (structnum,mcnum)
669   ltx.__tag.struct[structnum]=ltx.__tag.struct[structnum] or { }
670   ltx.__tag.struct[structnum]["mc"]=ltx.__tag.struct[structnum]["mc"] or { }
671   -- a structure can contain more than on mc chunk, the content should be ordered
672   tableinsert(ltx.__tag.struct[structnum]["mc"],mcnum)
673   __tag_log("INFO TEX-MC-INTO-STRUCT: "..
674                     mcnum.." inserted in struct "..structnum,3)
675   -- but every mc can only be in one structure
676   ltx.__tag.mc[mcnum]= ltx.__tag.mc[mcnum] or { }
677   ltx.__tag.mc[mcnum]["parent"] = structnum
678 end
679
```

(*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.

```
680 -- pay attention: lua counts arrays from 1, tex pages from one
681 -- mcid and arrays in pdf count from 0.
682 function ltx.__tag.func.store_mc_in_page (mcnum,mcpagecnt,page)
683   ltx.__tag.page[page] = ltx.__tag.page[page] or {}
684   ltx.__tag.page[page][mcpagecnt] = mcnum
685   __tag_log("INFO TAG-MC-INTO-PAGE: page " .. page ..
686                     ": inserting MCID " .. mcpagecnt .. " => " .. mcnum,3)
687 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)

```
688 local function __tag_update_mc_attributes (head,mcnum,type)
689   for n in node.traverse(head) do
690     node.set_attribute(n,mccntattributeid,mcnum)
691     node.set_attribute(n,mctypeattributeid,type)
692     if n.id == HLIST or n.id == VLIST then
693       __tag_update_mc_attributes (n.list,mcnum,type)
694     end
695   end
696   return head
697 end
698 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.

```
699 --[[
700     Now follows the core function
701     It wades through the shipout box and checks the attributes
702     ARGUMENTS
703     box: is a box,
704     mcpagecnt: num, the current page cnt of mc (should start at -1 in shipout box), needed fo
```

```
705    mccntprev: num, the attribute cnt of the previous node/whatever - if different we have a
706    mcopen: num, records if some bdc/emc is open
707    These arguments are only needed for log messages, if not present are replaces by fix stri
708    name: string to describe the box
709    mctypeprev: num, the type attribute of the previous node/whatever
710
711    there are lots of logging messages currently. Should be cleaned up in due course.
712    One should also find ways to make the function shorter.
713  --]]
714
715  function ltx.__tag.func.mark_page_elements (box,mcpagecnt,mccntprev,mcopen,name,mctypeprev)
716    local name = name or ("SOMEBOX")
717    local mctypeprev = mctypeprev or -1
718    local abspage = status.total_pages + 1  -- the real counter is increased
719                                            -- inside the box so one off
720                                            -- if the callback is not used. (???)
721    __tag_log ("INFO TAG-ABSPAGE: " .. abspage,3)
722    __tag_log ("INFO TAG-ARGS: pagecnt".. mcpagecnt..
723                    " prev "..mccntprev ..
724                    " type prev "..mctypeprev,4)
725    __tag_log ("INFO TAG-TRAVERSING-BOX: ".. tostring(name)..
726                    " TYPE ".. node.type(node.getid(box)),3)
727    local head = box.head -- ShipoutBox is a vlist?
728    if head then
729      mccnthead, mctypehead,taghead = __tag_get_mc_cnt_type_tag (head)
730      __tag_log ("INFO TAG-HEAD: " ..
731                      node.type(node.getid(head))..
732                      " MC"..tostring(mccnthead)..
733                      " => TAG " .. tostring(mctypehead)..
734                      " => ".. tostring(taghead),3)
735    else
736      __tag_log ("INFO TAG-NO-HEAD: head is "..
737                      tostring(head),3)
738    end
739    for n in node.traverse(head) do
740      local mccnt, mctype, tag = __tag_get_mc_cnt_type_tag (n)
741      local spaceattr = nodegetattribute(n,iwspaceattributeid)  or -1
742      __tag_log ("INFO TAG-NODE: "..
743                      node.type(node.getid(n))..
744                      " MC".. tostring(mccnt)..
745                      " => TAG ".. tostring(mctype)..
746                      " => " ..  tostring(tag),3)
747      if n.id == HLIST
748      then -- enter the hlist
749       mcopen,mcpagecnt,mccntprev,mctypeprev=
750       ltx.__tag.func.mark_page_elements (n,mcpagecnt,mccntprev,mcopen,"INTERNAL HLIST",mctype
751      elseif n.id == VLIST then -- enter the vlist
752       mcopen,mcpagecnt,mccntprev,mctypeprev=
753       ltx.__tag.func.mark_page_elements (n,mcpagecnt,mccntprev,mcopen,"INTERNAL VLIST",mctype
754      elseif n.id == GLUE and not n.leader then -- at glue real space chars are inserted, but t
755                                      -- been done if the previous shipout wandering, so here it
756      elseif n.id == LOCAL_PAR then  -- local_par is ignored
757      elseif n.id == PENALTY then    -- penalty is ignored
758      elseif n.id == KERN then       -- kern is ignored
```

173

```lua
759      __tag_log ("INFO TAG-KERN-SUBTYPE: "..
760        node.type(node.getid(n)).." "..n.subtype,4)
761    else
762     -- math is currently only logged.
763     -- we could mark the whole as math
764     -- for inner processing the mlist_to_hlist callback is probably needed.
765     if n.id == MATH then
766      __tag_log("INFO TAG-MATH-SUBTYPE: "..
767        node.type(node.getid(n)).." "..__tag_get_mathsubtype(n),4)
768     end
769     -- endmath
770     __tag_log("INFO TAG-MC-COMPARE: current "..
771              mccnt.." prev "..mccntprev,4)
772     if mccnt~=mccntprev then -- a new mc chunk
773      __tag_log ("INFO TAG-NEW-MC-NODE: "..
774                          node.type(node.getid(n))..
775                          " MC"..tostring(mccnt)..
776                          " <=> PREVIOUS "..tostring(mccntprev),4)
777      if mcopen~=0 then -- there is a chunk open, close it (hope there is only one ...
778       box.list=__tag_insert_emc_node (box.list,n)
779       mcopen = mcopen - 1
780       __tag_log ("INFO TAG-INSERT-EMC: " ..
781         mcpagecnt .. " MCOPEN = " .. mcopen,3)
782       if mcopen ~=0 then
783        __tag_log ("WARN TAG-OPEN-MC: " .. mcopen,1)
784       end
785      end
786      if ltx.__tag.mc[mccnt] then
787       if ltx.__tag.mc[mccnt]["artifact"] then
788        __tag_log("INFO TAG-INSERT-ARTIFACT: "..
789                          tostring(ltx.__tag.mc[mccnt]["artifact"]),3)
790        if ltx.__tag.mc[mccnt]["artifact"] == "" then
791         box.list = __tag_insert_bmc_node (box.list,n,"Artifact")
792        else
793         box.list = __tag_insert_bdc_node (box.list,n,"Artifact", "/Type /"..ltx.__tag.mc[mcc
794        end
795       else
796        __tag_log("INFO TAG-INSERT-TAG: "..
797                          tostring(tag),3)
798        mcpagecnt = mcpagecnt +1
799        __tag_log ("INFO TAG-INSERT-BDC: "..mcpagecnt,3)
800        local dict= "/MCID "..mcpagecnt
801        if ltx.__tag.mc[mccnt]["raw"] then
802         __tag_log("INFO TAG-USE-RAW: "..
803           tostring(ltx.__tag.mc[mccnt]["raw"]),3)
804         dict= dict .. " " .. ltx.__tag.mc[mccnt]["raw"]
805        end
806        if ltx.__tag.mc[mccnt]["alt"] then
807         __tag_log("INFO TAG-USE-ALT: "..
808            tostring(ltx.__tag.mc[mccnt]["alt"]),3)
809         dict= dict .. " " .. ltx.__tag.mc[mccnt]["alt"]
810        end
811        if ltx.__tag.mc[mccnt]["lang"] then
812         __tag_log("INFO TAG-USE-LANG: "..
```

```
813              tostring(ltx.__tag.mc[mccnt]["lang"]),3)
814          dict= dict .. " " .. ltx.__tag.mc[mccnt]["lang"]
815        end
816        if ltx.__tag.mc[mccnt]["actualtext"] then
817         __tag_log("INFO TAG-USE-ACTUALTEXT: "..
818            tostring(ltx.__tag.mc[mccnt]["actualtext"]),3)
819         dict= dict .. " " .. ltx.__tag.mc[mccnt]["actualtext"]
820        end
821        box.list = __tag_insert_bdc_node (box.list,n,tag, dict)
822        ltx.__tag.func.store_mc_kid (mccnt,mcpagecnt,abspage)
823        ltx.__tag.func.store_mc_in_page(mccnt,mcpagecnt,abspage)
824        ltx.__tag.trace.show_mc_data (mccnt,3)
825       end
826       mcopen = mcopen + 1
827      else
828       if tagunmarkedbool.mode == truebool.mode then
829        __tag_log("INFO TAG-NOT-TAGGED: this has not been tagged, using artifact",2)
830        box.list = __tag_insert_bmc_node (box.list,n,"Artifact")
831        mcopen = mcopen + 1
832       else
833        __tag_log("WARN TAG-NOT-TAGGED: this has not been tagged",1)
834       end
835      end
836      mccntprev = mccnt
837     end
838    end -- end if
839   end -- end for
840   if head then
841    mccnthead, mctypehead,taghead = __tag_get_mc_cnt_type_tag (head)
842    __tag_log ("INFO TAG-ENDHEAD: " ..
843                    node.type(node.getid(head))..
844                    " MC"..tostring(mccnthead)..
845                    " => TAG "..tostring(mctypehead)..
846                    " => "..tostring(taghead),4)
847   else
848    __tag_log ("INFO TAG-ENDHEAD: ".. tostring(head),4)
849   end
850   __tag_log ("INFO TAG-QUITTING-BOX "..
851                    tostring(name)..
852                    " TYPE ".. node.type(node.getid(box)),4)
853  return mcopen,mcpagecnt,mccntprev,mctypeprev
854 end
855
```

(*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.

```
856 function ltx.__tag.func.mark_shipout (box)
857  mcopen = ltx.__tag.func.mark_page_elements (box,-1,-100,0,"Shipout",-1)
858  if mcopen~=0 then -- there is a chunk open, close it (hope there is only one ...
859   local emcnode = __tag_backend_create_emc_node ()
860   local list = box.list
861   if list then
```

```
862    list = node.insert_after (list,node.tail(list),emcnode)
863    mcopen = mcopen - 1
864    __tag_log ("INFO SHIPOUT-INSERT-LAST-EMC: MCOPEN " .. mcopen,3)
865  else
866    __tag_log ("WARN SHIPOUT-UPS: this shouldn't happen",0)
867  end
868  if mcopen ~=0 then
869    __tag_log ("WARN SHIPOUT-MC-OPEN: " .. mcopen,1)
870  end
871  end
872 end
```

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

7 Parenttree

<div style="margin-left:2em">

ltx.__tag.func.fill_parent_tree_line
ltx.__tag.func.output_parenttree

</div>

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.

```
873 function ltx.__tag.func.fill_parent_tree_line (page)
874     -- we need to get page-> i=kid -> mcnum -> structnum
875     -- pay attention: the kid numbers and the page number in the parent tree start with 0!
876     local numsentry =""
877     local pdfpage = page-1
878     if ltx.__tag.page[page] and ltx.__tag.page[page][0] then
879      mcchunks=#ltx.__tag.page[page]
880      __tag_log("INFO PARENTTREE-NUM:  page "..
881                   page.." has "..mcchunks.."+1 Elements ",4)
882      for i=0,mcchunks do
883      -- what does this log??
884       __tag_log("INFO PARENTTREE-CHUNKS:  "..
885         ltx.__tag.page[page][i],4)
886      end
887      if mcchunks == 0 then
888       -- only one chunk so no need for an array
889       local mcnum  = ltx.__tag.page[page][0]
890       local structnum = ltx.__tag.mc[mcnum]["parent"]
891       local propname  = "g__tag_struct_"..structnum.."_prop"
892       --local objref   =  ltx.__tag.tables[propname]["objref"] or "XXXX"
893       local objref = __tag_pdf_object_ref('__tag/struct',structnum)
894       __tag_log("INFO PARENTTREE-STRUCT-OBJREF:  =====>"..
895         tostring(objref),5)
896      numsentry = pdfpage .. " [".. objref .. "]"
897       __tag_log("INFO PARENTTREE-NUMENTRY: page " ..
898         page.. " num entry = ".. numsentry,3)
899      else
900       numsentry = pdfpage .. " ["
901        for i=0,mcchunks do
902         local mcnum  = ltx.__tag.page[page][i]
903         local structnum = ltx.__tag.mc[mcnum]["parent"] or 0
904         local propname  = "g__tag_struct_"..structnum.."_prop"
905         --local objref   =  ltx.__tag.tables[propname]["objref"] or "XXXX"
906         local objref = __tag_pdf_object_ref('__tag/struct',structnum)
907         numsentry = numsentry .. " ".. objref
908        end
```

```
909    numsentry = numsentry .. "] "
910    __tag_log("INFO PARENTTREE-NUMENTRY: page " ..
911      page.. " num entry = ".. numsentry,3)
912    end
913   else
914    __tag_log ("INFO PARENTTREE-NO-DATA: page "..page,3)
915    numsentry = pdfpage.." []"
916   end
917   return numsentry
918 end
919
920 function ltx.__tag.func.output_parenttree (abspage)
921  for i=1,abspage do
922   line = ltx.__tag.func.fill_parent_tree_line (i) .. "^^J"
923   tex.sprint(catlatex,line)
924  end
925 end
```

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

First some local definitions. Since these are only needed locally everything gets wrapped into a block.

```
926 do
927   local properties = node.get_properties_table()
928   local is_soft_hyphen_prop = 'tagpdf.rewrite-softhyphen.is_soft_hyphen'
929   local hyphen_char = 0x2D
930   local soft_hyphen_char = 0xAD
```

A lookup table to test if the font supports the soft hyphen glyph.

```
931   local softhyphen_fonts = setmetatable({}, {__index = function(t, fid)
932    local fdir = identifiers[fid]
933    local format = fdir and fdir.format
934    local result = (format == 'opentype' or format == 'truetype')
935    local characters = fdir and fdir.characters
936    result = result and (characters and characters[soft_hyphen_char]) ~= nil
937    t[fid] = result
938    return result
939   end})
```

A pre shaping callback to mark hyphens as being hyphenation hyphens. This runs before shaping to avoid affecting hyphens moved into discretionaries during shaping.

```
940   local function process_softhyphen_pre(head, _context, _dir)
941    if softhyphenbool.mode ~= truebool.mode then return true end
942    for disc, sub in node.traverse_id(DISC, head) do
943     if sub == explicit_disc or sub == regular_disc then
944      for n, _ch, _f in node.traverse_char(disc.pre) do
945       local props = properties[n]
946       if not props then
947        props = {}
948        properties[n] = props
949       end
950       props[is_soft_hyphen_prop] = true
951      end
952     end
953    end
```

177

```
954    return true
955   end
956
```

Finally do the actual replacement after shaping. No checking for double processing here since the operation is idempotent.

```
957   local function process_softhyphen_post(head, _context, _dir)
958     if softhyphenbool.mode ~= truebool.mode then return true end
959     for disc, sub in node.traverse_id(DISC, head) do
960       for n, ch, fid in node.traverse_glyph(disc.pre) do
961         local props = properties[n]
962         if softhyphen_fonts[fid] and ch == hyphen_char and props and props[is_soft_hyphen_pro
963           if nodegetattribute(n,softhyphenattribute) then
964             n.char = soft_hyphen_char
965             props.glyph_info = nil
966           else
967             nodesetattribute(n,mctypeattributeid,-2147483647)
968             nodesetattribute(n,mccntattributeid,-2147483647)
969           end
970         end
971       end
972     end
973     return true
974   end
975
976   luatexbase.add_to_callback('pre_shaping_filter', process_softhyphen_pre, 'tagpdf.rewrite-
   softhyphen')
977   luatexbase.add_to_callback('post_shaping_filter', process_softhyphen_post, 'tagpdf.rewrite-
   softhyphen')
978 end
```
(*End of definition for* process_softhyphen_pre   process_softhyphen_post. *This function is documented on page* **??**.)

8   parent-child rules

```
979 local function role_get_parent_child_rule (parent,child)
980   local state=
981   ltx.__tag.role.matrix[ltx.__tag.role.index[parent]]
982   and ltx.__tag.role.matrix[ltx.__tag.role.index[parent]][ltx.__tag.role.index[child]] or 0
983   return state
984 end
985 ltx.__tag.func.role_get_parent_child_rule=role_get_parent_child_rule
```
(*End of definition for* role_get_parent_child_rule *and* ltx.__tag.func.role_get_parent_child_rule. *This function is documented on page* **??**.)

These function allows to check the parent-child rules for the current set of structures. It should normally be used at the end of the document. Some stashed structures can still have a parentrole setting containing the STASHED keyword, there must be updated first, this is done with a helper command. To avoid that a faulty structure (where e.g. two structures point to each other) creates an endless loop we check for the real parent only for 10 loops.

```
986 function check_update_stashed (struct,loglevel,loop)
```

178

```
987   loop = (loop or 0)  + 1
988   if loop > 10 then
989     __tag_log ('Warning: Too deeply nested stashed structures',0)
990      return
991   end
992   __tag_log ('updating parentrole for stashed structure '..struct,loglevel)
993   local parent = ltx.__tag.tables['g__tag_struct_'..struct..'_prop']['parentnum']
994   if parent then
995     local ptag =
996       string.match(ltx.__tag.tables['g__tag_struct_'..parent..'_prop']['parentrole'], "{(.-
    )}{(.-)}")
997     if ptag == 'STASHED' then
998     -- look at the parent and update it first
999       check_update_stashed (parent,loglevel,loop)
1000    end
1001    -- now copy the parent role from the parent
1002      ltx.__tag.tables['g__tag_struct_'..struct..'_prop']['parentrole']
1003         =
1004      ltx.__tag.tables['g__tag_struct_'..parent..'_prop']['parentrole']
1005    __tag_log
1006     ('new parentrole: ' .. ltx.__tag.tables['g__tag_struct_'..struct..'_prop']['parentrole'],
1007   else
1008    __tag_log ('Warning: structure '..struct.. 'has no parent.',0)
1009   end
1010  end
1011
1012  function check_parent_child_rules (loglevel)
1013   texio.write_nl('\n')
1014   __tag_log ('checking parent-child rules ...' ,0)
1015   for i=2,ltx.tag.get_struct_counter() do
1016    local t,tNS=
1017      string.match(ltx.__tag.tables['g__tag_struct_'..i..'_prop']['tag'], "{(.-)}{(.-
    )}")
1018    local r,rNS=
1019      string.match(ltx.__tag.tables['g__tag_struct_'..i..'_prop']['rolemap'], "{(.-
    )}{(.-)}")
1020    local p,pNS=
1021      string.match(ltx.__tag.tables['g__tag_struct_'..i..'_prop']['parentrole'], "{(.-
    )}{(.-)}")
1022    local parent=ltx.__tag.tables['g__tag_struct_'..i..'_prop']['parentnum']
1023    if parent then
1024      __tag_log (i..': '.. t..':'..tNS,loglevel)
1025      __tag_log (i..': '.. r..':'..rNS,loglevel)
1026      __tag_log (i..': '.. p..':'..pNS,loglevel)
1027      __tag_log ('parent of ' ..i..': '.. parent,loglevel )
1028      if p == 'STASHED' then
1029       check_update_stashed  (i,loglevel,0)
1030       p,pNS=
1031         string.match(ltx.__tag.tables['g__tag_struct_'..i..'_prop']['parentrole'], "{(.-
    )}{(.-)}")
1032      end
1033      local pt,ptNS=
1034        string.match(ltx.__tag.tables['g__tag_struct_'..parent..'_prop']['tag'], "{(.-
    )}{(.-)}")
```

179

```
1035    local pr,prNS=
1036    string.match(ltx.__tag.tables['g__tag_struct_'..parent..'_prop']['rolemap'], "{(.-
   )}{(.-)}")
1037    local pp,ppNS=
1038    string.match(ltx.__tag.tables['g__tag_struct_'..parent..'_prop']['parentrole'], "{(.-
   )}{(.-)}")
1039    if pp == 'STASHED' then
1040      check_update_stashed  (parent,loglevel,0)
1041      pp,ppNS=
1042       string.match(ltx.__tag.tables['g__tag_struct_'..parent..'_prop']['parentrole'], "{(.-
   )}{(.-)}")
1043    end
1044    __tag_log (parent..': '.. pt..':'..ptNS,loglevel)
1045    __tag_log (parent..': '.. pr..':'..prNS,loglevel)
1046    __tag_log (parent..': '.. pp..':'..ppNS,loglevel)
1047   -- now check the rule.
1048   -- at first rolemap of child against rolemap of parent.
1049    local state=ltx.__tag.func.role_get_parent_child_rule (pr,r)
1050    __tag_log ('rule of '..pr.."->"..r..' is '..state,loglevel)
1051    -- if the state is 7 we check against parentrole of the parent
1052    if state == 7 then
1053     state=ltx.__tag.func.role_get_parent_child_rule (pp,r)
1054     __tag_log ('Parent-Child relation '..pp.."->"..r..' is '..state,loglevel)
1055    end
1056    if state == 0 then
1057      __tag_log
1058      ('Warning: Parent-Child relation '
1059       ..ptNS..':'..pt..' -> '..tNS..':'..t..' is unknown',0)
1060      __tag_log
1061       ('Structure ' ..parent..' -> '..i,0)
1062    end
1063    if state == -1 then
1064     __tag_log
1065      ('Warning: Parent-Child relation '
1066       ..ptNS..':'..pt..' -> '..tNS..':'..t..' is not allowed',0)
1067     __tag_log
1068       ('Structure ' ..parent..' -> '..i,0)
1069    end
1070    -- check also for MC
1071    state =ltx.__tag.func.role_get_parent_child_rule ( r ,'MC')
1072    local curtag=r
1073    if state == 7 then
1074     state =ltx.__tag.func.role_get_parent_child_rule ( p ,'MC')
1075     local curtag=p
1076    end
1077    if state == -1 then
1078     if ltx.__tag.struct[i] and NEXT(ltx.__tag.struct[i]) then
1079       __tag_log
1080       ('Warning: Real content (MC) is not allowed in ' ..curtag,0)
1081     end
1082    end
1083    __tag_log('======================================',loglevel)
1084   end
1085  end -- end for
```

180

```
1086   end
1087
1088  ltx.__tag.func.check_parent_child_rules=check_parent_child_rules
1089
```

(*End of definition for* check_update_stashed, check_parent_child_rules, *and* ltx.__tag.func.check_-parent_child_rules. *These functions are documented on page* **??**.)

9   Link annotations

If the linksplit code has been loaded we use it to add the OBJR of links to the structure tree.

```
1090  local linkbincnt = 0
1091
1092  function ltx.__tag.func.linkbin()
1093   return linkbincnt
1094  end
1095
1096  if luatexbase.callbacktypes['linksplit'] then
1097   luatexbase.add_to_callback('linksplit', function(start_link, position)
1098     if start_link == nil then return end
1099     local structnum =
1100       node.get_attribute(start_link,luatexbase.attributes.g__tag_structnum_attr)
1101     if structnum and structnum > -1 then
1102      local s = ltx.__tag.tables['g__tag_struct_'..structnum..'_prop']['rolemap']
1103      if s and (string.find(s,'Link') or string.find(s,'Reference')) then
1104          local struct_insert_annot_shipout = token.create'__tag_struct_insert_annot_shipout:n
1105          local parentnum = tex.count['c@g__tag_parenttree_obj_int']
1106          start_link.link_attr =
1107            start_link.link_attr ..
1108            ' /LTEX_position /' .. position ..
1109            '/StructParent ' .. parentnum
1110          tex.sprint(catlatex,struct_insert_annot_shipout,'{'..
1111            structnum..'}{'..
1112            start_link.objnum..' 0 R}{'..
1113            parentnum ..'}')
1114          -- the counter must be set explicitly as struct_insert_annot_shipout doesn't do it!
1115          tex.setcount('global','c@g__tag_parenttree_obj_int',parentnum +1)
1116           __tag_log(position .. " link part has object id " .. start_link.objnum .. " and str
1117      else
1118          local linkbin = token.get_macro("c__tag_struct_link_bin_tl")
1119          if linkbin then
1120            local struct_insert_annot_shipout = token.create'__tag_struct_insert_annot_shipout:
1121            local parentnum = tex.count['c@g__tag_parenttree_obj_int']
1122            start_link.link_attr = string.gsub (start_link.link_attr, "/Contents%s+<%w+>","")
1123            start_link.link_attr =
1124              start_link.link_attr ..
1125              ' /LTEX_position /' .. position ..
1126              '/StructParent ' .. parentnum .. '/Contents (artifact)'
1127            tex.sprint(catlatex,struct_insert_annot_shipout,'{'..
1128              linkbin..'}{'..
1129              start_link.objnum..' 0 R}{'..
1130              parentnum ..'}')
1131            tex.setcount('global','c@g__tag_parenttree_obj_int',parentnum +1)
```

```
1132            linkbincnt=linkbincnt+1
1133            if linkbincnt == 1 then
1134              __tag_log('Info: some Link annotation(s) are not in Link or Reference structure el
1135              __tag_log('        moved into a container at the end of the structure tree\n',0)
1136            end
1137          else
1138            __tag_log('Warning: Link not in Link or Reference structure element',0)
1139            __tag_log('OBJR not created',0)
1140            __tag_log('',0)
1141          end
1142       end
1143     end
1144  end, 'tagpdf')
1145 end
```

1146 ⟨/lua⟩

The **tagpdf-roles** module
Tags, roles and namespace code
Part of the tagpdf package

Ulrike Fischer

Part X

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:nnTF` `\tag_check_child:nnTF {⟨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} {2026-03-19} {0.99z}
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.

183

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.

`\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_linked: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 dependent 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

latex-book https://www.latex-project.org/ns/book

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 \__tag_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        \__tag_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        \__tag_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}{}
78  \__tag_role_NS_new:nnn {latex-book} {https://www.latex-project.org/ns/book}{}
79  \exp_args:Nne
80    \__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.

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

```
84       {
85         \cs_new_protected:Npn \__tag_role_alloctag:nnn #1 #2 #3 %#1 tagname, ns, type
86          {
87            \lua_now:e { ltx.__tag.func.alloctag ('#1') }
88            \prop_gput:Nnn \g__tag_role_tags_NS_prop    {#1}{#2}
89            \__tag_prop_gput:cnn {g__tag_role_NS_#2_prop}  {#1}{{}{}}
90            \prop_gput:Nnn \g__tag_role_tags_class_prop {#1}{#3}
91            \prop_gput:cnn {g__tag_role_NS_#2_class_prop}  {#1}{--UNUSED--}
92          }
93       }
94       {
95         \cs_new_protected:Npn \__tag_role_alloctag:nnn #1 #2 #3
96          {
97            \prop_gput:Nnn \g__tag_role_tags_NS_prop    {#1}{#2}
98            \__tag_prop_gput:cnn {g__tag_role_NS_#2_prop}  {#1}{{}{}}
99            \prop_gput:Nnn \g__tag_role_tags_class_prop {#1}{#3}
100           \prop_gput:cnn {g__tag_role_NS_#2_class_prop}  {#1}{--UNUSED--}
101         }
102      }
103    }
104    {
105     \sys_if_engine_luatex:TF
106      {
107        \cs_new_protected:Npn \__tag_role_alloctag:nnn #1 #2 #3 %#1 tagname, ns, type
108         {
109           \lua_now:e { ltx.__tag.func.alloctag ('#1') }
110           \prop_gput:Nnn \g__tag_role_tags_NS_prop    {#1}{#2}
111           \__tag_prop_gput:cnn {g__tag_role_NS_#2_prop}  {#1}{{}{}}
112           \prop_gput:Nnn \g__tag_role_tags_class_prop {#1}{--UNUSED--}
113           \prop_gput:cnn {g__tag_role_NS_#2_class_prop}  {#1}{#3}
114         }
115      }
116      {
117        \cs_new_protected:Npn \__tag_role_alloctag:nnn #1 #2 #3
118         {
119           \prop_gput:Nnn \g__tag_role_tags_NS_prop    {#1}{#2}
120           \__tag_prop_gput:cnn {g__tag_role_NS_#2_prop}  {#1}{{}{}}
121           \prop_gput:Nnn \g__tag_role_tags_class_prop {#1}{--UNUSED--}
122           \prop_gput:cnn {g__tag_role_NS_#2_class_prop}  {#1}{#3}
123         }
124      }
125    }
126  \cs_generate_variant:Nn  \__tag_role_alloctag:nnn {nno}
```
(*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.

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

checks and messages

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

now the addition

```
137     \prop_get:NnNF \g__tag_role_tags_class_prop {#2}\l__tag_tmpa_tl
138       {
139         \tl_set:Nn\l__tag_tmpa_tl{--UNKNOWN--}
140       }
141     \__tag_role_alloctag:nno {#1}{user} { \l__tag_tmpa_tl }
```

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

```
142     \tl_if_empty:nF { #2 }
143       {
144         \prop_get:NnNTF \g__tag_role_rolemap_prop {#2}\l__tag_tmpa_tl
145           {
146             \__tag_prop_gput:Nno \g__tag_role_rolemap_prop {#1}{\l__tag_tmpa_tl}
147           }
148           {
149             \__tag_prop_gput:Nne \g__tag_role_rolemap_prop {#1}{\tl_to_str:n{#2}}
150           }
151       }
152   }
153 \cs_generate_variant:Nn \__tag_role_add_tag:nn {oo,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. Note: this is quite fast and a move to lua doesn't improve speed.

__tag_role_get:nnNN

```
154 \pdf_version_compare:NnT < {2.0}
155 {
156   \cs_new:Npn \__tag_role_get:nnNN #1#2#3#4 %#1 tag, #2 NS, #3 tlvar which hold the role tag
157     {
158       \prop_get:NnNF \g__tag_role_rolemap_prop {#1}#3
159         {
160           \tl_set:Nn #3 {#1}
161         }
162       \tl_set:Nn #4 {}
163     }
164   \cs_generate_variant:Nn \__tag_role_get:nnNN {ooNN}
165 }
166
```
(*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

```
167 \cs_new_protected:Nn \__tag_role_add_tag:nnnn %tag/namespace/role/namespace
168   {
169     \__tag_check_add_tag_role:nnn {#1/#2}{#3}{#4}
170     \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
171       {
172         \msg_info:nnn { tag }{new-tag}{#1}
173       }
174     \prop_if_exist:cTF
175     { g__tag_role_NS_#4_class_prop }
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       }
183     { \tl_set:Nn\l__tag_tmpa_tl{--UNKNOWN--} }
184     \__tag_role_alloctag:nno {#1}{#2}{ \l__tag_tmpa_tl }
```

Do not remap standard tags. TODO add warning?

```
185     \tl_if_in:nnF {-pdf-pdf2-mathml-}{-#2-}
186       {
187         \pdfdict_gput:nne {g__tag_role/RoleMapNS_#2_dict}{#1}
188           {
189             [
190               \pdf_name_from_unicode_e:n{#3}
191               \c_space_tl
192               \pdf_object_ref:n {tag/NS/#4}
193             ]
194           }
195       }
```

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

```
196     \tl_if_empty:nF { #2 }
197       {
198         \prop_get:cnN { g__tag_role_NS_#4_prop } {#3}\l__tag_tmpa_tl
199         \quark_if_no_value:NTF \l__tag_tmpa_tl
200           {
201             \__tag_prop_gput:cne { g__tag_role_NS_#2_prop } {#1}
202               {{\tl_to_str:n{#3}}{\tl_to_str:n{#4}}}
203           }
204           {
205             \__tag_prop_gput:cno { g__tag_role_NS_#2_prop } {#1}{\l__tag_tmpa_tl}
206           }
207       }
```

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

```
208     \bool_if:NT \l__tag_role_update_bool
209       {
210         \tl_if_empty:nF { #3 }
```

190

```
211            {
212              \tl_if_eq:nnF{#1}{#3}
213                {
214                  \prop_get:NnN \g__tag_role_rolemap_prop {#3}\l__tag_tmpa_tl
215                  \quark_if_no_value:NTF \l__tag_tmpa_tl
216                    {
217                      \__tag_prop_gput:Nne \g__tag_role_rolemap_prop {#1}{\tl_to_str:n{#3}}
218                    }
219                    {
220                      \__tag_prop_gput:Nno \g__tag_role_rolemap_prop {#1}{\l__tag_tmpa_tl}
221                    }
222                }
223            }
224        }
225    }
226  \cs_generate_variant:Nn \__tag_role_add_tag:nnnn {oooo}
```
(*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. Note: this is quite fast and a move to lua doesn't improve speed.

__tag_role_get:nnNN

```
227  \pdf_version_compare:NnF < {2.0}
228    {
229      \cs_new:Npn \__tag_role_get:nnNN #1#2#3#4
230        %#1 tag, #2 NS,
231        %#3 tlvar which hold the role tag
232        %#4 tlvar which hold the name of the target NS
233        {
234          \prop_if_exist:cTF {g__tag_role_NS_#2_prop}
235            {
236              \prop_get:cnNTF {g__tag_role_NS_#2_prop} {#1}\l__tag_get_tmpc_tl
237                {
238                  \tl_set:Ne #3 {\exp_last_unbraced:No\use_i:nn  {\l__tag_get_tmpc_tl}}
239                  \tl_set:Ne #4 {\exp_last_unbraced:No\use_ii:nn {\l__tag_get_tmpc_tl}}
240                }
241                {
242                  \msg_warning:nnn { tag } {role-unknown-tag} { #1 }
243                  \tl_set:Nn #3 {#1}
244                  \tl_set:Nn #4 {#2}
245                }
246            }
247            {
248              \msg_warning:nnn { tag } {role-unknown-NS} { #2 }
249              \tl_set:Nn #3 {#1}
250              \tl_set:Nn #4 {#2}
251            }
252        }
253      \cs_generate_variant:Nn \__tag_role_get:nnNN {ooNN}
254    }
```
(*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.

__tag_role_read_namespace_line:nw This command will process a line in the name space file. The first argument is the name of the name space. The definition differ for pdf 2.0. as we have proper name spaces there. With pdf<2.0 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.

```
255 \bool_new:N\l__tag_role_update_bool
256 \bool_set_true:N \l__tag_role_update_bool
257 \pdf_version_compare:NnTF < {2.0}
258 {
259   \cs_new_protected:Npn \__tag_role_read_namespace_line:nw #1#2,#3,#4,#5,#6\q_stop %
260   % #1 NS, #2 tag, #3 rolemap, #4 NS rolemap #5 type
261     {
262       \tl_if_empty:nF { #2 }
263         {
264           \bool_if:NTF \l__tag_role_update_bool
265           {
266             \tl_if_empty:nTF {#5}
267               {
268                 \prop_get:NnN \g__tag_role_tags_class_prop  {#3}\l__tag_tmpa_tl
269                 \quark_if_no_value:NT \l__tag_tmpa_tl
270                   {
271                     \tl_set:Nn\l__tag_tmpa_tl{--UNKNOWN--}
272                   }
273               }
274               {
275                 \tl_set:Nn \l__tag_tmpa_tl {#5}
276               }
277             \__tag_role_alloctag:nno {#2} {#1} { \l__tag_tmpa_tl }
278             \tl_if_eq:nnF {#2}{#3}
279               {
280                 \__tag_role_add_tag:nn {#2}{#3}
281               }
282             \__tag_prop_gput:cnn {g__tag_role_NS_#1_prop}  {#2}{{#3}{}}
283           }
284           {
285             \__tag_prop_gput:cnn {g__tag_role_NS_#1_prop}  {#2}{{#3}{}}
286             \prop_gput:cnn {g__tag_role_NS_#1_class_prop}  {#2}{--UNUSED--}
287           }
288         }
289     }
290 }
291 {
292   \cs_new_protected:Npn \__tag_role_read_namespace_line:nw #1#2,#3,#4,#5,#6\q_stop %
293   % #1 NS, #2 tag, #3 rolemap, #4 NS rolemap #5 type
294     {
295       \tl_if_empty:nF {#2}
296         {
297           \tl_if_empty:nTF {#5}
298             {
299               \prop_get:cnN { g__tag_role_NS_#4_class_prop } {#3}\l__tag_tmpa_tl
300               \quark_if_no_value:NT \l__tag_tmpa_tl
```

192

```
301                     {
302                        \tl_set:Nn\l__tag_tmpa_tl{--UNKNOWN--}
303                     }
304                  }
305                  {
306                     \tl_set:Nn \l__tag_tmpa_tl {#5}
307                  }
308               \__tag_role_alloctag:nno {#2} {#1} { \l__tag_tmpa_tl }
309               \bool_lazy_and:nnT
310                  { ! \tl_if_empty_p:n {#3} }{! \str_if_eq_p:nn {#1}{pdf2}}
311                  {
312                     \__tag_role_add_tag:nnnn {#2}{#1}{#3}{#4}
313                  }
314               \__tag_prop_gput:cnn {g__tag_role_NS_#1_prop}  {#2}{{#3}{#4}}
315            }
316         }
317    }
```

(*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

```
318 \cs_new_protected:Npn \__tag_role_read_namespace:nn #1 #2 %name of namespace #2 name of file
319   {
320     \prop_if_exist:cF {g__tag_role_NS_#1_prop}
321        { \msg_warning:nnn {tag}{namespace-unknown}{#1} }
322     \file_if_exist:nTF { tagpdf-ns-#2.def }
323       {
324         \ior_open:Nn \g_tmpa_ior {tagpdf-ns-#2.def}
325         \msg_info:nnn {tag}{read-namespace}{#2}
326         \ior_map_inline:Nn \g_tmpa_ior
327           {
328             \__tag_role_read_namespace_line:nw {#1} ##1,,,,\q_stop
329           }
330         \ior_close:N\g_tmpa_ior
331       }
332       {
333         \msg_info:nnn{tag}{namespace-missing}{#2}
334       }
335   }
336
```

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

__tag_role_read_namespace:n   This command reads the default namespace file.

```
337 \cs_new_protected:Npn \__tag_role_read_namespace:n #1 %name of namespace
338   {
339     \__tag_role_read_namespace:nn {#1}{#1}
340   }
```

(*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.

```
341 \__tag_role_read_namespace:n {pdf}
```

193

```
342  \__tag_role_read_namespace:n {pdf2}
343  \__tag_role_read_namespace:n {mathml}
```

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

```
344  \bool_set_false:N\l__tag_role_update_bool
345  \__tag_role_read_namespace:n {latex-book}
346  \bool_set_true:N\l__tag_role_update_bool
347  \__tag_role_read_namespace:n {latex}
348  \__tag_role_read_namespace:nn {latex} {latex-lab}
349  \__tag_role_read_namespace:n {pdf}
350  \__tag_role_read_namespace:n {pdf2}
```

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

```
351  \pdf_version_compare:NnTF < {2.0}
352    {
353      \hook_gput_code:nnn {begindocument}{tagpdf}
354        {
355          \bool_lazy_and:nnT
356            {
357              \cs_if_exist_p:N \chapter
358            }
359            {
360              \cs_if_exist_p:N \c@chapter
361            }
362            {
363              \prop_map_inline:cn{g__tag_role_NS_latex-book_prop}
364                {
365                  \__tag_role_add_tag:ne {#1}{\use_i:nn #2\c_empty_tl\c_empty_tl}
366                }
367            }
368        }
369    }
370    {
371      \hook_gput_code:nnn {begindocument}{tagpdf}
372        {
373          \bool_lazy_and:nnT
374            {
375              \cs_if_exist_p:N \chapter
376            }
377            {
378              \cs_if_exist_p:N \c@chapter
379            }
380            {
381              \prop_map_inline:cn{g__tag_role_NS_latex-book_prop}
382                {
383                  \prop_gput:Nnn \g__tag_role_tags_NS_prop    { #1 }{ latex-book }
384                  \__tag_prop_gput:Nne
385                   \g__tag_role_rolemap_prop {#1}{\use_i:nn  #2\c_empty_tl\c_empty_tl}
386                }
387            }
388        }
389    }
```

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.

```
390 \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!

```
391 \sys_if_engine_luatex:TF
392 {
393   \cs_new_protected:Npn \__tag_store_parent_child_rule:nnn #1 #2 #3 % num parent, num child,
394     {
395       \prop_get:NeNTF \c__tag_role_rules_prop{#3} \l__tag_tmp_unused_tl
396         {
397           \intarray_gset:Nnn \g__tag_role_parent_child_intarray
398           { #1#2 }{0\l__tag_tmp_unused_tl}
399           \lua_now:e
400             {
401               ltx.__tag.role.matrix[#1] = ltx.__tag.role.matrix[#1] or {}
402               ltx.__tag.role.matrix[#1][#2] = 0\l__tag_tmp_unused_tl
403             }
404         }
405         {
406           \intarray_gset:Nnn \g__tag_role_parent_child_intarray
407           { #1#2 }{0}
408           \lua_now:e
409             {
410               ltx.__tag.role.matrix[#1] = ltx.__tag.role.matrix[#1] or {}
411               ltx.__tag.role.matrix[#1][#2] = 0
412             }
413         }
414     }
415 }
416 {
417   \cs_new_protected:Npn \__tag_store_parent_child_rule:nnn #1 #2 #3 % num parent, num child,
418     {
419       \prop_get:NeNTF \c__tag_role_rules_prop{#3} \l__tag_tmp_unused_tl
420         {
421           \intarray_gset:Nnn \g__tag_role_parent_child_intarray
422           { #1#2 }{0\l__tag_tmp_unused_tl}
423         }
424         {
425           \intarray_gset:Nnn \g__tag_role_parent_child_intarray
426           { #1#2 }{0}
427         }
428     }
429 }
```

(*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

```
430 \int_zero:N  \l__tag_tmpa_int
```

Open the file depending on the PDF version

```
431 \pdf_version_compare:NnTF < {2.0}
432   {
433     \ior_open:Nn \g_tmpa_ior {tagpdf-parent-child.csv}
434   }
435   {
436     \ior_open:Nn \g_tmpa_ior {tagpdf-parent-child-2.csv}
437   }
```

Now the main loop over the file

```
438 \ior_map_inline:Nn \g_tmpa_ior
439   {
```

ignore lines containing only comments

```
440     \tl_if_empty:nF{#1}
441       {
```

count the lines ...

```
442       \int_incr:N\l__tag_tmpa_int
```

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

```
443       \seq_set_from_clist:Nn\l__tag_tmpa_seq { #1 }
444       \int_compare:nNnTF {\l__tag_tmpa_int}=1
```

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

```
445         {
446           \seq_map_indexed_inline:Nn \l__tag_tmpa_seq
447             {
448               \prop_gput:Nne\g__tag_role_index_prop
449                 {##2}
450                 {\int_compare:nNnT{##1}<{10}{0}##1}
451             }
452         }
```

now the data lines.

```
453         {
454           \seq_set_from_clist:Nn\l__tag_tmpa_seq { #1 }
```

get the name of the child tag from the first column

```
455           \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

```
456           \prop_get:NoN \g__tag_role_index_prop { \l__tag_tmpa_tl } \l__tag_tmpb_tl
```

remove column 2+3

```
457           \seq_pop_left:NN\l__tag_tmpa_seq\l__tag_tmpa_tl
458           \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.

```
459                \seq_map_indexed_inline:Nn \l__tag_tmpa_seq
460                  {
461                    \exp_args:Nne
462                    \__tag_store_parent_child_rule:nnn {##1}{\l__tag_tmpb_tl}{ ##2 }
463                  }
464              }
465          }
466      }
```

close the read handle.

```
467    \ior_close:N\g_tmpa_ior
```

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

```
468    \prop_get:NnN\g__tag_role_index_prop{StructTreeRoot}\l__tag_tmpa_tl
469    \prop_gput:Nne\g__tag_role_index_prop{Root}{\l__tag_tmpa_tl}
470    \prop_get:NnN\g__tag_role_index_prop{Hn}\l__tag_tmpa_tl
471    \pdf_version_compare:NnTF < {2.0}
472      {
473        \int_step_inline:nn{6}
474          {
475            \prop_gput:Nne\g__tag_role_index_prop{H#1}{\l__tag_tmpa_tl}
476          }
477      }
478      {
479        \int_step_inline:nn{10}
480          {
481            \prop_gput:Nne\g__tag_role_index_prop{H#1}{\l__tag_tmpa_tl}
482          }
```

all mathml tags are currently handled identically with the exception of math and mtext

```
483      \prop_get:NnN\g__tag_role_index_prop {mathml}\l__tag_tmpa_tl
484      \prop_get:NnN\g__tag_role_index_prop {math}\l__tag_tmpb_tl
485      \prop_get:NnN\g__tag_role_index_prop {mtext}\l__tag_tmpc_tl
486      \prop_map_inline:Nn \g__tag_role_NS_mathml_prop
487        {
488          \prop_gput:Nno\g__tag_role_index_prop {#1} {\l__tag_tmpa_tl}
489        }
490      \prop_gput:Nno\g__tag_role_index_prop{math}{\l__tag_tmpb_tl}
491      \prop_gput:Nno\g__tag_role_index_prop{mtext}{\l__tag_tmpc_tl}
492    }
493    \sys_if_engine_luatex:T
494    {
495      \prop_map_inline:Nn\g__tag_role_index_prop
496        {
497          \lua_now:e { ltx.__tag.role.index['#1']=#2 }
498        }
499    }
```

1.6.2 Retrieving the parent-child rule

`\__tag_role_get_parent_child_rule:nnN` This command retrieves the rule (as a number) and stores it in the tl-var. It assumes that the tags in #1 and #2 are standard tags after role mapping for which a rule exist.

197

If the parent is one of Part, Div, NonStruct the result can be state 7, which means that a check must be repeated for the "real parent".

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

```
500 \tl_new:N \l__tag_parent_child_check_tl
501 \sys_if_engine_luatex:TF
502   {
503     \cs_new_protected:Npn \__tag_role_get_parent_child_rule:nnN #1 #2 #3
504       % #1 parent (string, standard tag after rolemapping!)
505       % #2 child (string, standard tag after rolemapping!)
506       % #3 tl for state
507       {
508         \tl_set:Ne#3
509           {
510             \lua_now:e{tex.print(\int_use:N\c_document_cctab,ltx.__tag.func.role_get_parent_ch
511           }
```

Debugging messages, this can perhaps go into debug mode.

```
512         \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
513           {
514             \prop_get:NoNF\c__tag_role_rules_num_prop {#3} \l__tag_tmpa_tl
515               {
516                 \tl_set:Nn \l__tag_tmpa_tl {unknown}
517               }
518             \tl_set:Nn \l__tag_tmpb_tl {#1}
519             \msg_note:nneee
520               { tag }
521               { role-parent-child-result }
522               { #1 }
523               { #2 }
524               {
525                 #3~(='\l__tag_tmpa_tl')
526               }
527           }
528         \int_compare:nNnT {#3} = { 0 }
529           {
530             \msg_warning:nneee
531               { tag }
532               {role-parent-child-result}
533               { #1 }
534               { #2 }
535               { unknown! }
536           }
537
538       }
539 }
540   {
541     \cs_new_protected:Npn \__tag_role_get_parent_child_rule:nnN #1 #2 #3
542       % #1 parent (string, standard tag after rolemapping)
543       % #2 child (string, standard tag after rolemapping)
544       % #3 tl for state
545       {
546         \prop_get:NnN \g__tag_role_index_prop{#1}\l__tag_tmpa_tl
547         \prop_get:NnN \g__tag_role_index_prop{#2}\l__tag_tmpb_tl
548         \bool_lazy_and:nnTF
```

```
549                { ! \quark_if_no_value_p:N \l__tag_tmpa_tl }
550                { ! \quark_if_no_value_p:N \l__tag_tmpb_tl }
551                {
```

Get the rule from the intarray

```
552                  \tl_set:Ne#3
553                    {
554                      \intarray_item:Nn
555                      \g__tag_role_parent_child_intarray
556                      {\l__tag_tmpa_tl\l__tag_tmpb_tl}
557                    }
558                }
559                {
560                  \tl_set:Nn#3 {0}
561                }
```

Debugging messages, this can perhaps go into debug mode.

```
562            \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
563              {
564                \prop_get:NoNF\c__tag_role_rules_num_prop {#3} \l__tag_tmpa_tl
565                  {
566                    \tl_set:Nn \l__tag_tmpa_tl {unknown}
567                  }
568                \tl_set:Nn \l__tag_tmpb_tl {#1}
569                \msg_note:nneee
570                  { tag }
571                  { role-parent-child-result }
572                  { #1 }
573                  { #2 }
574                  {
575                    #3~(='\l__tag_tmpa_tl')
576                  }
577              }
578            \int_compare:nNnT {#3} = { 0 }
579              {
580                \msg_warning:nneee
581                  { tag }
582                  {role-parent-child-result}
583                  { #1 }
584                  { #2 }
585                  { unknown! }
586              }
587          }
588      }
589  \cs_generate_variant:Nn\__tag_role_get_parent_child_rule:nnN {ooN}
```

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

__tag_role_check_parent_child:nnnnN This command rolemaps its arguments and then calls __tag_role_get_parent_-
child_rule:nnN to retrieve the parent-child rule between both. It does not try to resolve
inheritance rules of Part, Div and NonStruct but instead gives back the state 7. It is
then the task of the caller command to find the real parent and run the check again. 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.

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

```
591    {
592      \cs_new_protected:Npn \__tag_role_check_parent_child:nnnnN #1 #2 #3 #4 #5
593        % #1 parent tag,% not necessarily rolemapped, but often the case
594        % #2 NS (empty in pdf 1.x)
595        % #3 child tag, % not necessarily rolemapped, but often the case
596        % #4 NS (empty in pdf 1.x)
597        % #5 tl var: to give the result back.
598          {
```

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

```
599            \prop_get:NnNTF \g__tag_role_index_prop {#1}\l__tag_tmpa_tl
600              {
601                \tl_set:Nn \l__tag_tmpa_tl {#1}
602              }
603              {
604                \prop_get:NnNF \g__tag_role_rolemap_prop {#1}\l__tag_tmpa_tl
605                  {
606                    \tl_set:Nn \l__tag_tmpa_tl {\q_no_value}
607                  }
608              }
```

now the child

```
609            \prop_get:NnNTF \g__tag_role_index_prop {#3}\l__tag_tmpb_tl
610              {
611                \tl_set:Nn \l__tag_tmpb_tl {#3}
612              }
613              {
614                \prop_get:NnNF \g__tag_role_rolemap_prop {#3}\l__tag_tmpb_tl
615                  {
616                    \tl_set:Nn \l__tag_tmpb_tl {\q_no_value}
617                  }
618              }
```

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

```
619            \bool_lazy_and:nnTF
620              { ! \quark_if_no_value_p:N \l__tag_tmpa_tl }
621              { ! \quark_if_no_value_p:N \l__tag_tmpb_tl }
622              {
623                \__tag_role_get_parent_child_rule:ooN
624                  { \l__tag_tmpa_tl}
625                  { \l__tag_tmpb_tl}
626                  #5
627              }
628              {
629                \tl_set:Nn #5 {0}
630                \msg_warning:nneee
631                  { tag }
632                  {role-parent-child-result}
633                  { #1 }
634                  { #3 }
635                  { unknown! }
636              }
637          }
638      }
```

and now the pdf 2.0 version

```
639      {
640      \cs_new_protected:Npn \__tag_role_check_parent_child:nnnnN #1 #2 #3 #4 #5 %tag,NS,tag,NS,
641        {
642
```

If the namespace is empty, we assume a standard tag, otherwise we retrieve the rolemapping from the namespace

```
643          \tl_if_empty:nTF  {#2}
644            {
645              \tl_set:Nn \l__tag_tmpa_tl {#1}
646            }
647            {
648              \prop_if_exist:cTF { g__tag_role_NS_#2_prop }
649                {
650                  \prop_get:cnNTF
651                    { g__tag_role_NS_#2_prop }
652                    {#1}
653                    \l__tag_tmpa_tl
654                    {
655                      \tl_set:Ne \l__tag_tmpa_tl {\tl_head:N\l__tag_tmpa_tl}
656                      \tl_if_empty:NT\l__tag_tmpa_tl
657                        {
658                          \tl_set:Nn \l__tag_tmpa_tl {#1}
659                        }
660                    }
661                    {
662                      \tl_set:Nn \l__tag_tmpa_tl {\q_no_value}
663                    }
664                }
665                {
666                  \msg_warning:nnn { tag } {role-unknown-NS} { #2}
667                  \tl_set:Nn \l__tag_tmpa_tl {\q_no_value}
668                }
669            }
```

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

```
670          \tl_if_empty:nTF  {#4}
671            {
672              \tl_set:Nn \l__tag_tmpb_tl {#3}
673            }
674            {
675              \prop_if_exist:cTF { g__tag_role_NS_#4_prop }
676                {
677                  \prop_get:cnNTF
678                    { g__tag_role_NS_#4_prop }
679                    {#3}
680                    \l__tag_tmpb_tl
681                    {
682                      \tl_set:Ne \l__tag_tmpb_tl { \tl_head:N\l__tag_tmpb_tl }
683                      \tl_if_empty:NT\l__tag_tmpb_tl
684                        {
685                          \tl_set:Nn \l__tag_tmpb_tl {#3}
```

```
686                    }
687                  }
688                  {
689                    \tl_set:Nn \l__tag_tmpb_tl {\q_no_value}
690                  }
691                }
692                {
693                  \msg_warning:nnn { tag } {role-unknown-NS} { #4}
694                  \tl_set:Nn \l__tag_tmpb_tl {\q_no_value}
695                }
696            }
```

and now get the relation

```
697            \bool_lazy_and:nnTF
698              { ! \quark_if_no_value_p:N \l__tag_tmpa_tl }
699              { ! \quark_if_no_value_p:N \l__tag_tmpb_tl }
700              {
701                \__tag_role_get_parent_child_rule:ooN
702                  { \l__tag_tmpa_tl }
703                  { \l__tag_tmpb_tl }
704                  #5
705              }
706              {
707                \tl_set:Nn #5 {0}
708                \msg_warning:nneee
709                  { tag }
710                  {role-parent-child-result}
711                  { #2 : #1 }
712                  { #4 : #3 }
713                  { unknown! }
714              }
715        }
716    }
717 \cs_generate_variant:Nn\__tag_role_check_parent_child:nnnnN {oonnN,ooooN}
718 ⟨/package⟩
```

(*End of definition for* __tag_role_check_parent_child:nnnnN.)

\tag_check_child:nn*TF*

```
719 ⟨base⟩\prg_new_protected_conditional:Npnn \tag_check_child:nn #1 #2 {T,F,TF}{\prg_return_true:
720 ⟨*package⟩
721 \prg_set_protected_conditional:Npnn \tag_check_child:nn #1 #2 {T,F,TF} %#1 tag, #2 NS
722   {
723     \seq_get:NN\g__tag_struct_stack_seq\l__tag_tmpa_tl
724     \__tag_struct_get_role:enNN
725       {\l__tag_tmpa_tl}
726       {rolemap}
727       \l__tag_get_parent_tmpa_tl
728       \l__tag_get_parent_tmpb_tl
729     \__tag_role_check_parent_child:oonnN
730       { \l__tag_get_parent_tmpa_tl }
731       { \l__tag_get_parent_tmpb_tl }
732       {#1}{#2}
733       \l__tag_parent_child_check_tl
734     \int_compare:nNnT {\l__tag_parent_child_check_tl} = { \c__tag_role_rule_checkparent_tl }
```

```
735        {
736          \seq_get:NN\g__tag_struct_stack_seq\l__tag_tmpa_tl
737          \__tag_struct_get_role:enNN
738            {\l__tag_tmpa_tl}
739            {parentrole}
740            \l__tag_get_parent_tmpa_tl
741            \l__tag_get_parent_tmpb_tl
742          \__tag_role_check_parent_child:oonnN
743            { \l__tag_get_parent_tmpa_tl }
744            { \l__tag_get_parent_tmpb_tl }
745            {#1}{#2}
746            \l__tag_parent_child_check_tl
747        }
748      \int_compare:nNnTF {  \l__tag_parent_child_check_tl } < {0}
749        {\prg_return_false:}
750        {\prg_return_true:}
751  }
```
(*End of definition for* `\tag_check_child:nnTF`*. This function is documented on page* *.*)

1.7   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.

<div style="float:left">
tag (rolemap-key)
tag-namespace (rolemap-key)
role (rolemap-key)
role-namespace (rolemap-key)
role/new-tag (setup-key)
add-new-tag (deprecated)
</div>

```
752  \keys_define:nn { __tag / tag-role }
753    {
754      ,tag .tl_set:N = \l__tag_role_tag_tmpa_tl
755      ,tag-namespace   .tl_set:N = \l__tag_role_tag_namespace_tmpa_tl
756      ,role .tl_set:N = \l__tag_role_role_tmpa_tl
757      ,role-namespace .tl_set:N = \l__tag_role_role_namespace_tmpa_tl
758    }
759
760  \keys_define:nn { __tag / setup }
761    {
762       role/mathml-tags .bool_gset:N = \g__tag_role_add_mathml_bool
763      ,role/new-tag .code:n =
764      {
765        \keys_set_known:nnnN
766          {__tag/tag-role}
767          {
768            tag-namespace=user,
769            role-namespace=, %so that we can test for it.
770            #1
771          }{__tag/tag-role}\l__tag_tmpa_tl
772        \tl_if_empty:NF \l__tag_tmpa_tl
773          {
774            \exp_args:NNno \seq_set_split:Nnn \l__tag_tmpa_seq { / } {\l__tag_tmpa_tl/}
775            \tl_set:Ne \l__tag_role_tag_tmpa_tl  { \seq_item:Nn \l__tag_tmpa_seq {1} }
776            \tl_set:Ne \l__tag_role_role_tmpa_tl { \seq_item:Nn \l__tag_tmpa_seq {2} }
777          }
778        \tl_if_empty:NT \l__tag_role_role_namespace_tmpa_tl
779          {
780            \prop_get:NoNTF
```

```
781            \g__tag_role_tags_NS_prop
782            { \l__tag_role_role_tmpa_tl }
783            \l__tag_role_role_namespace_tmpa_tl
784            {
785               \prop_get:NoNF
786                 \g__tag_role_NS_prop
787                 { \l__tag_role_role_namespace_tmpa_tl }
788                 \l__tag_tmp_unused_tl
789                 {
790                    \tl_set:Nn \l__tag_role_role_namespace_tmpa_tl {user}
791                 }
792            }
793            {
794               \tl_set:Nn \l__tag_role_role_namespace_tmpa_tl {user}
795            }
796         }
797       \pdf_version_compare:NnTF < {2.0}
798         {
799          %TODO add check for emptyness?
800            \__tag_role_add_tag:oo
801              { \l__tag_role_tag_tmpa_tl }
802              { \l__tag_role_role_tmpa_tl }
803         }
804         {
805           \__tag_role_add_tag:oooo
806             { \l__tag_role_tag_tmpa_tl }
807             { \l__tag_role_tag_namespace_tmpa_tl }
808             { \l__tag_role_role_tmpa_tl }
809             { \l__tag_role_role_namespace_tmpa_tl }
810         }
811     }
812   ,role/map-tags .choice:
813   ,role/map-tags/false .code:n = { \socket_assign_plug:nn { tag/struct/tag } {latex-
   tags} }
814   ,role/map-tags/pdf   .code:n = { \socket_assign_plug:nn { tag/struct/tag } {pdf-
   tags} }

815   ,role/user-NS .code:n =
816     {
817       \pdf_version_compare:NnF < {2.0}
818       {
819         \pdf_string_from_unicode:nnN{utf8/string}{https://www.latex-project.org/ns/local/#1}
820         \tl_if_empty:NF \l__tag_tmpa_str
821           {
822              \pdfdict_gput:nne
823                {g__tag_role/Namespace_user_dict}
824                {NS}
825                {\l__tag_tmpa_str}
826           }
827       }
828     }
```
deprecated names
```
829     , mathml-tags .bool_gset:N = \g__tag_role_add_mathml_bool
830     , add-new-tag .meta:n = {role/new-tag={#1}}
```

₈₃₁   `}`
₈₃₂   ⟨/package⟩

(*End of definition for* `tag (rolemap-key)` *and others. These functions are documented on page* *183*.)

The `tagpdf-space` module
Code related to real space chars
Part of the tagpdf package
    Ulrike Fischer
    Version 0.99z, released 2026-03-19

Part XI

This key allows to activate/deactivate the real space chars if the engine supports it. The allowed values are `true`, `on`, `false`, `off`. The old name of the key `interwordspace` is still supported but deprecated.

This key is deprecated. Use `debug/show=spaces` instead. 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 completely reliable (and change affect other literals and tagging), so it should be used with care.

```
1  ⟨@@=tag⟩
2  ⟨∗header⟩
3  \ProvidesExplPackage {tagpdf-space-code} {2026-03-19} {0.99z}
4   {part of tagpdf - code related to real space chars}
5  ⟨/header⟩
```

1 Code for interword spaces

The code is engine/backend dependent. Basically only pdftex and luatex support real space chars. Most of the code for luatex which uses attributes is in the lua code, here are only the keys.

```
6  ⟨∗package⟩
7  \bool_new:N\l__tag_showspaces_bool
8  \keys_define:nn { __tag / setup }
9   {
10    activate/spaces .choice:,
11    activate/spaces/true .code:n =
12     { \msg_warning:nne {tag}{sys-no-interwordspace}{\c_sys_engine_str}  },
13    activate/spaces/false .code:n=
14     { \msg_warning:nne {tag}{sys-no-interwordspace}{\c_sys_engine_str}  },
15    activate/spaces .default:n = true,
16    debug/show/spaces     .code:n = {\bool_set_true:N \l__tag_showspaces_bool},
17    debug/show/spacesOff  .code:n = {\bool_set_false:N \l__tag_showspaces_bool},
```
deprecated versions:
```
18    interwordspace .choices:nn = {true,on}{\keys_set:nn{__tag/setup}{activate/spaces={true}}}
19    interwordspace .choices:nn = {false,off}{\keys_set:nn{__tag/setup}{activate/spaces={false
20    interwordspace .default:n = {true},
21    show-spaces    .choice:,
22    show-spaces/true .meta:n =  {debug/show=spaces},
23    show-spaces/false .meta:n = {debug/show=spacesOff},
24    show-spaces .default:n = true
25   }
26  \sys_if_engine_pdftex:T
27   {
```

```
28      \sys_if_output_pdf:TF
29        {
30          \pdfglyphtounicode{space}{0020}
31          \AddToHook{shipout/firstpage}[tagpdf/space]{}
32          \keys_define:nn { __tag / setup }
33            {
34              activate/spaces/true .code:n  = { \AddToHook{shipout/firstpage}[tagpdf/space]{\pd
35              activate/spaces/false .code:n = { \RemoveFromHook{shipout/firstpage}[tagpdf/space
36              activate/spaces .default:n = true,
37            }
38        }
39        {
40          \keys_define:nn { __tag / setup }
41            {
42              activate/spaces .choices:nn = { true, false }
43                { \msg_warning:nnn {tag}{sys-no-interwordspace}{dvi}  },
44              activate/spaces .default:n = true,
45            }
46        }
47    }
48
49
50  \sys_if_engine_luatex:T
51    {
52      \keys_define:nn { __tag / setup }
53        {
54          activate/spaces .choice:,
55          activate/spaces/true .code:n =
56                                  {
57                                    \bool_gset_true:N \g__tag_active_space_bool
58                                    \lua_now:e{ltx.__tag.func.markspaceon()}
59                                  },
60          activate/spaces/false .code:n =
61                                  {
62                                    \bool_gset_false:N \g__tag_active_space_bool
63                                    \lua_now:e{ltx.__tag.func.markspaceoff()}
64                                  },
65          activate/spaces .default:n = true,
66          debug/show/spaces    .code:n =
67                                  {\lua_now:e{ltx.__tag.trace.showspaces=true}},
68          debug/show/spacesOff .code:n =
69                                  {\lua_now:e{ltx.__tag.trace.showspaces=nil}},
70        }
71    }
```

(*End of definition for* `activate/spaces (setup-key)`, `interwordspace (deprecated)`, *and* `show-spaces` *(deprecated). These functions are documented on page* **??***.)*

`\__tag_fakespace:`   For luatex we need a command for the fake space as equivalent of the pdftex primitive.

```
72  \sys_if_engine_luatex:T
73    {
74      \cs_new_protected:Nn \__tag_fakespace:
75        {
76          \group_begin:
77          \lua_now:e{ltx.__tag.func.fakespace()}
```

207

```
78        \skip_horizontal:n{\c_zero_skip}
79        \group_end:
80      }
81   }
```

We need also a command to interrupt the insertion of real space chars in places where we want to insert manually special spaces. In pdftex this can be done with \pdfinterwordspaceoff and \pdfinterwordspaceon. These commands insert whatsits and this mean they act globally. In luatex a attribute is used to this effect, for consistency this is also set globally.

The off command sets the attributes in luatex.

```
82  \cs_new_protected:Npn \tag_spacechar_off: {}
83  \cs_new_protected:Npn \tag_spacechar_on: {}
84
85  \sys_if_engine_luatex:T
86    {
87      \cs_set_protected:Npn \tag_spacechar_off:
88        {
89          \lua_now:e
90            {
91              tex.setattribute
92                (
93                  "global",
94                  luatexbase.attributes.g__tag_interwordspaceOff_attr,
95                  1
96                )
97            }
98        }
99      \cs_set_protected:Npn \tag_spacechar_on:
100        {
101          \lua_now:e
102            {
103              tex.setattribute
104                (
105                  "global",
106                  luatexbase.attributes.g__tag_interwordspaceOff_attr,
107                  -2147483647
108                )
109            }
110        }
111  }
112  \sys_if_engine_pdftex:T
113    {
114      \sys_if_output_pdf:T
115        {
116          \cs_set_protected:Npn \tag_spacechar_off:
117            {
118              \pdfinterwordspaceoff
119            }
120          \cs_set_protected:Npn \tag_spacechar_on:
121            {
122              \pdfinterwordspaceon
123            }
124        }
```

208

125   `}`

126   ⟨/package⟩

(*End of definition for* `\__tag_fakespace:`*,* `\tag_spacechar_on:`*, and* `\tag_spacechar_off:`*. These functions are documented on page* **??**.)

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.

214

215

217

218

219

223

224

225