

tagpdf – L^AT_EX kernel code for PDF tagging^{*}

Ulrike Fischer[†]

Released 2025-07-16

Contents

I	7
1 Initialization and test if pdfmanagement is active.	8
2 base package	8
3 Package options	9
4 Packages	9
4.1 a LastPage label	9
5 Variables	9
6 Variants of l3 commands	11
7 Label and Reference commands	11
8 Setup label attributes	12
9 Commands to fill seq and prop	13
10 General tagging commands	13
11 Keys for tagpdfsetup	15
12 loading of engine/more dependent code	16
II	18
1 Commands	18

^{*}This file describes v0.99u, last revised 2025-07-16.

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

2	Description of log messages	18
2.1	\ShowTagging command	18
2.2	Messages in checks and commands	18
2.3	Messages from the ptagging code	19
2.4	Warning messages from the lua-code	19
2.5	Info messages from the lua-code	19
2.6	Debug mode messages and code	20
2.7	Messages	20
3	Messages	21
3.1	Messages related to mc-chunks	21
3.2	Messages related to structures	22
3.3	Attributes	25
3.4	Roles	25
3.5	Miscellaneous	29
4	Retrieving data	29
5	User conditionals	29
6	Internal checks	30
6.1	checks for active tagging	31
6.2	Checks related to structures	31
6.3	Checks related to roles	32
6.4	Check related to mc-chunks	33
6.5	Checks related to the state of MC on a page or in a split stream	36
6.6	Benchmarks	39
III		41
1	Setup commands	41
2	Commands related to mc-chunks	41
3	Commands related to structures	41
4	Debugging	42
5	Extension commands	42
5.1	Fake space	42
5.2	Tagging of paragraphs	43
5.3	Header and footer	43
5.4	Link tagging	44
6	Socket support	44
7	User commands and extensions of document commands	45
8	Setup and preamble commands	45
9	Commands for the mc-chunks	45

10	Commands for the structure	46
11	Socket support	47
12	Debugging	48
13	Commands to extend document commands	52
13.1	Document structure	52
13.2	Structure destinations	53
13.3	Fake space	53
13.4	Paratagging	53
13.5	Language support	60
13.6	Header and footer	60
13.7	Links	64
13.8	Attaching css-files for derivation	68
IV		70
1	Trees, pdfmanagement and finalization code	70
1.1	Check structure	70
1.2	Catalog: MarkInfo and StructTreeRoot and OpenAction	70
1.3	Writing the IDtree	72
1.4	Writing structure elements	73
1.5	ParentTree	74
1.6	Rolemap dictionary	77
1.7	Classmap dictionary	78
1.8	Namespaces	79
1.9	Finishing the structure	79
1.10	StructParents entry for Page	80
V		82
1	Public Commands	82
2	Public keys	83
3	Marked content code – shared	84
3.1	Variables and counters	84
3.2	Functions	86
3.3	Keys	89
VI		91
1	Marked content code – generic mode	91
1.1	Variables	91
1.2	Functions	92
1.3	Looking at MC marks in boxes	95
1.4	Keys	102

VII	104
1 Marked content code – luamode code	104
1.1 Commands	105
1.2 Key definitions	110
VIII	113
1 Public Commands	113
2 Public keys	114
2.1 Keys for the structure commands	114
2.2 Setup keys	116
3 Variables	116
3.1 Variables used by the keys	119
3.2 Variables used by tagging code of basic elements	120
4 Commands	120
4.1 Initialization of the StructTreeRoot	121
4.2 Adding the /ID key	122
4.3 Filling in the tag info	123
4.4 Handlings kids	124
4.5 Output of the object	128
4.6 Commands for the parent-child checks	133
5 Keys	136
6 User commands	145
7 Attributes and attribute classes	154
7.1 Variables	154
7.2 Commands and keys	155
IX	158
1 Loading the lua	158
2 User commands to access data	162
3 Logging functions	163
4 Helper functions	165
4.1 Retrieve data functions	165
4.2 Functions to insert the pdf literals	168
5 Function for the real space chars	170
6 Function for the tagging	174

7	Parenttree	179
8	parent-child rules	181
9	Link annotations	184
X		185
1	Code related to roles and structure names	185
1.1	Variables	185
1.2	Namespaces	188
1.3	Adding a new tag	189
1.3.1	pdf 1.7 and earlier	191
1.3.2	The pdf 2.0 version	192
1.4	Helper command to read the data from files	194
1.5	Reading the default data	196
1.6	Parent-child rules	197
1.6.1	Reading in the csv-files	198
1.6.2	Retrieving the parent-child rule	200
1.7	Key-val user interface	206
XI		209
1	Code for interword spaces	209
Index		213

The **tagpdf** main module
Part of the tagpdf package
Ulrike Fischer
Version 0.99u, released 2025-07-16

Part I

<code>\tag_suspend:n</code>	<code>\tag_suspend:n {⟨label⟩}</code>
<code>\tag_resume:n</code>	<code>\tag_resume:n {⟨label⟩}</code>
<code>\tag_stop:n</code>	<code>\tag_stop:n {⟨label⟩}</code> (<i>deprecated</i>)
<code>\tag_start:n</code>	<code>\tag_start:n {⟨label⟩}</code> (<i>deprecated</i>)

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.

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

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

`activate/tree` (*setup key*) This key activates the code that finalize the various trees. It should be used only in `activate-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. `activate-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 `activate-all` (*deprecated*) (*setup key*) activate tagging.

`activate/struct-dest` (*setup key*) The key allows to suppress the creation of structure destinations

`activate-struct-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.

`activate-tagunmarked` (*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 only is used in `luamode` and replaces hyphens by U+00AD if the font supports this.

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

tagstruct	These are attributes used by the label/ref system.
tagstructobj	
tagabspage	
tagmcabs	
tagmcid	

1 Initialization and test if pdfmanagement is active.

```

1 <@@=tag>
2 <*package>
3 \ProvidesExplPackage {tagpdf} {2025-07-16} {0.99u}
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>
20
21 <*debug>
22 \ProvidesExplPackage {tagpdf-debug} {2025-07-16} {0.99u}
23   { debug code for tagpdf }
24 \AtIfPackageLoaded{tagpdf}{\PackageWarning{tagpdf-debug}{tagpdf~not~loaded,~quitting}\endinput}
25
26 </debug> We map the internal module name “tag” to “tagpdf” in messages.
27 <*package>
28 \prop_gput:Nnn \g_msg_module_name_prop { tag }{ tagpdf }
29 </package>
30
31 Debug mode has its special mapping:
32 <*debug>
33 \prop_gput:Nnn \g_msg_module_type_prop { tag / debug } {}
34 \prop_gput:Nnn \g_msg_module_name_prop { tag / debug }{tagpdf-DEBUG}
35 </debug>

```

2 base package

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

```

36 <*base>
37 \ProvidesExplPackage {tagpdf-base} {2025-07-16} {0.99u}
38   {part of tagpdf - provide base, no-op versions of the user commands }
39 </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.

```

56 <*package>
57 \AddToHook{enddocument/afterlastpage}
58 {\property_record:nn{@tag@LastPage}{abspage,tagmcabs,tagstruct}}

```

5 Variables

```

\l__tag_tmpa_tl A few temporary variables
\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

```

```

63 \tl_new:N \l__tag_Ref_tmpa_tl
64 \tl_new:N \l__tag_get_tmpc_tl
65 \tl_new:N \l__tag_get_parent_tmpa_tl
66 \tl_new:N \l__tag_get_parent_tmpb_tl
67 \tl_new:N \l__tag_get_parent_tmpc_tl
68 \tl_new:N \l__tag_get_child_tmpa_tl
69 \tl_new:N \l__tag_get_child_tmpb_tl
70 \tl_new:N \l__tag_get_child_tmpc_tl
71 \str_new:N \l__tag_tmpa_str
72 \prop_new:N \l__tag_tmpa_prop
73 \seq_new:N \l__tag_tmpa_seq
74 \seq_new:N \l__tag_tmpb_seq
75 \clist_new:N \l__tag_tmpa_clist
76 \int_new:N \l__tag_tmpa_int
77 \box_new:N \l__tag_tmpa_box
78 \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
79 \clist_const:Nn \c__tag_property_mc_clist {tagabspage,tagmcabs,tagmcid}
80 \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.

```

81 \int_new:N \l__tag_loglevel_int

```

(End of definition for \l__tag_loglevel_int.)

`\g__tag_active_space_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.

```

82 \bool_new:N \g__tag_active_space_bool
83 \bool_new:N \g__tag_active_mc_bool
84 \bool_new:N \g__tag_active_tree_bool
85 \bool_new:N \g__tag_active_struct_bool
86 \bool_new:N \g__tag_active_struct_dest_bool
87 \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` 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.
`\l__tag_active_struct_bool`
`\l__tag_active_socket_bool` TODO: check if they are used everywhere as needed and as wanted.

```
88 \bool_new:N \l__tag_active_mc_bool
89 \bool_set_true:N \l__tag_active_mc_bool
90 \bool_new:N \l__tag_active_struct_bool
91 \bool_set_true:N \l__tag_active_struct_bool
92 \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 use it in generic mode, but this would create quite a lot of empty artifact mc-chunks.

```
93 \bool_new:N \g__tag_tagunmarked_bool
```

(End of definition for `\g__tag_tagunmarked_bool`.)

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

```
94 \bool_new:N \g__tag_softthyphen_bool
```

(End of definition for `\g__tag_softthyphen_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 a unique name. At every use it should be increased

```
95 \int_new:N \g__tag_unique_cnt_int
```

(End of definition for `\g__tag_unique_cnt_int`.)

6 Variants of l3 commands

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

7 Label and Reference commands

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

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

```

107 \cs_new_protected:Npn \__tag_property_record:nn #1#2
108 {
109   \@bsphack
110   \property_record:nn{#1}{#2}
111   \@esphack
112 }
113

```

And a few variants

```

114 \cs_generate_variant:Nn \property_ref:nnn {enn}
115 \cs_generate_variant:Nn \property_ref:nn {en}
116 \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.

```

117 \cs_new:Npn \__tag_property_ref_lastpage:nn #1 #2
118 {
119   \property_ref:nnn {@tag@LastPage}{#1}{#2}
120 }

```

(End of definition for __tag_property_ref_lastpage:nn.)

8 Setup label attributes

tagstruct 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**.

```

121 \property_new:nnnn
122 { tagstruct } { now }
123 {1} { \int_use:N \c@g__tag_struct_abs_int }
124 \property_new:nnnn { tagstructobj } { now } {}
125 {
126   \pdf_object_ref_indexed:nn { __tag/struct } { \c@g__tag_struct_abs_int }
127 }
128 \property_new:nnnn
129 { tagabspage } { shipout }
130 {0} { \int_use:N \g_shipout_readonly_int }
131 \property_new:nnnn { tagmcabs } { now }
132 {0} { \int_use:N \c@g__tag_MCID_abs_int }
133

```

```

134 \flag_new:n { __tag/mcid }
135 \property_new:nnnn {tagmcid} { shipout }
136 {0} { \flag_height:n { __tag/mcid } }
137

```

(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.

```

__tag_prop_new:N
__tag_prop_new_linked:N
__tag_seq_new:N38 \cs_set_eq:NN __tag_prop_new:N \prop_new:N
__tag_prop_gput:Nnn39 \cs_set_eq:NN __tag_prop_new_linked:N \prop_new_linked:N
__tag_seq_gput_right:Nn40 \cs_set_eq:NN __tag_seq_new:N \seq_new:N
__tag_seq_item:cn41 \cs_set_eq:NN __tag_prop_gput:Nnn \prop_gput:Nnn
__tag_prop_item:cn42 \cs_set_eq:NN __tag_seq_gput_right:Nn \seq_gput_right:Nn
__tag_seq_show:N43 \cs_set_eq:NN __tag_seq_gput_left:Nn \seq_gput_left:Nn
__tag_prop_show:N44 \cs_set_eq:NN __tag_seq_item:cn \seq_item:cn
45 \cs_set_eq:NN __tag_prop_item:cn \prop_item:cn
46 \cs_set_eq:NN __tag_seq_show:N \seq_show:N
47 \cs_set_eq:NN __tag_prop_show:N \prop_show:N
48 % cnx temporary needed for latex-lab-graphic code
49 \cs_generate_variant:Nn __tag_prop_gput:Nnn { Nen, Nee, Nne, Nno, cnn, cen, cne, cno, cnx }
50 \cs_generate_variant:Nn __tag_seq_gput_right:Nn { Ne , No, cn, ce }
51 \cs_generate_variant:Nn __tag_seq_gput_left:Nn { ce }
52 \cs_generate_variant:Nn __tag_prop_new:N { c }
53 \cs_generate_variant:Nn __tag_seq_new:N { c }
54 \cs_generate_variant:Nn __tag_seq_show:N { c }
55 \cs_generate_variant:Nn __tag_prop_show:N { c }
56 \</package>

```

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

10 General tagging commands

\tag_suspend: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 **\tag_start:** 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.

```

157 \<*package | debug>
158 \l__tag_tag_stop_int# \int_new:N \l__tag_tag_stop_int

```

```

159 \cs_set_protected:Npn \tag_stop:
160 {
161 <debug> \msg_note:nne {tag / debug }{tag-suspend}{ \int_use:N \l__tag_tag_stop_int }
162 \int_incr:N \l__tag_tag_stop_int
163 \bool_set_false:N \l__tag_active_struct_bool
164 \bool_set_false:N \l__tag_active_mc_bool
165 \bool_set_false:N \l__tag_active_socket_bool
166 \__tag_stop_para_ints:
167 }
168 \cs_set_protected:Npn \tag_start:
169 {
170 \int_if_zero:nF { \l__tag_tag_stop_int } { \int_decr:N \l__tag_tag_stop_int }
171 \int_if_zero:nT { \l__tag_tag_stop_int }
172 {
173 \bool_set_true:N \l__tag_active_struct_bool
174 \bool_set_true:N \l__tag_active_mc_bool
175 \bool_set_true:N \l__tag_active_socket_bool
176 \__tag_start_para_ints:
177 }
178 <debug> \msg_note:nne {tag / debug }{tag-resume}{ \int_use:N \l__tag_tag_stop_int }
179 }
180 \cs_set_eq:NN\tagstop\tag_stop:
181 \cs_set_eq:NN\tagstart\tag_start:

182 \cs_set_protected:Npn \tag_suspend:n #1
183 {
184 <debug> \msg_note:nnee {tag / debug }{tag-suspend}
185 <debug> { \int_use:N \l__tag_tag_stop_int }{\exp_not:n{#1}}
186 \int_incr:N \l__tag_tag_stop_int
187 \bool_set_false:N \l__tag_active_struct_bool
188 \bool_set_false:N \l__tag_active_mc_bool
189 \bool_set_false:N \l__tag_active_socket_bool
190 \__tag_stop_para_ints:
191 }
192 \cs_set_eq:NN \tag_stop:n \tag_suspend:n
193 \cs_set_protected:Npn \tag_resume:n #1
194 {
195 \int_if_zero:nF { \l__tag_tag_stop_int } { \int_decr:N \l__tag_tag_stop_int }
196 \int_if_zero:nT { \l__tag_tag_stop_int }
197 {
198 \bool_set_true:N \l__tag_active_struct_bool
199 \bool_set_true:N \l__tag_active_mc_bool
200 \bool_set_true:N \l__tag_active_socket_bool
201 \__tag_start_para_ints:
202 }
203 <debug> \msg_note:nnee {tag / debug }{tag-resume}
204 <debug> { \int_use:N \l__tag_tag_stop_int }{\exp_not:n{#1}}
205 }
206 \cs_set_eq:NN \tag_start:n \tag_resume:n
207 </package | debug>
208 <*base>
209 \cs_new_protected:Npn \tag_stop: {}
210 \cs_new_protected:Npn \tag_start: {}
211 \cs_new_protected:Npn \tagstop {}

```

```

212 \cs_new_protected:Npn \tagstart{}
213 \cs_new_protected:Npn \tag_stop:n #1 {}
214 \cs_new_protected:Npn \tag_start:n #1 {}

```

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

```

215 \cs_set_eq:NN \tag_suspend:n \tag_stop:n
216 \cs_set_eq:NN \tag_resume:n \tag_start:n
217 </base>

```

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

11 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 needed for interword spaces. It is defined in tagpdf-space. `activate/struct-dest` allows to activate or suppress structure destinations.

```

activate/all (setup key)
activate/struct-dest (setup key) <*package>
219 \keys_define:nn { __tag / setup }
220 {
221   activate/mc      .bool_gset:N = \g__tag_active_mc_bool,
222   activate/tree    .bool_gset:N = \g__tag_active_tree_bool,
223   activate/struct .bool_gset:N = \g__tag_active_struct_bool,
224   activate/all     .meta:n =
225     {activate/mc={#1},activate/tree={#1},activate/struct={#1}},
226   activate/all     .default:n = true,
227   activate/struct-dest .bool_gset:N = \g__tag_active_struct_dest_bool,

```

old, deprecated names

```

228   activate-mc      .bool_gset:N = \g__tag_active_mc_bool,
229   activate-tree    .bool_gset:N = \g__tag_active_tree_bool,
230   activate-struct .bool_gset:N = \g__tag_active_struct_bool,
231   activate-all     .meta:n =
232     {activate/mc={#1},activate/tree={#1},activate/struct={#1}},
233   activate-all     .default:n = true,
234   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.

```

235   debug/show      .choice:,

```

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

```

debug/uncompress (setup key)
log (deprecated) (setup key)
ompress (deprecated) (setup key)
237   debug/log      .choice:,
238   debug/log / none .code:n = {\int_set:Nn \l__tag_loglevel_int { 0 }},
239   debug/log / v   .code:n =
240     {

```

```

240     \int_set:Nn \l__tag_loglevel_int { 1 }
241     \cs_set_protected:Nn \__tag_check_typeout_v:n { \iow_term:e {##1} }
242   },
243   debug/log / vv      .code:n = {\int_set:Nn \l__tag_loglevel_int { 2 }},
244   debug/log / vvv     .code:n = {\int_set:Nn \l__tag_loglevel_int { 3 }},
245   debug/log / all     .code:n = {\int_set:Nn \l__tag_loglevel_int { 10 }},
246   debug/uncompress .code:n = { \pdf_uncompress: },

```

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

```

247   log      .meta:n = {debug/log={#1}},
248   uncompress .code:n = { \pdf_uncompress: },

```

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

```

249   activate/tagunmarked .bool_gset:N = \g__tag_tagunmarked_bool,
250   activate/tagunmarked .initial:n = true,

```

deprecated name

```

251   tagunmarked .bool_gset:N = \g__tag_tagunmarked_bool,

```

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

```

252   activate/softhyphen .bool_gset:N = \g__tag_softhyphen_bool,
253   activate/softhyphen .initial:n = true,

```

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

```

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

```

deprecated name

```

264   tabsorder .meta:n = {page/tabsorder={#1}},
265 }

```

12 loading of engine/more dependent code

```

266 \sys_if_engine luatex:T
267 {

```



```

268     \file_input:n {tagpdf-luatex.def}
269   }
270 </package>

271 <*mcloading>
272 \bool_if:NTF \g__tag_mode_lua_bool
273   {
274     \RequirePackage {tagpdf-mc-code-lua}
275   }
276   {
277     \RequirePackage {tagpdf-mc-code-generic} %
278   }
279 </mcloading>
280 <*debug>
281 \bool_if:NTF \g__tag_mode_lua_bool
282   {
283     \RequirePackage {tagpdf-debug-lua}
284   }
285   {
286     \RequirePackage {tagpdf-debug-generic} %
287   }
288 </debug>

```

The tagpdf-checks module
 Messages and check code
 Part of the tagpdf package
 Ulrike Fischer
 Version 0.99u, released 2025-07-16

Part II

1 Commands

`\tag_if_active_p: *` This command tests if tagging is active. It only gives true if all tagging has been activated,
`\tag_if_active:TF *` *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 <box> {<true code>} {<false code>}`

`\tag_if_box_tagged:NTF *` 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
<code>\ShowTaggingmc-data = num</code>	log+term	lua-only
<code>\ShowTaggingmc-current</code>	log+term	
<code>\ShowTaggingstruck-stack= [log show]</code>	log or term+stop	
<code>\ShowTaggingdebug/structures = num</code>	log+termn	debug mode only

2.2 Messages in checks and commands

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

2.3 Messages from the ptagging code

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

2.4 Warning messages from the lua-code

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

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

2.5 Info messages from the lua-code

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

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

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

2.6 Debug mode messages and code

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

command	name	action	remark
<code>\tag_mc_begin:n</code>	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

<code>show-struct</code>	These two messages are used in debug mode to show the current structures in the log and terminal.
<code>show-kids</code>	

<code>attr-unknown</code>	Message if an attribute is unknown.
---------------------------	-------------------------------------

<code>role-missing</code>	Messages related to role mapping.
<code>role-unknown</code>	
<code>role-unknown-tag</code>	
<code>role-unknown-NS</code>	
<code>role-tag</code>	
<code>new-tag</code>	
<code>role-parent-child-result</code>	
<code>role-remapping</code>	

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

<code>sys-no-interwordspace</code>	Message if an engine doesn't support inter word spaces
------------------------------------	--

<code>para-hook-count-wrong</code>	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} {2025-07-16} {0.99u}
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 issued if 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 20.)

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 20.)

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 20.)

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 20.)

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 20.)

mc-pushed Informational messages about mc-pushing.

mc-popped

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

(End of definition for mc-pushed and mc-popped. These functions are documented on page 20.)

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 20.)

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 20.)

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 20.)

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 20.)

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 20.)

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 20.)

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 20.)

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 20.)

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 20.)

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 20.)

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 20.)

The following messages are only needed in debug mode.

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

```

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   =====
82 }
83 </debug>
```

(End of definition for `show-struct` and `show-kids`. These functions are documented on page 21.)

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 21.)

3.4 Roles

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

role-unknown

```
role-unknown-tag86 \msg_new:nnn { tag } {role-missing} { tag~#1~has~no~role~assigned }
role-unknown-NS87 \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} }
```

(End of definition for role-missing and others. These functions are documented on page 21.)

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 21.)

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 ??.)

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_t1
145 \prop_get:cnN {g__tag_struct_#3_prop}{rolemap}\l__tag_get_child_tmpc_t1
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_t1 }
153 :
154 \exp_last_unbraced:No \use_i:nn { \l__tag_get_parent_tmpc_t1 }
155 }
156 {
157 \exp_last_unbraced:No \use_ii:nn { \l__tag_get_child_tmpc_t1 }
158 :
159 \exp_last_unbraced:No \use_i:nn { \l__tag_get_child_tmpc_t1 }
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_t1}
174 {
175 \msg_warning:nneeee
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 21.)

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 21.)

3.5 Miscellaneous

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

```
220 \msg_new:nnn { tag } {tree-mcid-index-wrong}
221   {something-is-wrong-with-the-mcid--rerun}
```

(End of definition for tree-mcid-index-wrong. This function is documented on page 21.)

sys-no-interwordspace Currently only pdf_latex and lua_latex have some support for real spaces.

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

(End of definition for sys-no-interwordspace. This function is documented on page 21.)

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

```
224 \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 breaks the structure.

```
225 \msg_new:nnnn { tag } {para-hook-count-wrong}
226   {The-number-of-automatic-begin-(#1)-and-end-(#2)-#3~para-hooks-differ!}
227   {This-quite-probably-a-coding-error-and-the-structure-will-be-wrong!}
228 \</package>
```

(End of definition for para-hook-count-wrong. This function is documented on page 21.)

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.

```
229 <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 18.)

5 User conditionals

\tag_if_active_p: 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.
\tag_if_active:TF

```
230 <*base>
231 \cs_if_exist:NF\tag_if_active:T
232 {
233   \prg_new_conditional:Npnn \tag_if_active: { p , T , TF , F }
234   { \prg_return_false: }
235 }
```

```

236 </base>
237 <*package>
238 \prg_set_conditional:Npnn \tag_if_active: { p , T , TF , F }
239 {
240     \bool_lazy_all:nTF
241     {
242         {\g__tag_active_struct_bool}
243         {\g__tag_active_mc_bool}
244         {\g__tag_active_tree_bool}
245         {\l__tag_active_struct_bool}
246         {\l__tag_active_mc_bool}
247     }
248     {
249         \prg_return_true:
250     }
251     {
252         \prg_return_false:
253     }
254 }
255 </package>

```

(End of definition for \tag_if_active:TF. This function is documented on page 18.)

\tag_if_box_tagged_p:N 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.

\tag_if_box_tagged:NTF

```

256 <*base>
257 \prg_new_conditional:Npnn \tag_if_box_tagged:N #1 {p,T,F,TF}
258 {
259     \tl_if_exist:cTF {l_tag_box_\int_use:N #1_tl}
260     {
261         \int_compare:nNnTF {0\int_use:c{l_tag_box_\int_use:N #1_tl}}>{0}
262         { \prg_return_true: }
263         { \prg_return_false: }
264     }
265     {
266         \prg_return_false:
267         % warning??
268     }
269 }
270 </base>

```

(End of definition for \tag_if_box_tagged:NTF. This function is documented on page 18.)

6 Internal checks

These are checks used in various places in the code.

6.1 checks for active tagging

`__tag_check_if_active_mc:TF` This checks if mc are active.

`__tag_check_if_active_struct:TF`

```

271 (*package)
272 \prg_new_conditional:Npnn __tag_check_if_active_mc: {T,F,TF}
273 {
274   \bool_lazy_and:nnTF { \g__tag_active_mc_bool } { \l__tag_active_mc_bool }
275   {
276     \prg_return_true:
277   }
278   {
279     \prg_return_false:
280   }
281 }
282 \prg_new_conditional:Npnn __tag_check_if_active_struct: {T,F,TF}
283 {
284   \bool_lazy_and:nnTF { \g__tag_active_struct_bool } { \l__tag_active_struct_bool }
285   {
286     \prg_return_true:
287   }
288   {
289     \prg_return_false:
290   }
291 }
```

(End of definition for __tag_check_if_active_mc:TF and __tag_check_if_active_struct:TF.)

6.2 Checks related to structures

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

```

292 \cs_new_protected:Npn __tag_check_structure_has_tag:n #1 %#1 struct num
293 {
294   \prop_get:cnNF
295     { g__tag_struct_#1_prop }
296     {S}
297     \l__tag_tmp_unused_tl
298     {
299       \msg_error:nn { tag } {struct-missing-tag}
300     }
301 }
```

(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.

```

302 \cs_new_protected:Npn __tag_check_structure_tag:N #1
303 {
304   \prop_get:NoNF \g__tag_role_tags_NS_prop {#1}\l__tag_tmp_unused_tl
305   {
```

```

306         \msg_warning:nne { tag } {role-unknown-tag} {#1}
307     }
308 }

```

(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.

```

309 \cs_new_protected:Npn \__tag_check_info_closing_struct:n #1 %#1 struct num
310 {
311     \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
312     {
313         \msg_info:nnn { tag } {struct-show-closing} {#1}
314     }
315 }
316
317 \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.

```

318 \cs_new_protected:Npn \__tag_check_no_open_struct:
319 {
320     \msg_error:nn { tag } {struct-faulty-nesting}
321 }

```

(End of definition for __tag_check_no_open_struct:.)

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

```

322 \cs_new_protected:Npn \__tag_check_struct_used:n #1 %#1 label
323 {
324     \prop_get:cnNT
325     {g__tag_struct\_property_ref:enn{tagpdfstruct-#1}{tagstruct}{unknown}_prop}
326     {parentnum}
327     \l__tag_tmpa_tl
328     {
329         \msg_warning:nnn { tag } {struct-used-twice} {#1}
330     }
331 }

```

(End of definition for __tag_check_struct_used:n.)

6.3 Checks related to roles

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

```

332 \cs_new_protected:Npn \__tag_check_add_tag_role:nn #1 #2 %#1 tag, #2 role
333 {
334     \tl_if_empty:nTF {#2}
335     {
336         \msg_error:nnn { tag } {role-missing} {#1}

```



```

337     }
338     {
339         \prop_get:NnNTF \g__tag_role_tags_NS_prop {#2} \l__tag_tmpa_tl
340         {
341             \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
342             {
343                 \msg_info:nnnn { tag } {role-tag} {#1} {#2}
344             }
345         }
346         {
347             \msg_error:nnn { tag } {role-unknown} {#2}
348         }
349     }
350 }

```

Similar with a namespace

```

351 \cs_new_protected:Npn \__tag_check_add_tag_role:nnn #1 #2 #3 %#1 tag/NS, #2 role #3 namespace
352 {
353     \tl_if_empty:nTF {#2}
354     {
355         \msg_error:nnn { tag } {role-missing} {#1}
356     }
357     {
358         \prop_get:cnNTF { g__tag_role_NS_#3_prop } {#2} \l__tag_tmpa_tl
359         {
360             \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
361             {
362                 \msg_info:nnnn { tag } {role-tag} {#1} {#2/#3}
363             }
364         }
365         {
366             \msg_error:nnn { tag } {role-unknown} {#2/#3}
367         }
368     }
369 }

```

(End of definition for __tag_check_add_tag_role:nn.)

6.4 Check related to mc-chunks

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

```

370 \cs_new_protected:Npn \__tag_check_mc_if_nested:
371 {
372     \__tag_mc_if_in:T
373     {
374         \msg_warning:nne { tag } {mc-nested} { \__tag_get_mc_abs_cnt: }
375     }
376 }
377
378 \cs_new_protected:Npn \__tag_check_mc_if_open:
379 {

```

```

380 \tag_mc_if_in:F
381 {
382   \msg_warning:nne { tag } {mc-not-open} { \tag_get_mc_abs_cnt: }
383 }
384 }

```

(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.

```

385 \cs_new_protected:Npn \tag_check_mc_pushed_popped:nn #1 #2
386 {
387   \int_compare:nNnT
388     { \l_tag_loglevel_int } = { 2 }
389     { \msg_info:nne {tag}{mc-#1}{#2} }
390   \int_compare:nNnT
391     { \l_tag_loglevel_int } > { 2 }
392     {
393       \msg_info:nne {tag}{mc-#1}{#2}
394       \seq_log:N \g_tag_mc_stack_seq
395     }
396 }

```

(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.

```

397 \cs_new_protected:Npn \tag_check_mc_tag:N #1 %#1 is var with a tag name in it
398 {
399   \tl_if_empty:NTF #1
400   {
401     \tl_set:No #1 { \g_tag_struct_tag_tl }
402     \msg_info:nnee { tag } {mc-tag-missing} { \g_tag_struct_tag_tl } { \tag_get_mc_abs_cnt: }
403   }
404   {
405     \prop_get:NoNF \g_tag_role_tags_NS_prop {#1}\l_tag_tmp_unused_tl
406     {
407       \msg_warning:nne { tag } {role-unknown-tag} {#1}
408     }
409   }
410 }

```

(End of definition for \tag_check_mc_tag:N.)

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

```

411 \cs_new_protected:Npn \__tag_check_init_mc_used:
412 {
413   \intarray_new:Nn \g__tag_check_mc_used_intarray { 65536 }
414   \cs_gset_eq:NN \__tag_check_init_mc_used: \prg_do_nothing:
415 }

```

(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.

```

416 \cs_new_protected:Npn \__tag_check_mc_used:n #1 %#1 mcid absent
417 {
418   \int_compare:nNnT {\l__tag_loglevel_int} > { 2 }
419   {
420     \__tag_check_init_mc_used:
421     \intarray_gset:Nnn \g__tag_check_mc_used_intarray
422       {#1}
423     { \intarray_item:Nn \g__tag_check_mc_used_intarray {#1} + 1 }
424     \int_compare:nNnT
425     {
426       \intarray_item:Nn \g__tag_check_mc_used_intarray {#1}
427     }
428     >
429     { 1 }
430     {
431       \msg_warning:nnn { tag } {mc-used-twice} {#1}
432     }
433   }
434 }

```

(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.

```

435 \cs_new_protected:Npn \__tag_check_show_MCID_by_page:
436 {
437   \tl_set:Nc \l__tag_tmpa_tl
438   {
439     \__tag_property_ref_lastpage:nn
440     {abspage}
441     {-1}
442   }
443   \int_step_inline:nnnn {1}{1}
444   {
445     \l__tag_tmpa_tl
446   }
447   {
448     \seq_clear:N \l__tag_tmpa_seq
449     \int_step_inline:nnnn
450     {1}
451     {1}
452     {
453       \__tag_property_ref_lastpage:nn
454       {tagmcabs}
455       {-1}

```

```

456     }
457     {
458         \int_compare:nT
459         {
460             \property_ref:enn
461             {mcid-###1}
462             {tagabspage}
463             {-1}
464             =
465             ##1
466         }
467         {
468             \seq_gput_right:Ne \l__tag_tmpa_seq
469             {
470                 Page##1-###1-
471                 \property_ref:enn
472                 {mcid-###1}
473                 {tagmcid}
474                 {-1}
475             }
476         }
477     }
478     \seq_show:N \l__tag_tmpa_seq
479 }
480 }

```

(End of definition for `__tag_check_show_MCID_by_page:.`)

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

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

`__tag_check_mc_in_galley_p:` 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.

`__tag_check_mc_in_galley:TF`

```

481 \prg_new_conditional:Npnn \__tag_check_if_mc_in_galley: { T,F,TF }
482 {
483     \tl_if_eq:NNTF \l__tag_mc_firstmarks_seq \l__tag_mc_botmarks_seq
484     { \prg_return_false: }
485     { \prg_return_true: }
486 }

```

(End of definition for `__tag_check_mc_in_galley:TF`.)

`__tag_check_if_mc_tmb_missing_p:` 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.

`__tag_check_if_mc_tmb_missing:TF`

```

487 \prg_new_conditional:Npnn \__tag_check_if_mc_tmb_missing: { T,F,TF }
488 {
489   \bool_if:nTF
490   {
491     \str_if_eq_p:ee {\seq_item:Nn \l__tag_mc_firstmarks_seq {1}}{e-}
492     ||
493     \str_if_eq_p:ee {\seq_item:Nn \l__tag_mc_firstmarks_seq {1}}{b+}
494   }
495   { \prg_return_true: }
496   { \prg_return_false: }
497 }

```

(End of definition for __tag_check_if_mc_tmb_missing:TF.)

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

```

498 \prg_new_conditional:Npnn \__tag_check_if_mc_tme_missing: { T,F,TF }
499 {
500   \str_if_eq:eeTF {\seq_item:Nn \l__tag_mc_botmarks_seq {1}}{b+}
501   { \prg_return_true: }
502   { \prg_return_false: }
503 }

```

(End of definition for __tag_check_if_mc_tme_missing:TF.)

504 </package>

505 <*debug>

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

```

506 \msg_new:nnn { tag / debug } {mc-begin} { MC~begin~#1~with~options:~\tl_to_str:n{#2}~[\msg_line_
507 \msg_new:nnn { tag / debug } {mc-end} { MC~end~#1~[\msg_line_context:] }
508
509 \cs_new_protected:Npn \__tag_debug_mc_begin_insert:n #1
510 {
511   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
512   {
513     \msg_note:nnnn { tag / debug } {mc-begin} {inserted} { #1 }
514   }
515 }
516 \cs_new_protected:Npn \__tag_debug_mc_begin_ignore:n #1
517 {
518   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
519   {
520     \msg_note:nnnn { tag / debug } {mc-begin } {ignored} { #1 }
521   }
522 }
523 \cs_new_protected:Npn \__tag_debug_mc_end_insert:
524 {
525   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
526   {

```

```

527     \msg_note:nnn { tag / debug } {mc-end} {inserted}
528   }
529 }
530 \cs_new_protected:Npn \__tag_debug_mc_end_ignore:
531 {
532   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
533   {
534     \msg_note:nnn { tag / debug } {mc-end} {ignored}
535   }
536 }

```

And now something for the structures

```

537 \msg_new:nnn { tag / debug } {struct-begin}
538 {
539   Struct~\tag_get:n{struct_num}~begin~#1~with~options:~\tl_to_str:n{#2}~\[\msg_line_context:]
540 }
541 \msg_new:nnn { tag / debug } {struct-end}
542 {
543   Struct~end~#1~[\msg_line_context:]
544 }
545 \msg_new:nnn { tag / debug } {struct-end-wrong}
546 {
547   Struct~end~'#1'~doesn't~fit~start~'#2'~[\msg_line_context:]
548 }
549
550 \cs_new_protected:Npn \__tag_debug_struct_begin_insert:n #1
551 {
552   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
553   {
554     \msg_note:nnnn { tag / debug } {struct-begin} {inserted} { #1 }
555     \seq_log:N \g__tag_struct_tag_stack_seq
556   }
557 }
558 \cs_new_protected:Npn \__tag_debug_struct_begin_ignore:n #1
559 {
560   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
561   {
562     \msg_note:nnnn { tag / debug } {struct-begin} {ignored} { #1 }
563   }
564 }
565 \cs_new_protected:Npn \__tag_debug_struct_end_insert:
566 {
567   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
568   {
569     \msg_note:nnn { tag / debug } {struct-end} {inserted}
570     \seq_log:N \g__tag_struct_tag_stack_seq
571   }
572 }
573 \cs_new_protected:Npn \__tag_debug_struct_end_ignore:
574 {
575   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
576   {
577     \msg_note:nnn { tag / debug } {struct-end} {ignored}
578   }

```

```

579 }
580 \cs_new_protected:Npn \__tag_debug_struct_end_check:n #1
581 {
582   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
583   {
584     \seq_get:NNT \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
585     {
586       \str_if_eq:eeF
587       {#1}
588       {\exp_last_unbraced:No \use_i:nn { \l__tag_tmpa_tl }}
589       {
590         \msg_warning:nnee { tag/debug }{ struct-end-wrong }
591         {#1}
592         {\exp_last_unbraced:No \use_i:nn { \l__tag_tmpa_tl }}
593       }
594     }
595   }
596 }

```

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.

```

597 \msg_new:nnn { tag / debug } {tag-suspend}
598 {
599   \int_if_zero:nTF
600   {#1}
601   {Tagging~suspended}
602   {Tagging~(not)~suspended~(already~inactive)}\
603   level:~#1~==>~\int_eval:n{#1+1}\tl_if_empty:nF{#2}{,~label:~#2}~[\msg_line_context:]
604 }
605 \msg_new:nnn { tag / debug } {tag-resume}
606 {
607   \int_if_zero:nTF
608   {#1}
609   {Tagging~resumed}
610   {Tagging~(not)~resumed}\
611   level:~\int_eval:n{#1+1}~==>~#1\tl_if_empty:nF{#2}{,~label:~#2}~[\msg_line_context:]
612 }
613 </debug>

```

6.6 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?

```

614 <*package>
615 \cs_new_protected:Npn \__tag_check_benchmark_tic:{}
616 \cs_new_protected:Npn \__tag_check_benchmark_toc:{}
617 \cs_new_protected:Npn \tag_check_benchmark_on:
618 {
619   \cs_if_exist:NT \benchmark_tic:
620   {
621     \cs_set_eq:NN \__tag_check_benchmark_tic: \benchmark_tic:

```

```

622         \cs_set_eq:NN \__tag_check_benchmark_toc: \benchmark_toc:
623     }
624 }
625 </package>

```

The tagpdf-user module

Code related to L^AT_EX2e user commands and document commands

Part of the tagpdf package

Ulrike Fischer

Version 0.99u, released 2025-07-16

Part III

1 Setup commands

<code>\tagpdfsetup</code>	<code>\tagpdfsetup{<key val list>}</code>
---------------------------	---

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).

<code>activate (setup-key)</code>	And additional setup key which combine the other activate keys <code>activate/mc</code> , <code>activate/tree</code> , <code>activate/struct</code> and additionally adds a document structure.
-----------------------------------	---

<code>\tag_tool:n</code>	<code>\tag_tool:n {<key val>}</code>
--------------------------	--

<code>\tagtool</code>	
-----------------------	--

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

<code>\tagmcbegin</code>	<code>\tagmcbegin{<key-val>}</code>
--------------------------	---

<code>\tagmcend</code>	<code>\tagmcend</code>
------------------------	------------------------

<code>\tagmcuse</code>	<code>\tagmcuse{<label>}</code>
------------------------	---------------------------------------

These are wrappers around `\tag_mc_begin:n`, `\tag_mc_end:` and `\tag_mc_use:n`. The commands and their argument are documented 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`.

<code>\tagmcifinTF</code>	<code>\tagmcifinTF{<true code>}{<false code>}</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

<code>\tagstructbegin</code>	<code>\tagstructbegin{<key-val>}</code>
------------------------------	---

<code>\tagstructend</code>	<code>\tagstructend</code>
----------------------------	----------------------------

<code>\tagstructuse</code>	<code>\tagstructuse{<label>}</code>
----------------------------	---

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` `\ShowTagging{<key-val>}`

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

`mc-data` (`show-key`) `mc-data = <number>`

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

`mc-current` (`show-key`) `mc-current`

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

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

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

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

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

`debug/structures` (`show-key`) `debug/structures = <structure number>`

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

5 Extension commands

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

The commands and keys should be view as experimental!

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

5.1 Fake space

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

5.2 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.

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

The `para/tagging` key can be used in `\tagpdfsetup` and enable/disable tagging of paragraphs. `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.

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

<code>\tagpdfsuppressmarks</code>	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.

<code>page/exclude-header-footer</code> (setup-key)	<code>page/exclude-header-footer = true false pagination</code>
---	---

5.4 Link tagging

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

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

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

6 Socket support

\tag_socket_use:n	\tag_socket_use:n {<socket name>}
\tag_socket_use:nnn	\tag_socket_use:nn {<socket name>} {<socket argument>}
\UseTaggingSocket	\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 it expects a socket starting with `tagsupport/` but the socket name is specified without this prefix, i.e.,

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

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 command. 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} {2025-07-16} {0.99u}
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 41.)

`\tag_tool:n` This is a first definition of the tool command. Currently it uses key-val, but this should
`\tagtool` 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 41.)

9 Commands for the mc-chunks

```

\tagmcbegin
\tagmcend
\tagmcuse22 <*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 41.)

\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 pdf_lat_ex as lua_lat_ex 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 41.)

10 Commands for the structure

\tagstructbegin

\tagstructend

\tagstructuse

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

```

45 \<base>
46 \NewDocumentCommand \tagstructbegin { m }
47 {
48   \tag_struct_begin:n {#1}
49 }
50
51 \NewDocumentCommand \tagstructend { }
52 {
53   \tag_struct_end:
54 }
55
56 \NewDocumentCommand \tagstructuse { m }
57 {
58   \tag_struct_use:n {#1}
59 }
60 \</base>

```

(End of definition for \tagstructbegin, \tagstructend, and \tagstructuse. These functions are documented on page 41.)

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:nnn72 <*package>
\UseTaggingSocket73 \cs_set_protected:Npn \tag_socket_use:n #1
\tag_socket_use_expandable:n74 {
\UseExpandableTaggingSocket75   \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 44.)

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 42.)

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 42.)

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
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 42.)

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 42.)

struct-stack (show-key)

```

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 42.)

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 42.)

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

```

\g__tag_root_default_tl
  activate (setup-key)
activate/socket (setup-key) 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 41.)

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 }
```

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 42.)

13.4 Paratagging

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

```

\l__tag_para_bool    At first some variables.
\l__tag_para_flattened_bool
\l__tag_para_show_bool </package>
\g__tag_para_begin_int <base>\bool_new:N \l__tag_para_flattened_bool
\g__tag_para_end_int <base>\bool_new:N \l__tag_para_bool
\g__tag_para_main_begin_int <*package>
\g__tag_para_main_end_int \int_new:N \g__tag_para_begin_int
\g__tag_para_main_struct_tl \int_new:N \g__tag_para_end_int
\l__tag_para_tag_default_tl \int_new:N \g__tag_para_main_begin_int
\l__tag_para_tag_tl \int_new:N \g__tag_para_main_end_int
\l__tag_para_main_tag_tl
\l__tag_para_attr_class_tl
\l__tag_para_main_attr_class_tl
```

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_set:Nn \l__tag_para_tag_default_tl { text }
334 \tl_new:N \l__tag_para_tag_tl
335 \tl_set:Nn \l__tag_para_tag_tl { \l__tag_para_tag_default_tl }
336 \tl_new:N \l__tag_para_main_tag_tl
337 \tl_set:Nn \l__tag_para_main_tag_tl {text-unit}

```

this is perhaps already defined by the block code

```

338 \tl_if_exist:NF \l__tag_para_attr_class_tl
339 {\tl_new:N \l__tag_para_attr_class_tl }
340 \tl_new:N \l__tag_para_main_attr_class_tl

```

(End of definition for \l__tag_para_bool and others.)

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

```

\__tag_gincr_para_main_end_int:
\__tag_gincr_para_end_int:
341 \cs_new_protected:Npn \__tag_gincr_para_main_begin_int:
342 {
343   \int_gincr:N \g__tag_para_main_begin_int
344 }
345 \cs_new_protected:Npn \__tag_gincr_para_begin_int:
346 {
347   \int_gincr:N \g__tag_para_begin_int
348 }
349 \cs_new_protected:Npn \__tag_gincr_para_main_end_int:
350 {
351   \int_gincr:N \g__tag_para_main_end_int
352 }
353 \cs_new_protected:Npn \__tag_gincr_para_end_int:
354 {
355   \int_gincr:N \g__tag_para_end_int
356 }

```

(End of definition for __tag_gincr_para_main_begin_int: and others.)

__tag_start_para_ints:

```

\__tag_stop_para_ints:
357 \cs_new_protected:Npn \__tag_start_para_ints:
358 {
359   \cs_set_protected:Npn \__tag_gincr_para_main_begin_int:
360   {
361     \int_gincr:N \g__tag_para_main_begin_int
362   }
363   \cs_set_protected:Npn \__tag_gincr_para_begin_int:
364   {
365     \int_gincr:N \g__tag_para_begin_int
366   }
367   \cs_set_protected:Npn \__tag_gincr_para_main_end_int:
368   {
369     \int_gincr:N \g__tag_para_main_end_int

```

```

370 }
371 \cs_set_protected:Npn \__tag_gincr_para_end_int:
372 {
373   \int_gincr:N \g__tag_para_end_int
374 }
375 }
376 \cs_new_protected:Npn \__tag_stop_para_ints:
377 {
378   \cs_set_eq:NN \__tag_gincr_para_main_begin_int:\prg_do_nothing:
379   \cs_set_eq:NN \__tag_gincr_para_begin_int:\prg_do_nothing:
380   \cs_set_eq:NN \__tag_gincr_para_main_end_int:\prg_do_nothing:
381   \cs_set_eq:NN \__tag_gincr_para_end_int:\prg_do_nothing:
382 }

```

(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:

```

383 \cs_new:Npn \__tag_para_main_store_struct:
384 {
385   \tl_gset:Ne \g__tag_para_main_struct_tl {\int_use:N \c@g__tag_struct_abs_int }
386 }

```

(End of definition for __tag_para_main_store_struct:.)

temporary adaption for the block module:

```

387 \AddToHook{package/latex-lab-testphase-block/after}
388 {
389   \tl_if_exist:NT \l_tag_para_attr_class_tl
390   {
391     \tl_set:Nn \l__tag_para_attr_class_tl { \l_tag_para_attr_class_tl }
392   }
393 }

```

para/tagging (setup-key) 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. **para/tagging (tool-key)** 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`

```

unittag (deprecated)
para-flattened (deprecated) 394 \keys_define:nm { __tag / setup }
395 {
396   para/tagging      .bool_set:N = \l__tag_para_bool,
397   debug/show/para   .code:n = {\bool_set_true:N \l__tag_para_show_bool},
398   debug/show/paraOff .code:n = {\bool_set_false:N \l__tag_para_show_bool},
399   para/tag          .tl_set:N   = \l__tag_para_tag_tl,
400   para/maintag      .tl_set:N   = \l__tag_para_main_tag_tl,
401   para/flattened     .bool_set:N = \l__tag_para_flattened_bool
402 }

```

```

403 \keys_define:nn { tag / tool}
404 {
405   para/tagging .bool_set:N = \l__tag_para_bool,
406   para/tag .tl_set:N = \l__tag_para_tag_tl,
407   para/maintag .tl_set:N = \l__tag_para_main_tag_tl,
408   para/flattened .bool_set:N = \l__tag_para_flattened_bool
409 }

```

the deprecated names

```

410 \keys_define:nn { __tag / setup }
411 {
412   paratagging .bool_set:N = \l__tag_para_bool,
413   paratagging-show .bool_set:N = \l__tag_para_show_bool,
414   paratag .tl_set:N = \l__tag_para_tag_tl
415 }
416 \keys_define:nn { tag / tool}
417 {
418   para .bool_set:N = \l__tag_para_bool,
419   paratag .tl_set:N = \l__tag_para_tag_tl,
420   unittag .tl_set:N = \l__tag_para_main_tag_tl,
421   para-flattened .bool_set:N = \l__tag_para_flattened_bool
422 }

```

(End of definition for para/tagging (setup-key) and others. These functions are documented on page 43.)

Helper command for debugging:

```

423 \cs_new_protected:Npn \__tag_check_para_begin_show:nn #1 #2
424   %#1 color, #2 prefix
425   {
426     \bool_if:NT \l__tag_para_show_bool
427     {
428       \tag_mc_begin:n{artifact}
429       \llap{\color_select:n{#1}\tiny#2\int_use:N\g__tag_para_begin_int\ }
430       \tag_mc_end:
431     }
432   }
433
434 \cs_new_protected:Npn \__tag_check_para_end_show:nn #1 #2
435   %#1 color, #2 prefix
436   {
437     \bool_if:NT \l__tag_para_show_bool
438     {
439       \tag_mc_begin:n{artifact}
440       \rlap{\color_select:n{#1}\tiny\ #2\int_use:N\g__tag_para_end_int}
441       \tag_mc_end:
442     }
443   }

```

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 should move into ltagging, so we add a test for the transition.


```

444 \str_if_exist:cF { l__socket_tagsupport/para/begin_plug_str }
445 {
446     \socket_new:nn      {tagsupport/para/begin}{0}
447     \socket_new:nn      {tagsupport/para/end}{0}
448
449     \socket_new_plug:nnn{tagsupport/para/begin}{plain}
450     {
451         \bool_if:NT \l__tag_para_bool
452         {
453             \bool_if:NF \l__tag_para_flattened_bool
454             {
455                 \__tag_gincr_para_main_begin_int:
456                 \tag_struct_begin:n
457                 {
458                     tag=\l__tag_para_main_tag_tl,
459                 }
460                 \__tag_para_main_store_struct:
461             }
462             \__tag_gincr_para_begin_int:
463             \tag_struct_begin:n {tag=\l__tag_para_tag_tl}
464             \__tag_check_para_begin_show:nn {green}{ }
465             \tag_mc_begin:n { }
466         }
467     }
468     \socket_new_plug:nnn{tagsupport/para/begin}{block}
469     {
470         \bool_if:NT \l__tag_para_bool
471         {
472             \legacy_if:NF { @inlabel }
473             {
474                 \__tag_check_typeout_v:n
475                 {==>~ @endpe = \legacy_if:NTF { @endpe }{true}{false} \on@line }
476                 \legacy_if:NF { @endpe }
477                 {
478                     \bool_if:NF \l__tag_para_flattened_bool
479                     {
480                         \__tag_gincr_para_main_begin_int:
481                         \tag_struct_begin:n
482                         {
483                             tag=\l__tag_para_main_tag_tl,
484                             attribute-class=\l__tag_para_main_attr_class_tl,
485                         }
486                         \__tag_para_main_store_struct:
487                     }
488                 }
489                 \__tag_gincr_para_begin_int:
490                 \__tag_check_typeout_v:n {==>~increment~ P \on@line }
491                 \tag_struct_begin:n
492                 {
493                     tag=\l__tag_para_tag_tl
494                     ,attribute-class=\l__tag_para_attr_class_tl
495                 }
496                 \__tag_check_para_begin_show:nn {green}{\PARALABEL}
497                 \tag_mc_begin:n { }

```

```

498     }
499   }
500 }

```

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

```

501 \socket_new_plug:nnn{tagsupport/para/end}{plain}
502 {
503   \bool_if:NT \l__tag_para_bool
504   {
505     \__tag_gincr_para_end_int:
506     \__tag_check_typeout_v:n {==>~increment~ /P \on@line }
507     \tag_mc_end:
508     \__tag_check_para_end_show:nn {red}{}
509     \tag_struct_end:
510     \bool_if:NF \l__tag_para_flattened_bool
511     {
512       \__tag_gincr_para_main_end_int:
513       \tag_struct_end:
514     }
515   }
516 }
517 }

```

By default we assign the plain plug:

```

518 \socket_assign_plug:nn { tagsupport/para/begin}{plain}
519 \socket_assign_plug:nn { tagsupport/para/end}{plain}

```

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

```

520 \AddToHook{para/begin}{ \socket_use:n { tagsupport/para/begin }
521 }
522 \AddToHook{para/end} { \socket_use:n { tagsupport/para/end } }

```

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

```

523 \AddToHook{package/latex-lab-testphase-block/after}
524 {
525   \RemoveFromHook{para/begin}[tagpdf]
526   \RemoveFromHook{para/end}[latex-lab-testphase-block]
527   \AddToHook{para/begin}[tagpdf]
528   {
529     \socket_use:n { tagsupport/para/begin }
530   }
531   \AddToHook{para/end}[tagpdf]
532   {
533     \socket_use:n { tagsupport/para/end }
534   }
535   \socket_assign_plug:nn { tagsupport/para/begin}{block}
536 }
537

```

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.

```

538 \AddToHook{enddocument/info}
539 {
540   \tag_if_active:F
541   {
542     \msg_redirect_name:nnn { tag } { para-hook-count-wrong } { warning }
543   }
544   \int_compare:nNnF {\g__tag_para_main_begin_int}={\g__tag_para_main_end_int}
545   {
546     \msg_error:nneee
547     {tag}
548     {para-hook-count-wrong}
549     {\int_use:N\g__tag_para_main_begin_int}
550     {\int_use:N\g__tag_para_main_end_int}
551     {text-unit}
552   }
553   \int_compare:nNnF {\g__tag_para_begin_int}={\g__tag_para_end_int}
554   {
555     \msg_error:nneee
556     {tag}
557     {para-hook-count-wrong}
558     {\int_use:N\g__tag_para_begin_int}
559     {\int_use:N\g__tag_para_end_int}
560     {text}
561   }
562 }
</package>

```

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

```

563 <base>\newcommand\tagpdfparaOn {}
564 <base>\newcommand\tagpdfparaOff{}
565 <*package>
566 \renewcommand\tagpdfparaOn {\bool_set_true:N \l__tag_para_bool}
567 \renewcommand\tagpdfparaOff{\bool_set_false:N \l__tag_para_bool}

```

(End of definition for \tagpdfparaOn and \tagpdfparaOff. These functions are documented on page 43.)

\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}%

```

```

568 \NewDocumentCommand\tagpdfsuppressmarks{m}
569   {{\use:c{__tag_mc_disable_marks:} #1}}

```

(End of definition for \tagpdfsuppressmarks. This function is documented on page 43.)

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)

```

570 \keys_define:nn { __tag / setup }
571   {
572     text / lang .tl_set:N = \l__tag_struct_lang_tl
573   }

```

(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 be there at the end. TODO check if Pagination should be changeable.

```

574 \cs_new_protected:Npn\__tag_hook_kernel_before_head:{}
575 \cs_new_protected:Npn\__tag_hook_kernel_after_head:{}
576 \cs_new_protected:Npn\__tag_hook_kernel_before_foot:{}
577 \cs_new_protected:Npn\__tag_hook_kernel_after_foot:{}

```

This can go once the new OR is active (June 2025)

```

578 \AddToHook{begindocument}
579   {
580     \cs_if_exist:NT \@kernel@before@head
581       {
582         \tl_put_right:Nn \@kernel@before@head {\__tag_hook_kernel_before_head:}
583         \tl_put_left:Nn \@kernel@after@head {\__tag_hook_kernel_after_head:}
584         \tl_put_right:Nn \@kernel@before@foot {\__tag_hook_kernel_before_foot:}
585         \tl_put_left:Nn \@kernel@after@foot {\__tag_hook_kernel_after_foot:}
586       }
587   }

```

If the new page sockets exist, we use them.

```

588 \str_if_exist:cT { l__socket_tagsupport/build/page/footer_plug_str }
589   {
590     \NewSocketPlug{tagsupport/build/page/header}{tagpdf}
591     {
592       \__tag_hook_kernel_before_head:
593       #2
594       \__tag_hook_kernel_after_head:
595     }
596     \AssignSocketPlug{tagsupport/build/page/header}{tagpdf}
597   }

```

```

598     \NewSocketPlug{tagsupport/build/page/footer}{tagpdf}
599     {
600         \__tag_hook_kernel_before_foot:
601         #2
602         \__tag_hook_kernel_after_foot:
603     }
604     \AssignSocketPlug{tagsupport/build/page/footer}{tagpdf}
605 }
606
607 \bool_new:N \g__tag_saved_in_mc_bool
608 \cs_new_protected:Npn \__tag_exclude_headfoot_begin:
609 {
610     \bool_set_false:N \l__tag_para_bool
611     \bool_if:NTF \g__tag_mode_lua_bool
612     {
613         \tag_mc_end_push:
614     }
615     {
616         \bool_gset_eq:NN \g__tag_saved_in_mc_bool \g__tag_in_mc_bool
617         \bool_gset_false:N \g__tag_in_mc_bool
618     }
619     \tag_mc_begin:n {artifact}
620     \tag_suspend:n{headfoot}
621 }
622 \cs_new_protected:Npn \__tag_exclude_headfoot_end:
623 {
624     \tag_resume:n{headfoot}
625     \tag_mc_end:
626     \bool_if:NTF \g__tag_mode_lua_bool
627     {
628         \tag_mc_begin_pop:n{ }
629     }
630     {
631         \bool_gset_eq:NN \g__tag_in_mc_bool \g__tag_saved_in_mc_bool
632     }
633 }

```

This version allows to use an Artifact structure

```

634 \__tag_attr_new_entry:nn {_tag/attr/pagination}{/0/Artifact/Type/Pagination}
635 \cs_new_protected:Npn \__tag_exclude_struct_headfoot_begin:n #1
636 {
637     \bool_set_false:N \l__tag_para_bool
638     \bool_if:NTF \g__tag_mode_lua_bool
639     {
640         \tag_mc_end_push:
641     }
642     {
643         \bool_gset_eq:NN \g__tag_saved_in_mc_bool \g__tag_in_mc_bool
644         \bool_gset_false:N \g__tag_in_mc_bool
645     }
646     \tag_struct_begin:n{tag=Artifact,attribute-class=__tag/attr/#1}
647     \tag_mc_begin:n {artifact=#1}
648     \tag_suspend:n{headfoot}

```

```

649 }
650
651 \cs_new_protected:Npn \__tag_exclude_struct_headfoot_end:
652 {
653   \tag_resume:n{headfoot}
654   \tag_mc_end:
655   \tag_struct_end:
656   \bool_if:NTF \g__tag_mode_lua_bool
657   {
658     \tag_mc_begin_pop:n{ }
659   }
660   {
661     \bool_gset_eq:NN \g__tag_in_mc_bool \g__tag_saved_in_mc_bool
662   }
663 }

```

And now the keys

`page/exclude-header-footer (setup-key)`
`exclude-header-footer (deprecated)`

```

664 \keys_define:nn { __tag / setup }
665 {
666   page/exclude-header-footer .choice:,
667   page/exclude-header-footer / true .code:n =
668   {
669     \cs_set_eq:NN \__tag_hook_kernel_before_head: \__tag_exclude_headfoot_begin:
670     \cs_set_eq:NN \__tag_hook_kernel_before_foot: \__tag_exclude_headfoot_begin:
671     \cs_set_eq:NN \__tag_hook_kernel_after_head: \__tag_exclude_headfoot_end:
672     \cs_set_eq:NN \__tag_hook_kernel_after_foot: \__tag_exclude_headfoot_end:
673   },
674   page/exclude-header-footer / pagination .code:n =
675   {
676     \cs_set:Nn \__tag_hook_kernel_before_head: { \__tag_exclude_struct_headfoot_begin:n {page}
677     \cs_set:Nn \__tag_hook_kernel_before_foot: { \__tag_exclude_struct_headfoot_begin:n {page}
678     \cs_set_eq:NN \__tag_hook_kernel_after_head: \__tag_exclude_struct_headfoot_end:
679     \cs_set_eq:NN \__tag_hook_kernel_after_foot: \__tag_exclude_struct_headfoot_end:
680   },
681   page/exclude-header-footer / false .code:n =
682   {
683     \cs_set_eq:NN \__tag_hook_kernel_before_head: \prg_do_nothing:
684     \cs_set_eq:NN \__tag_hook_kernel_before_foot: \prg_do_nothing:
685     \cs_set_eq:NN \__tag_hook_kernel_after_head: \prg_do_nothing:
686     \cs_set_eq:NN \__tag_hook_kernel_after_foot: \prg_do_nothing:
687   },
688   page/exclude-header-footer .default:n = true,
689   page/exclude-header-footer .initial:n = true,

```

deprecated name

```

690   exclude-header-footer .meta:n = { page/exclude-header-footer = {#1} }
691 }

```

(End of definition for `page/exclude-header-footer (setup-key)` and `exclude-header-footer (deprecated)`.
 These functions are documented on page 43.)

A special, experimental tagged version, which only works with fancyhdr or similar that uses parbox

```

692 \AtBeginDocument
693 {
694   \socket_if_exist:nT{tagsupport/parbox/before}
695   {
696     \NewTaggingSocketPlug{parbox/before}{tag/footer}
697     {
698       \tag_struct_begin:n{tag=Span}
699       \tag_mc_begin:n{ }
700     }
701
702     \NewTaggingSocketPlug{parbox/after}{tag/footer}
703     {
704       \tag_mc_end:
705       \tag_struct_end:
706     }
707   }
708 }
709
710 \cs_new_protected:Npn \__tag_headfoot_tagged_begin:n #1
711 {
712   \AssignTaggingSocketPlug{parbox/before}{tag/footer}
713   \AssignTaggingSocketPlug{parbox/after}{tag/footer}
714   \bool_set_false:N \l__tag_para_bool
715   \bool_if:NTF \g__tag_mode_lua_bool
716   {
717     \tag_mc_end_push:
718   }
719   {
720     \bool_gset_eq:NN \g__tag_saved_in_mc_bool \g__tag_in_mc_bool
721     \bool_gset_false:N \g__tag_in_mc_bool
722   }
723   \tag_struct_begin:n{tag=Artifact,attribute-class=__tag/attr/#1,parent=\tag_get:n{current_Sec
724 }
725
726 \cs_new_protected:Npn \__tag_headfoot_tagged_end:
727 {
728   \tag_struct_end:
729   \bool_if:NTF \g__tag_mode_lua_bool
730   {
731     \tag_mc_begin_pop:n{ }
732   }
733   {
734     \bool_gset_eq:NN \g__tag_in_mc_bool \g__tag_saved_in_mc_bool
735   }
736 }
737 \keys_define:nn { __tag / setup }
738 {
739   page/tag-header-footer .code:n =
740   {
741     \cs_set:Nn \__tag_hook_kernel_before_head: { \__tag_headfoot_tagged_begin:n {pagination}}
742     \cs_set:Nn \__tag_hook_kernel_before_foot: { \__tag_headfoot_tagged_begin:n {pagination}}
743     \cs_set_eq:NN \__tag_hook_kernel_after_head: \__tag_headfoot_tagged_end:

```

```

744     \cs_set_eq:NN \__tag_hook_kernel_after_foot: \__tag_headfoot_tagged_end:
745   }
746 }

```

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.

```

747 \bool_lazy_and:nnTF
748 { \sys_if_engine luatex_p: }
749 {
750   \tl_if_empty_p:e
751   {
752     \lua_now:e
753     { if~ luatexbase.in_callback('pre_shipout_filter','linksplit')~
754       then~else~tex.print('1')~end
755     }
756   }
757 }
758 {
759   \hook_gput_code:nnn
760     {pdfannot/link/URI/before}
761     {tagpdf}
762     {
763       \tag_mc_end_push:
764       \tag_struct_begin:n { tag=Link }
765       \tag_mc_begin:n { tag=Link }
766     }
767
768   \hook_gput_code:nnn
769     {pdfannot/link/URI/after}
770     {tagpdf}
771     {
772       \tag_mc_end:
773       \tag_struct_end:
774       \tag_mc_begin_pop:n{ }
775     }
776
777   \hook_gput_code:nnn
778     {pdfannot/link/GoTo/before}
779     {tagpdf}
780     {
781       \tag_mc_end_push:
782       \tag_struct_begin:n{tag=Link}
783       \tag_mc_begin:n{tag=Link}
784     }
785
786   \hook_gput_code:nnn
787     {pdfannot/link/GoTo/after}

```



```

788     {tagpdf}
789     {
790         \tag_mc_end:
791         \tag_struct_end:
792         \tag_mc_begin_pop:n{ }
793     }
794
795     \hook_gput_code:nnn
796     {pdfannot/link/GoToR/before}
797     {tagpdf}
798     {
799         \tag_mc_end_push:
800         \tag_struct_begin:n{tag=Link}
801         \tag_mc_begin:n{tag=Link}
802     }
803
804     \hook_gput_code:nnn
805     {pdfannot/link/GoToR/after}
806     {tagpdf}
807     {
808         \tag_mc_end:
809         \tag_struct_end:
810         \tag_mc_begin_pop:n{ }
811     }
812     \hook_gput_code:nnn
813     {pdfannot/link/Launch/before}
814     {tagpdf}
815     {
816         \tag_mc_end_push:
817         \tag_struct_begin:n{tag=Link}
818         \tag_mc_begin:n{tag=Link}
819     }
820
821     \hook_gput_code:nnn
822     {pdfannot/link/Launch/after}
823     {tagpdf}
824     {
825         \tag_mc_end:
826         \tag_struct_end:
827         \tag_mc_begin_pop:n{ }
828     }
829     \hook_gput_code:nnn
830     {pdfannot/link/Named/before}
831     {tagpdf}
832     {
833         \tag_mc_end_push:
834         \tag_struct_begin:n{tag=Link}
835         \tag_mc_begin:n{tag=Link}
836     }
837
838     \hook_gput_code:nnn
839     {pdfannot/link/Named/after}
840     {tagpdf}
841     {

```

```

842     \tag_mc_end:
843     \tag_struct_end:
844     \tag_mc_begin_pop:n{ }
845   }
846 }
847 {
848   \hook_gput_code:nnn
849   {pdfannot/link/URI/before}
850   {tagpdf}
851   {
852     \tag_mc_end_push:
853     \tag_struct_begin:n { tag=Link }
854     \tag_mc_begin:n { tag=Link }
855     \pdfannot_dict_put:nne
856     { link/URI }
857     { StructParent }
858     { \tag_struct_parent_int: }
859   }
860
861   \hook_gput_code:nnn
862   {pdfannot/link/URI/after}
863   {tagpdf}
864   {
865     \tag_struct_insert_annot:ee {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
866     \tag_mc_end:
867     \tag_struct_end:
868     \tag_mc_begin_pop:n{ }
869   }
870
871   \hook_gput_code:nnn
872   {pdfannot/link/GoTo/before}
873   {tagpdf}
874   {
875     \tag_mc_end_push:
876     \tag_struct_begin:n{tag=Link}
877     \tag_mc_begin:n{tag=Link}
878     \pdfannot_dict_put:nne
879     { link/GoTo }
880     { StructParent }
881     { \tag_struct_parent_int: }
882   }
883
884   \hook_gput_code:nnn
885   {pdfannot/link/GoTo/after}
886   {tagpdf}
887   {
888     \tag_struct_insert_annot:ee {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
889     \tag_mc_end:
890     \tag_struct_end:
891     \tag_mc_begin_pop:n{ }
892   }
893
894   \hook_gput_code:nnn
895   {pdfannot/link/GoToR/before}

```

```

896     {tagpdf}
897     {
898         \tag_mc_end_push:
899         \tag_struct_begin:n{tag=Link}
900         \tag_mc_begin:n{tag=Link}
901         \pdfannot_dict_put:nne
902         { link/GoToR }
903         { StructParent }
904         { \tag_struct_parent_int: }
905     }
906
907     \hook_gput_code:nnn
908     {pdfannot/link/GoToR/after}
909     {tagpdf}
910     {
911         \tag_struct_insert_annot:ee {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
912         \tag_mc_end:
913         \tag_struct_end:
914         \tag_mc_begin_pop:n{}
915     }
916
917     \hook_gput_code:nnn
918     {pdfannot/link/Named/before}
919     {tagpdf}
920     {
921         \tag_mc_end_push:
922         \tag_struct_begin:n{tag=Link}
923         \tag_mc_begin:n{tag=Link}
924         \pdfannot_dict_put:nne
925         { link/Named }
926         { StructParent }
927         { \tag_struct_parent_int: }
928     }
929
930     \hook_gput_code:nnn
931     {pdfannot/link/Named/after}
932     {tagpdf}
933     {
934         \tag_struct_insert_annot:ee {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
935         \tag_mc_end:
936         \tag_struct_end:
937         \tag_mc_begin_pop:n{}
938     }
939     \hook_gput_code:nnn
940     {pdfannot/link/Launch/before}
941     {tagpdf}
942     {
943         \tag_mc_end_push:
944         \tag_struct_begin:n{tag=Link}
945         \tag_mc_begin:n{tag=Link}
946         \pdfannot_dict_put:nne
947         { link/Launch }
948         { StructParent }
949         { \tag_struct_parent_int: }

```

```

950     }
951
952     \hook_gput_code:nnn
953     {pdfannot/link/Launch/after}
954     {tagpdf}
955     {
956         \tag_struct_insert_annot:ee {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
957         \tag_mc_end:
958         \tag_struct_end:
959         \tag_mc_begin_pop:n{ }
960     }
961 }

```

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 Catalog¹.

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.

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.

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

```

962 \prop_new:N \g__tag_css_prop
963 \prop_gset_from_keyval:Nn \g__tag_css_prop
964 {
965     latex-list-css.html=true,
966     latex-align-css.html=true
967 }
968
969
970 \bool_new:N \g__tag_css_bool
971 \bool_gset_true:N \g__tag_css_bool

```

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

```

972 \tl_gput_left:Nn \g__kernel_pdfmanagement_end_run_code_tl
973 {
974     \bool_lazy_and:nnT { \g__tag_css_bool }{ \tag_if_active_p: }
975     {
976         \prop_map_inline:Nn \g__tag_css_prop
977         {
978             \keys_set:nn { __tag / setup }{ catalog-supplemental-file= {#1} }
979         }
980     }
981 }
982
983 \keys_define:nn { __tag / setup }
984 {
985     attach-css .bool_gset:N = \g__tag_css_bool,
986     css-list .code:n =
987     {
988         \tl_if_empty:nTF{#1}
989         {\prop_gclear:N \g__tag_css_prop }
990         {\prop_gput:Nnn \g__tag_css_prop { #1 }{true}}
991     },
992     css-list-add .code:n = { \prop_gput:Nnn \g__tag_css_prop { #1 }{true} },
993     css-list-remove .code:n = { \prop_gremove:Nn \g__tag_css_prop { #1 } },
994 }

```

</package> The tagpdf-tree module
Commands trees and main dictionaries
Part of the tagpdf package
Ulrike Fischer
Version 0.99u, released 2025-07-16

Part IV

```
1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-tree-code} {2025-07-16} {0.99u}
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_tl
```

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:n{##1}~
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:Nx \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 clear i
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}{tagabspace}{-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:Nx\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.

`__tag/tree/namespaces`

```

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

```

(End of definition for `__tag/tree/namespaces:`)

`__tag_tree_write_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 0.99u, released 2025-07-16

Part V

1 Public Commands

<code>\tag_mc_begin:n</code>	<code>\tag_mc_begin:n {<key-values>}</code>
<code>\tag_mc_end:</code>	<code>\tag_mc_end:</code>

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.

<code>\tag_mc_use:n</code>	<code>\tag_mc_use:n {<label>}</code>
----------------------------	--

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.

<code>\tag_mc_artifact_group_begin:n</code>	<code>\tag_mc_artifact_group_begin:n {<name>}</code>
<code>\tag_mc_artifact_group_end:</code>	<code>\tag_mc_artifact_group_end:</code>

New: 2019-11-20

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

<code>\tag_mc_end_push:</code>	<code>\tag_mc_end_push:</code>
<code>\tag_mc_begin_pop:n</code>	<code>\tag_mc_begin_pop:n {<key-values>}</code>

New: 2021-04-22

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

<code>\tag_mc_if_in_p: *</code>	<code>\tag_mc_if_in:TF {<true code>} {<false code>}</code>
<code>\tag_mc_if_in:TF *</code>	Determines if a mc-chunk is open.

<code>\tag_mc_reset_box:N *</code>	<code>\tag_mc_reset_box:N <box></code>
------------------------------------	--

New: 2023-06-11

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

<code>\tag_mc_add_missing_to_stream:Nn</code>	<code>\tag_mc_add_missing_to_stream:Nn <box> {<stream name>}</code>
---	---

New: 2024-11-18

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.

<code>\tag_mc_new_stream:n</code>	<code>\tag_mc_new_stream:n {<stream name>}</code>
-----------------------------------	---

New: 2024-11-18

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`,

<code>tag (mc-key)</code>

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).

<code>artifact (mc-key)</code>

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).

<code>raw (mc-key)</code>

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.

<code>alt (mc-key)</code>

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.

<code>actualtext (mc-key)</code>

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.

<code>label (mc-key)</code>

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} {2025-07-16} {0.99u}
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` This booleans store the stash and artifact status of the mc-chunk.

`\l__tag_mc_artifact_bool`

20 \bool_new:N \l__tag_mc_key_stash_bool

21 \bool_new:N \l__tag_mc_artifact_bool

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

`\l__tag_mc_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` Variables used by the keys. `\l__@@mc_key_properties_tl` will collect a number of values. TODO: should this be a pdfdict now?

`\g__tag_mc_key_tag_tl`

`\l__tag_mc_key_label_tl`

`\l__tag_mc_key_properties_tl`²³

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 }
```

```

56     }
57   }
58 }
59 </shared>

```

(End of definition for \tag_mc_use:n. This function is documented on page 82.)

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

```

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 82.)

\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 82.)

\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 82.)

__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:NNTF \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}
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 support for the subtypes Header and Footer. Watermark,PageNum, LineNum,Redaction,Bates will be added if some use case emerges. If some use case for /BBox and /Attached emerges, it will be perhaps necessary to adapt the code.

```

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 84.)

220 </shared>

The tagpdf-mc-generic module
 Code related to Marked Content (mc-chunks), generic mode
 Part of the tagpdf package
 Ulrike Fischer
 Version 0.99u, released 2025-07-16

Part VI

1 Marked content code – generic mode

```
1 <@@=tag>
2 <*generic>
3 \ProvidesExplPackage {tagpdf-mc-code-generic} {2025-07-16} {0.99u}
4 {part of tagpdf - code related to marking chunks - generic mode}
5 </generic>
6 <*debug>
7 \ProvidesExplPackage {tagpdf-debug-generic} {2025-07-16} {0.99u}
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 tagabspace attribute retrieved from a label.

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

(End of definition for \l__tag_mc_ref_abspage_tl.)

`\l__tag_mc_tmpa_tl` temporary variable

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

(End of definition for \l__tag_mc_tmpa_tl.)

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

```
13 \newmarks \g__tag_mc_marks
```

(End of definition for \g__tag_mc_marks.)

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

```
14 \seq_new:N \g__tag_mc_main_marks_seq
```

```
15 \seq_new:N \g__tag_mc_footnote_marks_seq
```

```
16 \seq_new:N \g__tag_mc_multicol_marks_seq
```

(End of definition for \g__tag_mc_main_marks_seq, \g__tag_mc_footnote_marks_seq, and \g__tag_mc_multicol_marks_seq.)

`\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 83.)

`\l__tag_mc_firstmarks_seq` The marks content contains a number of data which we will have to access and compare,
`\l__tag_mc_botmarks_seq` 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

`__tag_mc_begin_marks:nn` Generic mode need to set marks for the page break and split stream handling. We always
`__tag_mc_artifact_begin_marks:n` set two marks to be able to detect the case when no mark is on a page/galley. MC-begin
`__tag_mc_end_marks:` 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 $\langle mc-num \rangle$ into the structure struct-num after the $\langle mc-prev \rangle$. 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_dict:n }
99   }
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 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:Nc \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:Nc
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     }
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 \OM
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` This is the main command to add mc to the stream. It is therefore guarded by the `__tag_add_missing_mcs_to_stream:Nn` 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 83.)

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

`\tag_mc_if_in_p:` 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 82.)

`__tag_mc_bmc:n` These are the low-level commands. There are now equal to the pdfmanagement commands generic mode, but we use an indirection in case luamode need something else.

`__tag_mc_emc:` 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.

`__tag_mc_bdc:nn`

```

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
\__tag_mc_handle_mcid:nn
\__tag_mc_handle_mcid:oo

```

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

```

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

```

295 <base>\cs_new_protected:Npn \tag_mc_begin:n #1 { \__tag_whatsits: \int_gincr:N \c@g__tag_MCID_abs_int
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
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 82.)

1.4 Keys

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

```

tag (mc-key)
raw (mc-key)
alt (mc-key) 384 <*generic>
actualtext (mc-key) 385 \keys_define:nn { __tag / mc }
label (mc-key) 386 {
artifact (mc-key) 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 83.)

The tagpdf-mc-luacode module

Code related to Marked Content (mc-chunks), luamode-specific

Part of the tagpdf package

Ulrike Fischer

Version 0.99u, released 2025-07-16

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 ... \tagmcbend` 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} {2025-07-16} {0.99u}
4 {tagpdf - mc code only for the luamode }
5 </luamode>
6 <*debug>
7 \ProvidesExplPackage {tagpdf-debug-lua} {2025-07-16} {0.99u}
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", "after"
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 83.)

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

__tag_mc_if_in:TF

```

\tag_mc_if_in_p:68 \prg_new_conditional:Nnn \__tag_mc_if_in: {p,T,F,TF}
\tag_mc_if_in:TF69 {

```

```

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__tag_
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 82.)

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

__tag_mc_lua_set_mc_type_attr:o

```

\__tag_mc_lua_unset_mc_type_attr: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:Nel__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 These commands will in the finish code replace the dummy for a mc by the real mcid
 __tag_mc_insert_mcid_single_kids:n 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 This is the lua variant for the command to put an mcid absolute number in the current
 __tag_mc_handle_stash:e 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> {
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
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 82.)

\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 82.)

\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 82.)

__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

```

tag (mc-key)  TODO: check conversion, check if local/global setting is right.
raw (mc-key)
alt (mc-key)  \keys_define:nn { __tag / mc }
lang (mc-key) {
actualtext (mc-key) tag .code:n = %
label (mc-key)  {
artifact (mc-key) \tl_set:Ne \l__tag_mc_key_tag_tl { #1 }
                  \tl_gset:Ne \g__tag_mc_key_tag_tl { #1 }
                  \lua_now:e
                  {
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         }
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 83.)

The tagpdf-struct module
 Commands to create the structure
 Part of the tagpdf package
 Ulrike Fischer
 Version 0.99u, released 2025-07-16

Part VIII

1 Public Commands

<code>\tag_struct_begin:n</code>	<code>\tag_struct_begin:n {<key-values>}</code>
<code>\tag_struct_end:</code>	<code>\tag_struct_end:</code>
<code>\tag_struct_end:n</code>	<code>\tag_struct_end:n {<tag>}</code>

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!

<code>\tag_struct_use:n</code>	<code>\tag_struct_use:n {<label>}</code>
<code>\tag_struct_use_num:n</code>	<code>\tag_struct_use_num:n {<structure number>}</code>

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.

<code>\tag_struct_object_ref:n</code>	<code>\tag_struct_object_ref:n {<structure number>}</code>
<code>\tag_struct_object_ref:e</code>	

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{<structnum>}`. 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.

<code>\tag_struct_insert_annot:nn</code>	<code>\tag_struct_insert_annot:nn {<object reference>} {<struct parent number>}</code>
--	--

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:.`

<code>\tag_struct_parent_int:</code>	<code>\tag_struct_parent_int:</code>
--------------------------------------	--------------------------------------

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

<code>\tag_struct_gput:nnn</code>	<code>\tag_struct_gput:nnn {<structure number>} {<keyword>} {<value>}</code>
-----------------------------------	--

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*) This keys allows to set the dictionary entry `/Title` in the structure object. The value

title-o (*struct key*) is handled as verbatim string and hex encoded. Commands are not expanded. **title-o** will expand the value once.

alt (*struct key*) This key inserts an `/Alt` value in the dictionary of structure object. The value is handled as verbatim string and hex encoded. The value will be expanded first once. If it is empty, nothing will happen.

actualtext (*struct key*) This key inserts an `/ActualText` value in the dictionary of structure object. The value is handled as verbatim string and hex encoded. The value will be expanded first once. If it is empty, nothing will happen.

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

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

E (*struct key*) This key sets the `/E` key, the expanded form of an abbreviation or an acronym (I couldn't think of a better name, so I stucked 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 `l3pdf` 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`.

2.2 Setup keys

`role/new-attribute` (setup-key) `role/new-attribute = {\langle name \rangle}{\langle Content \rangle}`
`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 = \langle object name \rangle`

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} {2025-07-16} {0.99u}
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.

14 `\seq_new:N \g__tag_struct_tag_stack_seq`

15 `\seq_gpush:Nn \g__tag_struct_tag_stack_seq {{Root}}{StructTreeRoot}}`

(End of definition for `\g__tag_struct_tag_stack_seq`.)

`\g__tag_struct_stack_current_tl` 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.

`\l__tag_struct_stack_parent_tmpa_tl`

16 `\</package>`

17 `\<base>\tl_new:N \g__tag_struct_stack_current_tl`

18 `\<base>\tl_gset:Nn \g__tag_struct_stack_current_tl {\int_use:N\c@g__tag_struct_abs_int}`

19 `\<*package>`

20 `\tl_new:N \l__tag_struct_stack_parent_tmpa_tl`

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

In luatex we will store the structure number as attribute.

21 `\sys_if_engine_luatex:TF`

22 `{`

23 `\cs_new:Npn __tag_struct_set_attribute:`

24 `{`

25 `\lua_now:e`

26 `{`

27 `tex.setattribute`

28 `(`

29 `"global",`

30 `luatexbase.attributes.g__tag_structnum_attr,`

31 `\g__tag_struct_stack_current_tl`

32 `)`

33 `}`

```

34     }
35   }
36   {
37     \cs_new_eq:NN \__tag_struct_set_attribute: \prg_do_nothing:
38   }

```

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, language, 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.

```

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

```

```

69     E,                % abbreviation
70     ActualText,
71     AF,                %pdf 2.0, array of dict, associated files
72     NS,                %pdf 2.0, dict, namespace
73     PhoneticAlphabet, %pdf 2.0
74     Phoneme            %pdf 2.0
75 }

(End of definition for \c__tag_struct_StructTreeRoot_entries_seq and \c__tag_struct_StructElem-
entries_seq.)

```

3.1 Variables used by the keys

<pre> \g__tag_struct_tag_t1 \g__tag_struct_tag_NS_t1 \l__tag_struct_roletag_t1 \g__tag_struct_roletag_NS_t1 \l__tag_struct_parenttag_t1 \l__tag_struct_parenttag_NS_t1 </pre>	<p>Use by the tag key to store the tag and the namespace. The <code>roletag</code> variables will hold locally rolemapping info needed for the parent-child checks. The <code>parenttag</code> variables allow to set the target role of the parent of stashed structures.</p> <pre> 76 \tl_new:N \g__tag_struct_tag_t1 77 \tl_new:N \g__tag_struct_tag_NS_t1 78 \tl_new:N \l__tag_struct_roletag_t1 79 \tl_new:N \l__tag_struct_roletag_NS_t1 80 \tl_new:N \l__tag_struct_parenttag_t1 81 \tl_set:Nn \l__tag_struct_parenttag_t1 {STASHED} 82 \tl_new:N \l__tag_struct_parenttag_NS_t1 83 \tl_set:Nn \l__tag_struct_parenttag_NS_t1 {latex} </pre> <p>(End of definition for <code>\g__tag_struct_tag_t1</code> and others.)</p>
<pre> \g__tag_struct_label_num_prop </pre>	<p>This will hold for every structure label the associated structure number. The prop will allow to fill the <code>/Ref</code> key directly at the first compilation if the <code>ref</code> key is used.</p> <pre> 84 \prop_new_linked:N \g__tag_struct_label_num_prop </pre> <p>(End of definition for <code>\g__tag_struct_label_num_prop</code>.)</p>
<pre> \l__tag_struct_elem_stash_bool </pre>	<p>This will keep track of the stash status</p> <pre> 85 \bool_new:N \l__tag_struct_elem_stash_bool </pre> <p>(End of definition for <code>\l__tag_struct_elem_stash_bool</code>.)</p>
<pre> \l__tag_struct_addkid_t1 </pre>	<p>This decides if a structure kid is added at the left or right of the parent. The default is right.</p> <pre> 86 \tl_new:N \l__tag_struct_addkid_t1 87 \tl_set:Nn \l__tag_struct_addkid_t1 {right} </pre> <p>(End of definition for <code>\l__tag_struct_addkid_t1</code>.)</p>

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.

```
88 \endpackage
89 \base\prop_new_linked:N \g__tag_struct_dest_num_prop
90 \endpackage
```

(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.

```
91 \prop_new_linked:N \g__tag_struct_ref_by_dest_prop
```

(End of definition for `\g__tag_struct_ref_by_dest_prop`.)

4 Commands

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

```
\__tag_struct_output_prop_aux:nn
\__tag_new_output_prop_handler:n
92 \cs_new:Npn \__tag_struct_output_prop_aux:nn #1 #2 %#1 num, #2 key
93 {
94   \prop_if_in:cnT
95     { \g__tag_struct_#1_prop }
96     { #2 }
97   {
98     \c_space_tl/#2~ \prop_item:cn{ \g__tag_struct_#1_prop } { #2 }
99   }
100 }
101
102 \cs_new_protected:Npn \__tag_new_output_prop_handler:n #1
103 {
104   \cs_new:cn { \__tag_struct_output_prop_#1:n }
105   {
106     \__tag_struct_output_prop_aux:nn {#1}{##1}
107   }
108 }
109 \endpackage
```

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

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


```

110 <*package | debug>
111 <package>\cs_new_protected:Npn \__tag_struct_prop_gput:nnn #1 #2 #3
112 <debug>\cs_set_protected:Npn \__tag_struct_prop_gput:nnn #1 #2 #3
113 {
114   \__tag_struct_prop_gput:cnn
115   { g__tag_struct_#1_prop }{#2}{#3}
116 <debug>\prop_gput:cnn { g__tag_struct_debug_#1_prop } {#2} {#3}
117 }
118 \cs_generate_variant:Nn \__tag_struct_prop_gput:nnn {onn,nne,nee,nnn}
119 </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.

```

120 <*package>
121 \tl_gset:Nn \g__tag_struct_stack_current_tl {1}

\__tag_pdf_name_e:n

122 \cs_new:Npn \__tag_pdf_name_e:n #1{\pdf_name_from_unicode_e:n{#1}}
123 </package>

```

(End of definition for __tag_pdf_name_e:n.)

```

g__tag_struct_1_prop
g__tag_struct_kids_1_seq

124 <*package>
125 \__tag_struct_prop_new:c { g__tag_struct_1_prop }
126 \__tag_struct_new_output_prop_handler:n {1}
127 \__tag_struct_seq_new:c { g__tag_struct_kids_1_seq }
128
129 \__tag_struct_prop_gput:nne
130 { 1 }
131 { Type }
132 { \pdf_name_from_unicode_e:n {StructTreeRoot} }
133
134 \__tag_struct_prop_gput:nne
135 { 1 }
136 { S }
137 { \pdf_name_from_unicode_e:n {StructTreeRoot} }
138
139 \__tag_struct_prop_gput:nne
140 { 1 }
141 { tag }
142 { {StructTreeRoot}{pdf} }
143
144 \__tag_struct_prop_gput:nne

```

```

145 { 1 }
146 { rolemap }
147 { {StructTreeRoot}{pdf} }
148
149 \__tag_struct_prop_gput:nne
150 { 1 }
151 { parentrole }
152 { {StructTreeRoot}{pdf} }
153

```

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

```

154 \pdf_version_compare:NnF < {2.0}
155 {
156   \__tag_struct_prop_gput:nne
157   { 1 }
158   { Namespaces }
159   { \pdf_object_ref:n { __tag/tree/namespaces } }
160 }
161 </package>

```

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

```

162 <debug>\prop_new:c { g__tag_struct_debug_1_prop }
163 <debug>\seq_new:c { g__tag_struct_debug_kids_1_seq }
164 <debug>\prop_gset_eq:cc { g__tag_struct_debug_1_prop }{ g__tag_struct_1_prop }
165 <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

```

166 <*package>
167 \cs_new:Npn \__tag_struct_get_id:n #1 %#1=struct num
168 {
169   (
170     ID.
171     \prg_replicate:nn
172     { \int_abs:n{\g__tag_tree_id_pad_int - \tl_count:e { \int_to_arabic:n { #1 } }} }
173     { 0 }
174     \int_to_arabic:n { #1 }
175   )
176 }

```

(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.

```

177 \\pdf_version_compare:NnTF < {2.0}
178 {
179   \\cs_new_protected:Npn \\_tag_struct_set_tag_info:nnn #1 #2 #3
180     {%#1 structure number, #2 tag, #3 NS
181     {
182       \\_tag_struct_prop_gput:nne
183         { #1 }
184         { S }
185         { \\pdf_name_from_unicode_e:n {#2} } %
186       \\_tag_struct_prop_gput:nnn
187         { #1 }
188         { tag }
189         { {#2} {} }
190     }
191   }
192 {
193   \\cs_new_protected:Npn \\_tag_struct_set_tag_info:nnn #1 #2 #3
194     {
195       \\_tag_struct_prop_gput:nne
196         { #1 }
197         { S }
198         { \\pdf_name_from_unicode_e:n {#2} } %
199       \\prop_get:NnNT \\g__tag_role_NS_prop {#3} \\l__tag_get_tmpc_tl
200       {
201         \\_tag_struct_prop_gput:nne
202           { #1 }
203           { NS }
204           { \\l__tag_get_tmpc_tl } %
205       }
206       \\_tag_struct_prop_gput:nnn
207         { #1 }
208         { tag }
209         { {#2} {#3} }
210     }
211   }
212 \\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.

```

213 \\cs_new_protected:Npn \\_tag_struct_get_role:nnNN #1 #2 #3 #4
214   {%#1 :struct num,
215   %#2 :rolemap or parentrole
216   %#3 :tlvar for tag (rolemapped)
217   %#4 :tlvar for NS (rolemapped, so standard or empty or UNKNOWN)
218   {

```

```

219 \prop_get:cnNTF
220 { g__tag_struct_#1_prop }
221 { #2 }
222 \l__tag_get_tmpc_tl
223 {
224   \tl_set:Ne #3{\exp_last_unbraced:No\use_i:nn { \l__tag_get_tmpc_tl }}
225   \tl_set:Ne #4{\exp_last_unbraced:No\use_ii:nn { \l__tag_get_tmpc_tl }}
226 }
227 {
228   \tl_clear:N#3
229   \tl_clear:N#4
230 }
231 }
232 \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 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!

```

233 \cs_new:Npn \__tag_struct_mcid_dict:n #1 %#1 MCID absnum
234 {
235   <<
236   /Type \c_space_tl /MCR \c_space_tl
237   /Pg
238   \c_space_tl
239   \pdf_pageobject_ref:n { \property_ref:enn{mcid-#1}{tagabspage}{1} }
240   /MCID \c_space_tl \property_ref:enn{mcid-#1}{tagmcid}{1}
241   >>
242 }
243 </package>

244 <*package | debug>
245 <package>\cs_new_protected:Npn \__tag_struct_kid_mc_gput_right:nn #1 #2
246 <debug>\cs_set_protected:Npn \__tag_struct_kid_mc_gput_right:nn #1 #2
247 %#1 structure num, #2 MCID absnum%
248 {
249   \__tag_seq_gput_right:ce
250   { g__tag_struct_kids_#1_seq }
251   {
252     \__tag_struct_mcid_dict:n {#2}
253   }

```

```

254 <debug> \seq_gput_right:cn
255 <debug> { g__tag_struct_debug_kids_#1_seq }
256 <debug> {
257 <debug> MC~#2
258 <debug> }
259 \__tag_seq_gput_right:cn
260 { g__tag_struct_kids_#1_seq }
261 {
262 \prop_item:Nn \g__tag_struct_cont_mc_prop {#2}
263 }
264 }
265 <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 This commands adds a structure as kid. We only need to record the object reference in the sequence.

__tag_struct_kid_struct_gput_right:ee

```

266 <package>\cs_new_protected:Npn\__tag_struct_kid_struct_gput_right:nn #1 #2
267 <debug>\cs_set_protected:Npn\__tag_struct_kid_struct_gput_right:nn #1 #2
268 %%#1 num of parent struct, #2 kid struct
269 {
270 \__tag_seq_gput_right:ce
271 { g__tag_struct_kids_#1_seq }
272 {
273 \pdf_object_ref_indexed:nn { __tag/struct }{ #2 }
274 }
275 <debug> \seq_gput_right: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_right:nn {ee}

(End of definition for \__tag_struct_kid_struct_gput_right:nn.)

```

__tag_struct_kid_struct_gput_left:nn 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.

__tag_struct_kid_struct_gput_left:ee

```

282 <package>\cs_new_protected:Npn\__tag_struct_kid_struct_gput_left:nn #1 #2
283 <debug>\cs_set_protected:Npn\__tag_struct_kid_struct_gput_left:nn #1 #2
284 %%#1 num of parent struct, #2 kid struct
285 {
286 \__tag_seq_gput_left:ce
287 { g__tag_struct_kids_#1_seq }
288 {
289 \pdf_object_ref_indexed:nn { __tag/struct }{ #2 }
290 }
291 <debug> \seq_gput_left:cn
292 <debug> { g__tag_struct_debug_kids_#1_seq }
293 <debug> {
294 <debug> Struct~#2
295 <debug> }
296 }
297 <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` 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

```

298 <package>\cs_new_protected:Npn\__tag_struct_kid_OBJR_gput_right:nnn #1 #2 #3
299 <package>
300 <package>
301 <debug>\cs_set_protected:Npn\__tag_struct_kid_OBJR_gput_right:nnn #1 #2 #3
302 %%#1 num of parent struct,#2 obj reference,#3 page object reference
303 {
304   \pdf_object_unnamed_write:nn
305   { dict }
306   {
307     /Type/OBJR/Obj~#2/Pg~#3
308   }
309   \__tag_seq_gput_right:ce
310   { g__tag_struct_kids_#1_seq }
311   {
312     \pdf_object_ref_last:
313   }
314 <debug>   \seq_gput_right:ce
315 <debug>   { g__tag_struct_debug_kids_#1_seq }
316 <debug>   {
317 <debug>     OBJR~reference
318 <debug>   }
319 }
320 </package|debug>
321 <*package>
322 \cs_generate_variant:Nn\__tag_struct_kid_OBJR_gput_right:nnn { eee }

```

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

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

```

323 \cs_new_protected:Npn\__tag_struct_exchange_kid_command:N #1 %#1 = seq var
324 {
325   \seq_gpop_left:NN #1 \l__tag_tmpa_tl
326   \tl_replace_once:Nnn \l__tag_tmpa_tl
327   {\__tag_mc_insert_mcid_kids:n}
328   {\__tag_mc_insert_mcid_single_kids:n}
329   \seq_gput_left:No #1 { \l__tag_tmpa_tl }
330 }
331
332 \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.

```

333 \cs_new_protected:Npn \__tag_struct_fill_kid_key:n #1 %#1 is the struct num
334 {
335   \bool_if:NF \g__tag_mode_lua_bool
336   {
337     \seq_clear:N \l__tag_tmpa_seq
338     \seq_map_inline:cn { g__tag_struct_kids_#1_seq }
339     { \seq_put_right:Ne \l__tag_tmpa_seq { ##1 } }
340     %\seq_show:c { g__tag_struct_kids_#1_seq }
341     %\seq_show:N \l__tag_tmpa_seq
342     \seq_remove_all:Nn \l__tag_tmpa_seq {}
343     %\seq_show:N \l__tag_tmpa_seq
344     \seq_gset_eq:cN { g__tag_struct_kids_#1_seq } \l__tag_tmpa_seq
345   }
346
347   \int_case:nnF
348   {
349     \seq_count:c
350     {
351       g__tag_struct_kids_#1_seq
352     }
353   }
354   {
355     { 0 }
356     { } %no kids, do nothing
357     { 1 } % 1 kid, insert
358     {
359       % in this case we need a special command in
360       % luamode to get the array right. See issue #13
361       \sys_if_engine luatex:TF
362       {
363         \__tag_struct_exchange_kid_command:c
364         {g__tag_struct_kids_#1_seq}

```

check if we get null

```

365       \tl_set:Nc\l__tag_tmpa_tl
366       { \use:e{ \seq_item:cn { g__tag_struct_kids_#1_seq } {1} } }
367       \tl_if_eq:NNF \l__tag_tmpa_tl \c__tag_struct_null_tl
368       {
369         \__tag_struct_prop_gput:nne
370         {#1}
371         {K}
372         {
373           \seq_item:cn
374           {
375             g__tag_struct_kids_#1_seq
376           }
377           {1}
378         }
379       }
380     }
381     {
382       \__tag_struct_prop_gput:nne
383       {#1}

```

```

384         {K}
385         {
386             \seq_item:cn
387             {
388                 g__tag_struct_kids_#1_seq
389             }
390             {1}
391         }
392     }
393 } %
394 }
395 { %many kids, use an array
396     \__tag_struct_prop_gput:nne
397     {#1}
398     {K}
399     {
400         [
401             \seq_use:cn
402             {
403                 g__tag_struct_kids_#1_seq
404             }
405             {
406                 \c_space_tl
407             }
408         ]
409     }
410 }
411 }
412

```

(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.

```

413 \cs_new_protected:Npn \__tag_struct_get_dict_content:nN #1 #2 %#1: structure num
414 {
415     \tl_clear:N #2
416     \prop_map_inline:cn { g__tag_struct_#1_prop }
417     {

```

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.

```

418     \cs_if_exist_use:cTF {__tag_struct_format_##1:nnN}
419     {
420         {##1}{##2}#2
421     }
422     {
423         \tl_put_right:Ne #2 { \c_space_tl/##1~##2 }
424     }
425 }
426 }

```


(End of definition for `__tag_struct_get_dict_content:nn`.)

This three entries should not end in the PDF. Todo: check if the S/NS keys can be dropped and replaced by a processing of the tag key.

```

427 \cs_new:Nn\__tag_struct_format_rolemap:nnN{
428 \cs_new:Nn\__tag_struct_format_parentrole:nnN{
429 \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.

```

430 \cs_new_protected:Nn\__tag_struct_format_parentnum:nnN
431 {
432   \tl_put_right:Ne #3 { ~/P~\pdf_object_ref_indexed:nn { __tag/struct} { #2 } }
433 }

```

(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.

```

434 \cs_new_protected:Nn\__tag_struct_format_Ref:nnN
435 {
436   \tl_put_right:Nn #3 { ~/#1~[ ] }
437   \clist_map_inline:nn{ #2 }
438   {
439     ##1 #3
440   }
441   \tl_put_right:Nn #3
442   { %[
443     \c_space_tl]
444   }
445 }

```

(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.

```

446 \cs_new_protected:Npn \__tag_struct_write_obj:n #1 % #1 is the struct num
447 {
448   \prop_if_exist:cTF { g__tag_struct_#1_prop }
449   {

```

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.

```

450   \prop_get:cnNF { g__tag_struct_#1_prop } {parentnum}\l__tag_tmpb_tl
451   {
452     % \prop_gput:cne { g__tag_struct_#1_prop } {P}
453     % {\pdf_object_ref_indexed:nn { __tag/struct }{1}}
454     \prop_gput:cne { g__tag_struct_#1_prop } {parentnum}{1}

```

```

455     \prop_gput:cne { g__tag_struct_#1_prop } {S}{/Artifact}
456     \seq_if_empty:cF {g__tag_struct_kids_#1_seq}
457     {
458         \msg_warning:nnee
459         {tag}
460         {struct-orphan}
461         { #1 }
462         {\seq_count:c{g__tag_struct_kids_#1_seq}}
463     }
464 }
465 \__tag_struct_fill_kid_key:n { #1 }
466 \__tag_struct_get_dict_content:nN { #1 } \l__tag_tmpa_tl
467 \pdf_object_write_indexed:nne
468 { __tag/struct }{ #1 }
469 {dict}
470 {
471     \l__tag_tmpa_tl\c_space_tl
472     /ID~\__tag_struct_get_id:n{#1}
473 }
474 }
475 {
476     \msg_error:nnn { tag } { struct-no-objnum } { #1}
477 }
478 }
479 }

```

(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

```

(1) \tag_struct_begin:n { tag=Link }
    \tag_mc_begin:n { tag=Link }
    \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:

```

```

480 \cs_new_protected:Npn \__tag_struct_insert_annot:nn #1 #2
481   {%#1 object reference to the annotation/xform

```

```

482 % #2 structparent number
483 {
484   \bool_if:NT \g__tag_active_struct_bool
485   {
486     %get the number of the parent structure:
487     \seq_get:NNF
488       \g__tag_struct_stack_seq
489       \l__tag_struct_stack_parent_tmpa_tl
490       {
491         \msg_error:nn { tag } { struct-faulty-nesting }
492       }
493     %put the obj number of the annot in the kid entry, this also creates
494     %the OBJR object
495     \__tag_property_record:nn {@tag@objr@page@#2 }{ tagabspage }
496     \__tag_struct_kid_OBJR_gput_right:eee
497     {
498       \l__tag_struct_stack_parent_tmpa_tl
499     }
500     {
501       #1 %
502     }
503     {
504       \pdf_pageobject_ref:n
505       { \property_ref:nnn {@tag@objr@page@#2 }{ tagabspage }{1} }
506     }
507     % add the parent obj number to the parent tree:
508     % the command always expands its arguments!
509     \__tag_parenttree_add_objr:nn
510     {
511       #2
512     }
513     {
514       \pdf_object_ref_indexed:nn
515       { __tag/struct }{ \l__tag_struct_stack_parent_tmpa_tl }
516     }
517     % increase the int:
518     \int_gincr:N \c@g__tag_parenttree_obj_int
519   }
520 }

```

(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.

```

521 \cs_new_protected:Npn \__tag_struct_insert_annot_shipout:nnn #1#2#3
522 % #1 structnum, #2 object reference, #3 StructParentNum
523 {
524   \__tag_struct_kid_OBJR_gput_right:eee
525   {

```

```

526     #1
527   }
528   {
529     #2
530   }
531   {
532     \pdf_pageobject_ref:n
533     { \int_use:N \g_shipout_readonly_int } %
534   }
535   % add the parent obj number to the parent tree:
536   % the command always expands its arguments!
537   \__tag_parenttree_add_objr:nn
538   {
539     #3
540   }
541   {
542     \pdf_object_ref_indexed:nn
543     { __tag/struct }{ #1 }
544   }
545 }

```

(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.

```

546 \cs_new:Npn \__tag_get_data_struct_tag:
547 {
548   \exp_args:Ne
549   \tl_tail:n
550   {
551     \prop_item:cn {g__tag_struct_\g__tag_struct_stack_current_tl _prop}{S}
552   }
553 }

```

(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.

```

554 \cs_new:Npn \__tag_get_data_struct_id:
555 {
556   \__tag_struct_get_id:n {\g__tag_struct_stack_current_tl}
557 }
558 \</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

```

559 \< *base>
560 \cs_new:Npn \__tag_get_data_struct_num:
561 {
562   \g__tag_struct_stack_current_tl
563 }
564 \</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.

```

565 ⟨*base⟩
566 \cs_new:Npn \__tag_get_data_struct_counter:
567 {
568   \int_use:N \c@g__tag_struct_abs_int
569 }
570 ⟨/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`

```

571 ⟨*package⟩
572 \cs_new_protected:Npn \__tag_struct_check_parent_child_aux:nnnnN #1#2#3#4#5
573 {
574   % #1 structure number of parent
575   % #2 key to use to retrieve role of parent (either rolemap or parentrole field)
576   % #3 structure number of parent
577   % #4 key to use to retrieve role of child (either rolemap or parentrole field)
578   % #5 t1 for return value

```

get parent rolemap

```

579   \__tag_struct_get_role:nnNN
580     {#1}
581     {#2}
582     \l__tag_get_parent_tmpa_tl
583     \l__tag_get_parent_tmpb_tl

```

get child rolemap

```

584   \__tag_struct_get_role:nnNN
585     {#3}
586     {#4}
587     \l__tag_get_child_tmpa_tl
588     \l__tag_get_child_tmpb_tl

```

check

```

589   \__tag_role_check_parent_child:ooooN
590     { \l__tag_get_parent_tmpa_tl } % rolemapped from above
591     { \l__tag_get_parent_tmpb_tl } % rolemapped from above
592     { \l__tag_get_child_tmpa_tl } %
593     { \l__tag_get_child_tmpb_tl } %
594   #5
595 }

```

(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.

```

596 \cs_new_protected:Npn \_tag_struct_check_parent_child:nn #1 #2
597 % #1 structure number of parent
598 % #2 structure number of child. %
599 % This assumes that the fields rolemap/parentrole has already been filled.
600 {

```

This records if logging is on

```

601 \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
602 {
603   \prop_get:cnN{g__tag_struct_#1_prop}{tag}\l__tag_get_parent_tmpa_tl
604   \prop_get:cnN{g__tag_struct_#2_prop}{tag}\l__tag_get_parent_tmpb_tl
605   \msg_note:nnee
606   { tag }
607   { role-parent-child-check }
608   {
609     \quark_if_no_value:NTF \l__tag_get_parent_tmpa_tl
610     {??}
611     {
612       \exp_last_unbraced:No\use_ii:nn
613       { \l__tag_get_parent_tmpa_tl }
614       :
615       \exp_last_unbraced:No\use_i:nn
616       { \l__tag_get_parent_tmpa_tl }
617     }
618   }
619   {
620     \quark_if_no_value:NTF \l__tag_get_parent_tmpb_tl
621     {??}
622     {
623       \exp_last_unbraced:No\use_ii:nn
624       { \l__tag_get_parent_tmpb_tl }
625       :
626       \exp_last_unbraced:No\use_i:nn
627       { \l__tag_get_parent_tmpb_tl }
628     }
629   }
630 }
631 \_tag_struct_check_parent_child_aux:nnnnN
632 {#1}
633 {rolemap}
634 {#2}
635 {rolemap}
636 \l__tag_parent_child_check_tl

```

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

```

637 \int_compare:nNnT {\l__tag_parent_child_check_tl} = { \c__tag_role_rule_checkparent_tl }
638 {
639   \_tag_struct_check_parent_child_aux:nnnnN

```

```

640         {#1}
641         {parentrole}
642         {#2}
643         {rolemap}
644         \l__tag_parent_child_check_tl
645     }
646     \__tag_check_struct_forbidden_parent_child:onn
647     {\l__tag_parent_child_check_tl}
648     {#1}
649     {#2}
650 }
651 \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.

```

652 \cs_new_protected:Npn \__tag_struct_use_check_parent_child:nn #1 #2
653 % #1 structure number of parent
654 % #2 structure number of child. %
655 {
656     \__tag_struct_get_role:enNN
657     {#2}
658     {rolemap}
659     \l__tag_get_child_tmpa_tl
660     \l__tag_get_child_tmpb_tl
661     \str_case:onTF { \l__tag_get_child_tmpa_tl }
662     {
663         {Part} {}
664         {Div} {}
665         {NonStruct} {}
666     }
667     { %child=Part etc
668         \__tag_struct_get_role:enNN
669         {#2}
670         {parentrole}
671         \l__tag_get_child_tmpa_tl
672         \l__tag_get_child_tmpb_tl
673         \str_if_eq:noTF
674         {STASHED}{\l__tag_get_child_tmpa_tl}
675         {
676             % warn about unknown relationship
677         }
678         {

```

```

679         % test if
680         \__tag_struct_get_role:enNN
681         {#1}
682         {parentrole}
683         \l__tag_get_parent_tmpa_tl
684         \l__tag_get_parent_tmpb_tl
685         \tl_if_eq:NNTF\l__tag_get_parent_tmpa_tl \l__tag_get_child_tmpa_tl
686         {
687             \__tag_struct_check_parent_child:nn {#1}{#2}
688         }
689         {
690             %warn that parent-tag was misused.
691         }
692     }
693 }
694 {
695     %child not Part etc, normal parent child test.
696     \__tag_struct_check_parent_child:nn {#1}{#2}
697 }
698 }
699 \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.

```

700 \socket_new:nn { tag/struct/tag }{1}
701 \socket_new_plug:nnn { tag/struct/tag }{ latex-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_check_structure_tag:N \g__tag_struct_tag_tl
715 }
716
717 \socket_new_plug:nnn { tag/struct/tag }{ pdf-tags }
718 {
719     \prop_get:NeNTF \g__tag_role_tags_NS_prop {#1} \l__tag_tmp_unused_tl
720     {
721         \seq_set_split:Nne \l__tag_tmpa_seq { / }

```



```

722     {#1/\l__tag_tmp_unused_tl}
723   }
724   {
725     \seq_set_split:Nne \l__tag_tmpa_seq { / }
726     {#1/}
727   }
728   \tl_gset:Ne \g__tag_struct_tag_tl { \seq_item:Nn\l__tag_tmpa_seq {1} }
729   \tl_gset:Ne \g__tag_struct_tag_NS_tl{ \seq_item:Nn\l__tag_tmpa_seq {2} }
730   \__tag_role_get:ooNN
731   { \g__tag_struct_tag_tl }
732   { \g__tag_struct_tag_NS_tl}
733   \l__tag_tmpa_tl
734   \l__tag_tmpp_tl
735   \tl_gset:Ne \g__tag_struct_tag_tl {\l__tag_tmpa_tl}
736   \tl_gset:Ne \g__tag_struct_tag_NS_tl{\l__tag_tmpp_tl}
737   \__tag_check_structure_tag:N \g__tag_struct_tag_tl
738 }
739 \socket_assign_plug:nn { tag/struct/tag } {latex-tags}

label (struct key)
stash (struct key)
parent (struct key) \keys_define:nn { __tag / struct }
firstkid (struct key) {
  tag (struct key) label .code:n =
  {
    title (struct key) {
743       \prop_gput:Nee\g__tag_struct_label_num_prop
744       {#1}{\int_use:N \c@g__tag_struct_abs_int}
745     }
746     alt (struct key) \__tag_property_record:eo
actualtext (struct key) {tagpdfstruct-#1}
747     lang (struct key) { \c__tag_property_struct_clist }
748     ref (struct key) },
749     E (struct key) stash .bool_set:N = \l__tag_struct_elem_stash_bool,
phoneme (struct key) parent .code:n =
750     {
751       \bool_lazy_and:nnTF
752       {
753         \prop_if_exist_p:c { g__tag_struct\_int_eval:n {#1}_prop }
754       }
755       {
756         \int_compare_p:nNn {#1}<{\c@g__tag_struct_abs_int}
757       }
758       { \tl_set:Ne \l__tag_struct_stack_parent_tmpa_tl { \int_eval:n {#1} } }
759     }
760     {
761       \msg_warning:nnee { tag } { struct-unknown }
762       { \int_eval:n {#1} }
763       { parent-key-ignored }
764     }
765   },
766   parent .default:n = {-1},
767   parent-tag .code:n =
768   {
769     \prop_get:NeNTF \g__tag_role_tags_NS_prop {#1} \l__tag_tmp_unused_tl
770     {
771       \seq_set_split:Nne \l__tag_tmpa_seq { / }
772     }

```

```

773         {#1/\l__tag_tmp_unused_tl}
774     }
775     {
776         \seq_set_split:Nne \l__tag_tmpa_seq { / }
777         {#1/}
778     }
779     \tl_set:Ne \l__tag_struct_parenttag_tl { \seq_item:Nn\l__tag_tmpa_seq {1} }
780     \tl_set:Ne \l__tag_struct_parenttag_NS_tl{ \seq_item:Nn\l__tag_tmpa_seq {2} }
781     \__tag_role_get:ooNN
782     { \l__tag_struct_parenttag_tl }
783     { \l__tag_struct_parenttag_NS_tl}
784     \l__tag_tmpa_tl
785     \l__tag_tmpb_tl
786     \tl_set:No \l__tag_struct_parenttag_tl { \l__tag_tmpa_tl}
787     \tl_set:No \l__tag_struct_parenttag_NS_tl{ \l__tag_tmpb_tl}
788     \__tag_check_structure_tag:N \l__tag_struct_parenttag_tl
789 },
790 firstkid .code:n = { \tl_set:Nn \l__tag_struct_addkid_tl {left} },
791 tag .code:n      = % S property
792 {
793     \socket_use:nn { tag/struct/tag }{#1}
794 },
795 title .code:n    = % T property
796 {
797     \str_set_convert:Nnnn
798     \l__tag_tmpa_str
799     { #1 }
800     { default }
801     { utf16/hex }
802     \__tag_struct_prop_gput:nne
803     { \int_use:N \c@g__tag_struct_abs_int }
804     { T }
805     { <\l__tag_tmpa_str> }
806 },
807 title-o .code:n  = % T property
808 {
809     \str_set_convert:Nonn
810     \l__tag_tmpa_str
811     { #1 }
812     { default }
813     { utf16/hex }
814     \__tag_struct_prop_gput:nne
815     { \int_use:N \c@g__tag_struct_abs_int }
816     { T }
817     { <\l__tag_tmpa_str> }
818 },
819 alt .code:n      = % Alt 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     { Alt }
831     { <\l__tag_tmpa_str> }
832 }
833 },
834 alttext .meta:n = {alt=#1},
835 actualtext .code:n = % ActualText property
836 {
837     \tl_if_empty:oF{#1}
838     {
839         \str_set_convert:Noon
840         \l__tag_tmpa_str
841         { #1 }
842         { default }
843         { utf16/hex }
844         \__tag_struct_prop_gput:nne
845         { \int_use:N \c@g__tag_struct_abs_int }
846         { ActualText }
847         { <\l__tag_tmpa_str>}
848     }
849 },
850 phoneme .code:n = % Phoneme property
851 {
852     \tl_if_empty:oF{#1}
853     {
854         \str_set_convert:Noon
855         \l__tag_tmpa_str
856         { #1 }
857         { default }
858         { utf16/hex }
859         \__tag_struct_prop_gput:nne
860         { \int_use:N \c@g__tag_struct_abs_int }
861         { Phoneme }
862         { <\l__tag_tmpa_str>}
863     }
864 },
865 lang .code:n = % Lang property
866 {
867     \__tag_struct_prop_gput:nne
868     { \int_use:N \c@g__tag_struct_abs_int }
869     { Lang }
870     { (#1) }
871 },
872 }

```

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

`__tag_struct_Ref_obj:n` this command is a helper command that is stored as a list in the Ref key of a structure.
`__tag_struct_Ref_label:n` They are executed when the structure elements are written in `__tag_struct_write_obj`.
`__tag_struct_Ref_dest:n` They are used in `__tag_struct_format_Ref`. They allow to add a Ref by object
`__tag_struct_Ref_num:n`

reference, label, destname and structure number

```

873 \cs_new_protected:Npn \__tag_struct_Ref_obj:nN #1 #2 %#1 a object reference
874 {
875   \tl_put_right:Ne#2
876   {
877     \c_space_tl#1
878   }
879 }
880
881 \cs_new_protected:Npn \__tag_struct_Ref_label:nN #1 #2 %#1 a label
882 {
883   \prop_get:NnNTF \g__tag_struct_label_num_prop {#1} \l__tag_tmpb_tl
884   {
885     \tl_put_right:Ne#2
886     {
887       \c_space_tl\tag_struct_object_ref:e{ \l__tag_tmpb_tl }
888     }
889   }
890   {
891     \msg_warning:nnn {tag}{struct-Ref-unknown}{Label~'#1'}
892   }
893 }
894 \cs_new_protected:Npn \__tag_struct_Ref_dest:nN #1 #2 %#1 a dest name
895 {
896   \prop_get:NnNTF \g__tag_struct_dest_num_prop {#1} \l__tag_tmpb_tl
897   {
898     \tl_put_right:Ne#2
899     {
900       \c_space_tl\tag_struct_object_ref:e{ \l__tag_tmpb_tl }
901     }
902   }
903   {
904     \msg_warning:nnn {tag}{struct-Ref-unknown}{Destination~'#1'}
905   }
906 }
907 \cs_new_protected:Npn \__tag_struct_Ref_num:nN #1 #2 %#1 a structure number
908 {
909   \tl_put_right:Ne#2
910   {
911     \c_space_tl\tag_struct_object_ref:e{ #1 }
912   }
913 }
914

```

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

```

ref (struct key)
E (struct key)
915 \keys_define:nn { __tag / struct }
916 {
917   ref .code:n          = % ref property
918   {
919     \clist_map_inline:on {#1}

```

```

920     {
921       \tag_struct_gput:nne
922       {\int_use:N \c@g__tag_struct_abs_int}{ref_label}{ ##1 }
923     }
924   },
925   E .code:n      = % E property
926   {
927     \str_set_convert:Nnon
928     \l__tag_tmpa_str
929     { #1 }
930     { default }
931     { utf16/hex }
932     \__tag_struct_prop_gput:nne
933     { \int_use:N \c@g__tag_struct_abs_int }
934     { E }
935     { <\l__tag_tmpa_str> }
936   },
937 }

```

AF (*struct key*) keys for the AF keys (associated files). They use commands from l3pdf file! The stream
AFref (*struct key*) variants use txt as extension to get the mimetype. TODO: check if this should be
AFinline (*struct key*) configurable. For math we will perhaps need another extension. AF/AFref is an array
AFinline-o (*struct key*) and can be used more than once, so we store it in a tl. which is expanded. AFinline
texsource (*struct key*) currently uses the fix extension txt. texsource is a special variant which creates a tex-file,
mathml (*struct key*) 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

```

938 \int_new:N\g__tag_struct_AFobj_int

939 \cs_generate_variant:Nn \pdf file_embed_stream:nnN {neN}
940 \cs_new_protected:Npn \__tag_struct_add_inline_AF:nn #1 #2
941 % #1 content, #2 extension
942 {
943   \tl_if_empty:nF{#1}
944   {
945     \group_begin:
946     \int_gincr:N \g__tag_struct_AFobj_int
947     \pdf file_embed_stream:neN
948     {#1}
949     {tag-AFfile\int_use:N\g__tag_struct_AFobj_int.#2}
950     \l__tag_tmpa_tl
951     \__tag_struct_add_AF:ee
952     { \int_use:N \c@g__tag_struct_abs_int }
953     { \l__tag_tmpa_tl }
954     \__tag_struct_prop_gput:nne
955     { \int_use:N \c@g__tag_struct_abs_int }
956     { AF }
957     {
958       [
959         \tl_use:c
960         { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
961       ]
962     }
963   }
964 }

```

```

963     \group_end:
964   }
965 }
966
967 \cs_generate_variant:Nn \__tag_struct_add_inline_AF:nn {on}

968 \cs_new_protected:Npn \__tag_struct_add_AF:nn #1 #2
969 % #1 struct num #2 object reference
970 {
971   \tl_if_exist:cTF
972   {
973     g__tag_struct_#1_AF_tl
974   }
975   {
976     \tl_gput_right:ce
977     { g__tag_struct_#1_AF_tl }
978     { \c_space_tl #2 }
979   }
980   {
981     \tl_new:c
982     { g__tag_struct_#1_AF_tl }
983     \tl_gset:ce
984     { g__tag_struct_#1_AF_tl }
985     { #2 }
986   }
987 }
988 \cs_generate_variant:Nn \__tag_struct_add_AF:nn {en,ee}
989 \keys_define:nn { __tag / struct }
990 {
991   AF .code:n      = % AF property
992   {
993     \pdf_object_if_exist:eTF {#1}
994     {
995       \__tag_struct_add_AF:ee
996       { \int_use:N \c@g__tag_struct_abs_int }{\pdf_object_ref:e {#1}}
997       \__tag_struct_prop_gput:nne
998       { \int_use:N \c@g__tag_struct_abs_int }
999       { AF }
1000      {
1001        [
1002          \tl_use:c
1003          { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
1004        ]
1005      }
1006    }
1007    {
1008      % message?
1009    }
1010  },
1011  AFref .code:n    = % AF property
1012  {
1013    \tl_if_empty:eF {#1}
1014    {
1015      \__tag_struct_add_AF:ee { \int_use:N \c@g__tag_struct_abs_int }{#1}

```

```

1016     \__tag_struct_prop_gput:nne
1017     { \int_use:N \c@g__tag_struct_abs_int }
1018     { AF }
1019     {
1020     [
1021     \tl_use:c
1022     { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
1023     ]
1024     }
1025   }
1026 },
1027 ,AFinline .code:n =
1028 {
1029   \__tag_struct_add_inline_AF:nn {#1}{txt}
1030 }
1031 ,AFinline-o .code:n =
1032 {
1033   \__tag_struct_add_inline_AF:on {#1}{txt}
1034 }
1035 ,texsource .code:n =
1036 {
1037   \group_begin:
1038   \pdfdict_put:nnn { l_pdffile/Filespec } {Desc}{(TeX-source)}
1039   \pdfdict_put:nnn { l_pdffile/Filespec }{AFRelationship} { /Source }
1040   \__tag_struct_add_inline_AF:on {#1}{tex}
1041   \group_end:
1042 }
1043 ,mathml .code:n =
1044 {
1045   \group_begin:
1046   \pdfdict_put:nnn { l_pdffile/Filespec } {Desc}{(mathml~representation)}
1047   \pdfdict_put:nnn { l_pdffile/Filespec }{AFRelationship} { /Supplement }
1048   \pdfdict_put:nne { l_pdffile }{Subtype}
1049   { \pdf_name_from_unicode_e:n{application/mathml+xml} }
1050   \__tag_struct_add_inline_AF:on {#1}{xml}
1051   \group_end:
1052 }
1053 }

```

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!

```

1054 \keys_define:nn { __tag / setup }
1055 {
1056   root-AF .code:n =
1057   {
1058     \pdf_object_if_exist:nTF {#1}
1059     {
1060       \__tag_struct_add_AF:ee { 1 }{\pdf_object_ref:n {#1}}
1061       \__tag_struct_prop_gput:nne
1062       { 1 }
1063       { AF }
1064       {
1065         [

```

```

1066         \tl_use:c
1067         { g__tag_struct_1_AF_tl }
1068     ]
1069 }
1070 }
1071 {
1072 }
1073 }
1074 },
1075 }

```

root-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

```

1076 \keys_define:nn { __tag / setup }
1077 {
1078     root-supplemental-file .code:n =
1079     {
1080         \group_begin:
1081         \pdfdict_put:nnn {l_pdffile/Filespec} {AFRelationship}{/Supplement}
1082         \int_gincr:N \g__tag_unique_cnt_int
1083         \pdffile_embed_file:eee
1084         {#1}
1085         {#1}
1086         {__tag_latex_css \int_use:N \g__tag_unique_cnt_int}
1087         \keys_set:nn
1088         {__tag / setup}
1089         {root-AF={__tag_latex_css \int_use:N \g__tag_unique_cnt_int}}
1090         \group_end:
1091     }
1092 }

```

log-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.

```

1093 \keys_define:nn { __tag / setup }
1094 {
1095     catalog-supplemental-file .code:n =
1096     {
1097         \group_begin:
1098         \pdfdict_put:nnn {l_pdffile/Filespec} {AFRelationship}{/Supplement}
1099         \int_gincr:N \g__tag_unique_cnt_int
1100         \pdffile_embed_file:eee
1101         {#1}
1102         {#1}
1103         {__tag_latex_css \int_use:N \g__tag_unique_cnt_int}
1104         \pdfmanagement_add:nne
1105         {Catalog}
1106         {AF}
1107         {\pdf_object_ref:e{__tag_latex_css \int_use:N \g__tag_unique_cnt_int }}
1108         \group_end:
1109     }
1110 }

```


6 User commands

We allow to set a language by default

`\l__tag_struct_lang_tl`

```
1111 \tl_new:N \l__tag_struct_lang_tl
1112 \</package>
```

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

`\tag_struct_begin:n`

`\tag_struct_end:`

```
1113 <base>\cs_new_protected:Npn \tag_struct_begin:n #1 { \int_gincr:N \c@g__tag_struct_abs_int }
1114 <base>\cs_new_protected:Npn \tag_struct_end: {}
1115 <base>\cs_new_protected:Npn \tag_struct_end:n {}
1116 <*package | debug>
1117 <package>\cs_set_protected:Npn \tag_struct_begin:n #1 %#1 key-val
1118 <debug>\cs_set_protected:Npn \tag_struct_begin:n #1 %#1 key-val
1119 {
1120 <package>\__tag_check_if_active_struct:T
1121 <debug>\__tag_check_if_active_struct:TF
1122 {
1123   \group_begin:
1124   \int_gincr:N \c@g__tag_struct_abs_int
1125   \__tag_prop_new:c { g__tag_struct_\int_eval:n { \c@g__tag_struct_abs_int }_prop }
1126 <debug>   \prop_new:c { g__tag_struct_debug_\int_eval:n { \c@g__tag_struct_abs_int }_prop }
1127   \__tag_new_output_prop_handler:n { \int_eval:n { \c@g__tag_struct_abs_int } }
1128   \__tag_seq_new:c { g__tag_struct_kids_\int_eval:n { \c@g__tag_struct_abs_int }_seq }
1129 <debug>   \seq_new:c { g__tag_struct_debug_kids_\int_eval:n { \c@g__tag_struct_abs_int }_seq }
1130   \pdf_object_new_indexed:nn { __tag/struct }
1131   { \c@g__tag_struct_abs_int }
1132   \__tag_struct_prop_gput:nnn
1133   { \int_use:N \c@g__tag_struct_abs_int }
1134   { Type }
1135   { /StructElem }
1136   \tl_if_empty:NF \l__tag_struct_lang_tl
1137   {
1138     \__tag_struct_prop_gput:nne
1139     { \int_use:N \c@g__tag_struct_abs_int }
1140     { Lang }
1141     { (\l__tag_struct_lang_tl) }
1142   }
1143   \__tag_struct_prop_gput:nnn
1144   { \int_use:N \c@g__tag_struct_abs_int }
1145   { Type }
1146   { /StructElem }
1147
1148   \tl_set:Nn \l__tag_struct_stack_parent_tmpa_tl {-1}
1149   \keys_set:nn { __tag / struct } { #1 }
1150
1151   \__tag_struct_set_tag_info:eoo
1152   { \int_use:N \c@g__tag_struct_abs_int }
1153   { g__tag_struct_tag_tl }
1154   { g__tag_struct_tag_NS_tl }
1155   \__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.

```

1155     \int_compare:nNnT { \l__tag_struct_stack_parent_tmpa_tl } = { -1 }
1156     {
1157         \seq_get:NnF
1158             \g__tag_struct_stack_seq
1159             \l__tag_struct_stack_parent_tmpa_tl
1160         {
1161             \msg_error:nn { tag } { struct-faulty-nesting }
1162         }
1163     }
1164     \seq_gpush:NV \g__tag_struct_stack_seq \c@g__tag_struct_abs_int
1165     \__tag_role_get:ooNN
1166     { \g__tag_struct_tag_tl }
1167     { \g__tag_struct_tag_NS_tl }
1168     \l__tag_struct_roletag_tl
1169     \l__tag_struct_roletag_NS_tl

```

We push the role tag on the stack:

```

1170     \seq_gpush:Ne \g__tag_struct_tag_stack_seq
1171     {{\g__tag_struct_tag_tl}{\l__tag_struct_roletag_tl}}
1172     \tl_gset:NV \g__tag_struct_stack_current_tl \c@g__tag_struct_abs_int
1173     \__tag_struct_set_attribute:
1174     %\seq_show:N \g__tag_struct_stack_seq

```

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

```

1175     \__tag_struct_prop_gput:nne
1176     { \int_use:N \c@g__tag_struct_abs_int }
1177     { rolemap }
1178     {
1179         {\l__tag_struct_roletag_tl}{\l__tag_struct_roletag_NS_tl}
1180     }

```

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.

```

1181     \str_case:onTF { \l__tag_struct_roletag_tl }
1182     {
1183         {Part} {}
1184         {Div} {}
1185         {NonStruct} {}
1186     }
1187     {
1188         \bool_if:NTF \l__tag_struct_elem_stash_bool
1189         {
1190             \__tag_struct_prop_gput:nne

```

```

1191         { \int_use:N \c@g__tag_struct_abs_int }
1192         { parentrole }
1193         {
1194             {\l__tag_struct_parenttag_tl}{\l__tag_struct_parenttag_NS_tl}
1195         }
1196     }
1197     {
1198         \prop_get:cnNT
1199         { g__tag_struct_ \l__tag_struct_stack_parent_tmpa_tl _prop }
1200         { parentrole }
1201         \l__tag_get_tmpc_tl
1202         {
1203             \__tag_struct_prop_gput:nno
1204             { \int_use:N \c@g__tag_struct_abs_int }
1205             { parentrole }
1206             {
1207                 \l__tag_get_tmpc_tl
1208             }
1209         }
1210     }
1211 }
1212 {
1213     \__tag_struct_prop_gput:nne
1214     { \int_use:N \c@g__tag_struct_abs_int }
1215     { parentrole }
1216     {
1217         {\l__tag_struct_roletag_tl}{\l__tag_struct_roletag_NS_tl}
1218     }
1219 }
1220 \bool_if:NF
1221 \l__tag_struct_elem_stash_bool
1222 {

```

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.

```

1223     \socket_use:nn{tag/check/parent-child}
1224     {
1225         \__tag_struct_check_parent_child:oo
1226         { \l__tag_struct_stack_parent_tmpa_tl }
1227         { \int_use:N \c@g__tag_struct_abs_int }
1228     }

```

Set the Parent structure number.

```

1229     \__tag_struct_prop_gput:nne
1230     { \int_use:N \c@g__tag_struct_abs_int }
1231     { parentnum }
1232     {
1233         \l__tag_struct_stack_parent_tmpa_tl
1234     }

```

```

1235         %record this structure as kid:
1236         %\tl_show:N \g__tag_struct_stack_current_tl
1237         %\tl_show:N \l__tag_struct_stack_parent_tmpa_tl
1238         \use:c { __tag_struct_kid_struct_gput_ \l__tag_struct_addkid_tl :ee }
1239         { \l__tag_struct_stack_parent_tmpa_tl }
1240         { \g__tag_struct_stack_current_tl }
1241         %\prop_show:c { g__tag_struct_\g__tag_struct_stack_current_tl _prop }
1242         %\seq_show:c {g__tag_struct_kids_\l__tag_struct_stack_parent_tmpa_tl _seq}
1243     }

```

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.

```

1244 <debug>         \prop_gset_eq:cc
1245 <debug>         { g__tag_struct_debug_\int_eval:n { \c@g__tag_struct_abs_int}_prop }
1246 <debug>         { g__tag_struct_\int_eval:n { \c@g__tag_struct_abs_int}_prop }
1247 <debug>         \prop_gput:cne
1248 <debug>         { g__tag_struct_debug_\int_eval:n { \c@g__tag_struct_abs_int}_prop }
1249 <debug>         { parentnum }
1250 <debug>         {
1251 <debug>             \bool_if:NTF \l__tag_struct_elem_stash_bool
1252 <debug>             {no-parent:~stashed}
1253 <debug>             {
1254 <debug>                 \l__tag_struct_stack_parent_tmpa_tl\c_space_tl =~
1255 <debug>                 \prop_item:cn{ g__tag_struct_\l__tag_struct_stack_parent_tmpa_tl _prop
1256 <debug>             }
1257 <debug>         }
1258 <debug>         \prop_gput:cne
1259 <debug>         { g__tag_struct_debug_\int_eval:n { \c@g__tag_struct_abs_int}_prop }
1260 <debug>         { NS }
1261 <debug>         { \g__tag_struct_tag_NS_tl }

1262         %\prop_show:c { g__tag_struct_\g__tag_struct_stack_current_tl _prop }
1263         %\seq_show:c {g__tag_struct_kids_\l__tag_struct_stack_parent_tmpa_tl _seq}
1264 <debug> \__tag_debug_struct_begin_insert:n { #1 }
1265 \group_end:
1266 }
1267 <debug>{ \__tag_debug_struct_begin_ignore:n { #1 }}
1268 }
1269 <package>\cs_set_protected:Nn \tag_struct_end:
1270 <debug>\cs_set_protected:Nn \tag_struct_end:
1271 { %take the current structure num from the stack:
1272     %the objects are written later, lua mode hasn't all needed info yet
1273     %\seq_show:N \g__tag_struct_stack_seq
1274 <package>\__tag_check_if_active_struct:T
1275 <debug>\__tag_check_if_active_struct:TF
1276 {
1277     \seq_gpop:NN \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
1278     \seq_gpop:NNTF \g__tag_struct_stack_seq \l__tag_tmpa_tl
1279     {
1280         \__tag_check_info_closing_struct:o { \g__tag_struct_stack_current_tl }
1281     }
1282     { \__tag_check_no_open_struct: }

```

```

1283 % get the previous one, shouldn't be empty as the root should be there
1284 \seq_get:NNTF \g__tag_struct_stack_seq \l__tag_tmpa_tl
1285 {
1286   \tl_gset:No \g__tag_struct_stack_current_tl { \l__tag_tmpa_tl }
1287   \__tag_struct_set_attribute:
1288 }
1289 {
1290   \__tag_check_no_open_struct:
1291 }
1292 \seq_get:NNT \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
1293 {
1294   \tl_gset:Ne \g__tag_struct_tag_tl
1295   { \exp_last_unbraced:No\use_i:nn { \l__tag_tmpa_tl } }
1296   \prop_get:NoNT\g__tag_role_tags_NS_prop { \g__tag_struct_tag_tl } \l__tag_tmpa_tl
1297   {
1298     \tl_gset:Ne \g__tag_struct_tag_NS_tl { \l__tag_tmpa_tl }
1299   }
1300 }
1301 <debug>\__tag_debug_struct_end_insert:
1302 }
1303 <debug>\__tag_debug_struct_end_ignore:}
1304 }
1305
1306 \cs_set_protected:Npn \tag_struct_end:n #1
1307 {
1308   <debug> \__tag_check_if_active_struct:T{\__tag_debug_struct_end_check:n{#1}}
1309   \tag_struct_end:
1310 }
1311 </package | debug>

```

(End of definition for \tag_struct_begin:n and \tag_struct_end:. These functions are documented on page 113.)

\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.

```

1312 <base>\cs_new_protected:Npn \tag_struct_use:n #1 {}
1313 <*package | debug>
1314 \cs_set_protected:Npn \tag_struct_use:n #1 %#1 is the label
1315 {
1316   \__tag_check_if_active_struct:T
1317   {
1318     \prop_if_exist:cTF
1319     { \g__tag_struct_property_ref:enn{tagpdfstruct-#1}{tagstruct}{unknown}_prop } %
1320     {
1321       \__tag_check_struct_used:n {#1}
1322       \tl_set:Ne \l__tag_get_child_tmpa_tl
1323       { \property_ref:enn{tagpdfstruct-#1}{tagstruct}{1} }

```

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

```

1324   \__tag_struct_kid_struct_gput_right:ee
1325   { \g__tag_struct_stack_current_tl }
1326   { \l__tag_get_child_tmpa_tl }

```

add the current structure to the labeled one as parents

```

1327      \__tag_prop_gput:cne
1328      { g__tag_struct_ \l__tag_get_child_tmpa_tl _prop }
1329      { parentnum }
1330      {
1331      \g__tag_struct_stack_current_tl
1332      }

```

debug code

```

1333 <debug>      \prop_gput:cne
1334 <debug>      { g__tag_struct_debug_ \l__tag_get_child_tmpa_tl _prop }
1335 <debug>      { parentnum }
1336 <debug>      {
1337 <debug>      \g__tag_struct_stack_current_tl\c_space_tl=~
1338 <debug>      \g__tag_struct_tag_tl
1339 <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 rolemapping 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.

```

1340      \socket_use:nn{tag/check/parent-child}
1341      {
1342      \__tag_struct_use_check_parent_child:oo
1343      { \g__tag_struct_stack_current_tl }
1344      { \l__tag_get_child_tmpa_tl }
1345      }
1346      }
1347      {
1348      \msg_warning:nnn{ tag }{struct-label-unknown}{#1}
1349      }
1350      }
1351      }
1352 </package | debug>

```

(End of definition for `\tag_struct_use:n`. This function is documented on page 113.)

`\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.

```

1353 <base>\cs_new_protected:Npn \tag_struct_use_num:n #1 {}
1354 <*package | debug>
1355 \cs_set_protected:Npn \tag_struct_use_num:n #1 %#1 is structure number
1356 {
1357   \__tag_check_if_active_struct:T
1358   {
1359     \prop_if_exist:cTF
1360     { g__tag_struct_#1_prop } %

```

```

1361     {
1362         \prop_get:cnNT
1363         {g__tag_struct_#1_prop}
1364         {parentnum}
1365         \l__tag_tmpa_tl
1366         {
1367             \msg_warning:nnn { tag } {struct-used-twice} {#1}
1368         }

```

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

```

1369         \__tag_struct_kid_struct_gput_right:ee
1370         { \g__tag_struct_stack_current_tl }
1371         { #1 }

```

add the current structure to #1 as parent

```

1372         \__tag_struct_prop_gput:nne
1373         { #1 }
1374         { parentnum }
1375         {
1376             \g__tag_struct_stack_current_tl
1377         }
1378     <debug> \prop_gput:cne
1379     <debug> { g__tag_struct_debug_#1_prop }
1380     <debug> { parentnum }
1381     <debug> {
1382     <debug> \g__tag_struct_stack_current_tl\c_space_tl=~
1383     <debug> \g__tag_struct_tag_tl
1384     <debug> }

```

check if the tag is allowed as child.

```

1385         \socket_use:nn{tag/check/parent-child}
1386         {
1387             \__tag_struct_use_check_parent_child:oo
1388             {\g__tag_struct_stack_current_tl}
1389             {#1}
1390         }
1391     }
1392     {
1393         \msg_warning:nnn{ tag }{struct-label-unknown}{#1}
1394     }
1395 }
1396 }
1397 </package | debug>

```

(End of definition for \tag_struct_use_num:n. This function is documented on page 113.)

\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.

```

1398 <*package>
1399 \cs_new:Npn \tag_struct_object_ref:n #1
1400 {
1401   \pdf_object_ref_indexed:nn {__tag/struct}{ #1 }
1402 }
1403 \cs_generate_variant:Nn \tag_struct_object_ref:n {e}
1404 </package>

```

(End of definition for `\tag_struct_object_ref:n`. This function is documented on page 113.)

`\tag_struct_gput:nnn` This is a command that allows to update the data of a structure. This often can't be 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 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 then added to the `/A`.

```

1405 <(base)\cs_new_protected:Npn \tag_struct_gput:nnn #1 #2 #3{>
1406 <*package>
1407 \cs_set_protected:Npn \tag_struct_gput:nnn #1 #2 #3
1408 {
1409   \cs_if_exist_use:cF {__tag_struct_gput_data_#2:nn}
1410   { %warning??
1411     \use_none:nn
1412   }
1413   {#1}{#3}
1414 }
1415 \cs_generate_variant:Nn \tag_struct_gput:nnn {ene,nne}
1416 </package>

```

(End of definition for `\tag_struct_gput:nnn`. This function is documented on page 113.)

`__tag_struct_gput_data_ref_aux:nnn`

```

1417 <*package>
1418 \cs_new_protected:Npn \__tag_struct_gput_data_ref_aux:nnn #1 #2 #3
1419 % #1 receiving struct num, #2 key word #3 value
1420 {
1421   \prop_get:cnNTF
1422   { g__tag_struct_#1_prop }
1423   {Ref}
1424   \l__tag_get_tmpc_tl
1425   {
1426     \tl_put_right:No \l__tag_get_tmpc_tl
1427     {\cs:w __tag_struct_Ref_#2:nN \cs_end: {#3},}
1428   }

```



```

1429     {
1430         \tl_set:No \l__tag_get_tmpc_tl
1431         {\cs:w __tag_struct_Ref_#2:nN \cs_end: {#3},}
1432     }
1433     \__tag_struct_prop_gput:nno
1434     { #1 }
1435     { Ref }
1436     { \l__tag_get_tmpc_tl }
1437 }
1438 \cs_new_protected:Npn \__tag_struct_gput_data_ref:nn #1 #2
1439 {
1440     \__tag_struct_gput_data_ref_aux:nnn {#1}{obj}{#2}
1441 }
1442 \cs_new_protected:Npn \__tag_struct_gput_data_ref_label:nn #1 #2
1443 {
1444     \__tag_struct_gput_data_ref_aux:nnn {#1}{label}{#2}
1445 }
1446 \cs_new_protected:Npn \__tag_struct_gput_data_ref_dest:nn #1 #2
1447 {
1448     \__tag_struct_gput_data_ref_aux:nnn {#1}{dest}{#2}
1449 }
1450 \cs_new_protected:Npn \__tag_struct_gput_data_ref_num:nn #1 #2
1451 {
1452     \__tag_struct_gput_data_ref_aux:nnn {#1}{num}{#2}
1453 }
1454
1455 \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

```

1456 \cs_new_protected:Npn \__tag_struct_gput_data_attribute:nn #1 #2
1457 {
1458     \pdf_object_unnamed_write:nn {dict} {#2}
1459     \prop_get:cnNTF { g__tag_struct_#1_prop }{A} \l__tag_tmpa_tl
1460     {
1461         \tl_remove_once:Nn\l__tag_tmpa_tl{[] }
1462         \tl_remove_once:Nn\l__tag_tmpa_tl{[] }
1463         \__tag_prop_gput:cne { g__tag_struct_#1_prop }
1464         { A }
1465         {
1466             [ \l__tag_tmpa_tl \c_space_tl \pdf_object_ref_last: ]
1467         }
1468     }
1469     {
1470         \__tag_prop_gput:cne { g__tag_struct_#1_prop }
1471         { A }
1472         { \pdf_object_ref_last: }
1473     }
1474 }

```

(End of definition for __tag_struct_gput_data_attribute:nn.)

`\tag_struct_insert_annot:nn` 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:.`

`\tag_struct_insert_annot:ee`

`\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.

```

1475 \cs_new_protected:Npn \tag_struct_insert_annot:nn #1 #2 %#1 should be an object reference
1476                                     %#2 struct parent num
1477 {
1478   \__tag_check_if_active_struct:T
1479   {
1480     \__tag_struct_insert_annot:nn {#1}{#2}
1481   }
1482 }
1483
1484 \cs_generate_variant:Nn \tag_struct_insert_annot:nn {xx,ee}
1485 \cs_new:Npn \tag_struct_parent_int: {\int_use:c { c@g__tag_parenttree_obj_int }}
1486
1487 \</package>
1488

```

(End of definition for `\tag_struct_insert_annot:nn` and `\tag_struct_parent_int:.` These functions are documented on page 113.)

7 Attributes and attribute classes

```

1489 \<header>
1490 \ProvidesExplPackage {tagpdf-attr-code} {2025-07-16} {0.99u}
1491   {part of tagpdf - code related to attributes and attribute classes}
1492 \</header>

```

7.1 Variables

`\g__tag_attr_entries_prop` `\g__@@_attr_entries_prop` will store attribute names and their dictionary content.

`\g__tag_attr_class_used_prop` `\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`

`\g__tag_attr_objref_prop`

`\l__tag_attr_value_tl`

```

1493 \<package>
1494 \prop_new:N \g__tag_attr_entries_prop
1495 \prop_new_linked:N \g__tag_attr_class_used_prop
1496 \tl_new:N \l__tag_attr_value_tl
1497 \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.

```

1498 \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` This allows to define attributes. Defined attributes are stored in a global property. `role/new-attribute (setup-key)` `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},
}

1499 \cs_new_protected:Npn \__tag_attr_new_entry:nn #1 #2 %#1:name, #2: content
1500   {
1501     \prop_gput:Nen \g__tag_attr_entries_prop
1502       {\pdf_name_from_unicode_e:n{#1}}{#2}
1503   }
1504
1505 \cs_generate_variant:Nn \__tag_attr_new_entry:nn {ee}
1506 \keys_define:nn { __tag / setup }
1507   {
1508     role/new-attribute .code:n =
1509       {
1510         \__tag_attr_new_entry:nn #1
1511       }
1512   }
```

deprecated name

```
1512   ,newattribute .code:n =
1513     {
1514       \__tag_attr_new_entry:nn #1
1515     },
1516 }
```

(End of definition for `__tag_attr_new_entry:nn`, `role/new-attribute (setup-key)`, and `newattribute (deprecated)`. These functions are documented on page 116.)

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

```
1517 \keys_define:nn { __tag / struct }
1518   {
1519     attribute-class .code:n =
1520       {
1521         \clist_set:Nc \l__tag_tmpa_clist { #1 }
1522         \seq_set_from_clist:NN \l__tag_tmpb_seq \l__tag_tmpa_clist
1523       }
1524   }
```

we convert the names into pdf names with slash

```

1523 \seq_set_map_e:NNn \l__tag_tmpa_seq \l__tag_tmpb_seq
1524 {
1525   \pdf_name_from_unicode_e:n {##1}
1526 }
1527 \seq_map_inline:Nn \l__tag_tmpa_seq
1528 {
1529   \prop_get:NnNF \g__tag_attr_entries_prop {##1}\l__tag_tmpa_tl
1530   {
1531     \msg_error:nnn { tag } { attr-unknown } { ##1 }
1532   }
1533   \prop_gput:Nnn\g__tag_attr_class_used_prop { ##1} {}
1534 }
1535 \tl_set:Ne \l__tag_tmpa_tl
1536 {
1537   \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}
1538   \seq_use:Nn \l__tag_tmpa_seq { \c_space_tl }
1539   \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}]
1540 }
1541 \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 0 }
1542 {
1543   \__tag_struct_prop_gput:nne
1544   { \int_use:N \c@g__tag_struct_abs_int }
1545   { C }
1546   { \l__tag_tmpa_tl }
1547   %\prop_show:c { g__tag_struct\_int_eval:n {\c@g__tag_struct_abs_int}_prop }
1548 }
1549 }
1550 }

```

attribute (struct key)

```

1551 \keys_define:nn { __tag / struct }
1552 {
1553   attribute .code:n = % A property (attribute, value currently a dictionary)
1554   {
1555     \clist_set:Ne \l__tag_tmpa_clist { #1 }
1556     \clist_if_empty:NF \l__tag_tmpa_clist
1557     {
1558       \seq_set_from_clist:NN \l__tag_tmpb_seq \l__tag_tmpa_clist

```

we convert the names into pdf names with slash

```

1559 \seq_set_map_e:NNn \l__tag_tmpa_seq \l__tag_tmpb_seq
1560 {
1561   \pdf_name_from_unicode_e:n {##1}
1562 }
1563 \tl_set:Ne \l__tag_attr_value_tl
1564 {
1565   \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}%
1566 }
1567 \seq_map_inline:Nn \l__tag_tmpa_seq
1568 {
1569   \prop_get:NnNF \g__tag_attr_entries_prop {##1}\l__tag_tmp_unused_tl
1570   {

```

```

1571         \msg_error:nnn { tag } { attr-unknown } { ##1 }
1572     }
1573     \prop_get:NnNF \g__tag_attr_objref_prop {##1}\l__tag_tmpa_tl
1574     {%\prop_show:N \g__tag_attr_entries_prop
1575       \pdf_object_unnamed_write:ne
1576       { dict }
1577       {
1578         \prop_item:Nn\g__tag_attr_entries_prop {##1}
1579       }
1580       \prop_gput:Nne \g__tag_attr_objref_prop {##1} {\pdf_object_ref_last:}
1581     }
1582     \tl_put_right:Ne \l__tag_attr_value_tl
1583     {
1584       \c_space_tl
1585       \prop_item:Nn \g__tag_attr_objref_prop {##1}
1586     }
1587     % \tl_show:N \l__tag_attr_value_tl
1588     }
1589     \tl_put_right:Ne \l__tag_attr_value_tl
1590     { %[
1591       \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{]]}%
1592     }
1593     % \tl_show:N \l__tag_attr_value_tl
1594     \__tag_struct_prop_gput:nne
1595     { \int_use:N \c@g__tag_struct_abs_int }
1596     { A }
1597     { \l__tag_attr_value_tl }
1598   }
1599 },
1600 }
1601 \end{package}

```

The tagpdf-luatex.def
 Driver for luatex
 Part of the tagpdf package
 Ulrike Fischer
 Version 0.99u, released 2025-07-16

Part IX

```
1 <@@=tag>
2 <*luatex>
3 \ProvidesExplFile {tagpdf-luatex.def} {2025-07-16} {0.99u}
4 {tagpdf~driver~for~luatex}
```

1 Loading the lua

The space code requires that the fall back font has been loaded and initialized, so we force that first. But perhaps this could be done in the kernel.

```
5 {
6   \fontencoding{TU}\fontfamily{lmr}\fontseries{m}\fontshape{n}\fontsize{10pt}{10pt}\selectfont
7 }
8 \lua_now:e { tagpdf=require('tagpdf.lua') }
```

The following defines wrappers around prop and seq commands to store the data also in lua tables. I probably want also lua tables I put them in the ltx.@@.tables namespaces. The tables will be named like the variables but without backslash. To access such a table with a dynamical name create a string and then use ltx.@@.tables[string]. Old code, I'm not quite sure if this was a good idea. Now I have mix of table in ltx.@@.tables and ltx.@@.mc/struct. And a lot is probably not needed. TODO: this should be cleaned up, but at least roles are currently using the table!

```
    \__tag_prop_new:N
    \__tag_seq_new:N
    \__tag_prop_gput:Nnn 9 \cs_set_protected:Npn \__tag_prop_new:N #1
    \__tag_seq_gput_right:Nn10 {
    \__tag_seq_gput_left:Nn11   \prop_new:N #1
    \__tag_seq_item:cn12       \lua_now:e { ltx.__tag.tables['\cs_to_str:N#1'] = {} }
    \__tag_prop_item:cn13     }
    \__tag_seq_show:N14       \cs_set_protected:Npn \__tag_prop_new_linked:N #1
    \__tag_prop_show:N15     {
    \__tag_prop_show:N16     \prop_new_linked:N #1
    \__tag_prop_show:N17     \lua_now:e { ltx.__tag.tables['\cs_to_str:N#1'] = {} }
    \__tag_prop_show:N18     }
    \__tag_prop_show:N19
    \__tag_prop_show:N20
    \__tag_prop_show:N21
    \cs_set_protected:Npn \__tag_seq_new:N #1
    \__tag_seq_new:N22 {
    \__tag_seq_new:N23   \seq_new:N #1
    \__tag_seq_new:N24   \lua_now:e { ltx.__tag.tables['\cs_to_str:N#1'] = {} }
    \__tag_seq_new:N25   }
    \__tag_seq_new:N26
    \__tag_seq_new:N27
    \__tag_seq_new:N28
    \cs_set_protected:Npn \__tag_prop_gput:Nnn #1 #2 #3
    \__tag_prop_gput:Nnn29 {
    \__tag_prop_gput:Nnn30 { #2 } { #3 }
    \__tag_prop_gput:Nnn31 \lua_now:e { ltx.__tag.tables['\cs_to_str:N#1'] ["#2"] = "\lua_escape:n{#3}" }
    \__tag_prop_gput:Nnn32 }
    \__tag_prop_gput:Nnn33
    \__tag_prop_gput:Nnn34
```

```

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   = "0.99u",      --TAGVERSION
79   date      = "2025-07-16", --TAGDATE

```

```

80     description = "tagpdf lua code",
81     license      = "The LATEX Project Public License 1.3c"
82 }
83
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 up
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 of m
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 th
123 ltx.__tag.func.mark_page_elements(box,mcpagecnt,mcntprev,mcopen,name,mctypeprev) : the main fu
124 ltx.__tag.func.mark_shipout (): a wrapper around the core function which inserts the last EMC
125 ltx.__tag.func.fill_parent_tree_line (page): outputs the entries of the parenttree for this pag
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 posit
129 ltx.__tag.trace.show_mc_data (num,loglevel): shows ltx.__tag.mc[num] is the current log level i
130 ltx.__tag.trace.show_all_mc_data (max,loglevel): shows a maximum about mc's if the current log
131 ltx.__tag.trace.show_seq: shows a sequence (array)

```



```

132 ltx.__tag.trace.show_struct_data (num): shows data of structure num
133 ltx.__tag.trace.show_prop: shows a prop
134 ltx.__tag.trace.log
135 ltx.__tag.trace.showspace : boolean
136
137 ltx.tag.get_structnum: number, shows the current structure number
138 ltx.tag.get_structnum_next: number, shows the next structure number
139 --]]
140

```

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`).

```

141 local mctypeattributeid = luatexbase.new_attribute ("g__tag_mc_type_attr")
142 local mcntattributeid   = luatexbase.new_attribute ("g__tag_mc_cnt_attr")
143 local structnumattributeid = luatexbase.new_attribute ("g__tag_structnum_attr")
144 local iwspaceOffattributeid = luatexbase.new_attribute ("g__tag_interwordspaceOff_attr")
145 local iwspaceattributeid = luatexbase.new_attribute ("g__tag_interwordspace_attr")
146 local iwfontattributeid = luatexbase.new_attribute ("g__tag_interwordfont_attr")

```

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

```

147 local tagunmarkedbool= token.create("g__tag_tagunmarked_bool")
148 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.

```

149 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`.

```

150 local catlatex      = luatexbase.registernumber("catcodetable@latex")
151 local tableinsert   = table.insert
152 local nodeid        = node.id
153 local nodecopy      = node.copy
154 local nodegetattribute = node.get_attribute
155 local nodesetattribute = node.set_attribute
156 local nodehasattribute = node.has_attribute
157 local nodenew       = node.new
158 local nodetail      = node.tail
159 local nodeslide     = node.slide
160 local noderemove    = node.remove
161 local nodetraverseid = node.traverse_id
162 local nodetraverse  = node.traverse
163 local nodeinsertafter = node.insert_after

```

```

164 local nodeinsertbefore = node.insert_before
165 local pdfpageref       = pdf.pageref
166
167 local fonthashes        = fonts.hashes
168 local identifiers       = fonthashes.identifiers
169 local fontid            = font.id
170
171 local HLIST             = node.id("hlist")
172 local VLIST             = node.id("vlist")
173 local RULE              = node.id("rule")
174 local DISC              = node.id("disc")
175 local GLUE              = node.id("glue")
176 local GLYPH            = node.id("glyph")
177 local KERN              = node.id("kern")
178 local PENALTY           = node.id("penalty")
179 local LOCAL_PAR         = node.id("local_par")
180 local MATH              = node.id("math")
181
182 local NEXT = next
183 local explicit_disc = 1
184 local regular_disc = 3

```

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

```

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

```

\
203 local __tag_get_struct_num =
204 function()
205     local a = token.get_macro("g__tag_struct_stack_current_tl")
206     return a
207 end

```

```

208
209 local __tag_get_struct_counter =
210   function()
211     local a = tex.getcount("c@g__tag_struct_abs_int")
212     return a
213   end
214
215 local __tag_get_struct_num_next =
216   function()
217     local a = tex.getcount("c@g__tag_struct_abs_int") + 1
218     return a
219   end
220
221 ltx.tag.get_struct_num = __tag_get_struct_num
222 ltx.tag.get_struct_counter = __tag_get_struct_counter
223 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` 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.

```

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

```

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

```

(End of definition for ltx.__tag.trace.show_seq.)

`__tag_pairs_prop` This shows the content of a prop as stored in the tables table. It is used by the `\@@_prop_show:N` function.

`ltx.__tag.trace.show_prop`

```

241 local __tag_pairs_prop =
242   function (prop)
243     local a = {}
244     for n in pairs(prop) do tableinsert(a, n) end
245     table.sort(a)
246     local i = 0          -- iterator variable
247     local iter = function () -- iterator function
248       i = i + 1
249       if a[i] == nil then return nil
250       else return a[i], prop[a[i]]
251     end
252   end
253   return iter
254 end
255
256
257 function ltx.__tag.trace.show_prop (prop)
258   if (type(prop) == "table") then
259     for i,v in __tag_pairs_prop (prop) do
260       __tag_log ("[" .. i .. "]" => " .. tostring(v),1)
261     end
262   else
263     __tag_log ("prop " .. tostring(prop) .. " not found or not a table",1)
264   end
265 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

```

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

```

281 function ltx.__tag.trace.show_all_mc_data (min,max,loglevel)
282   for i = min, max do

```

```

283 ltx.__tag.trace.show_mc_data (i,loglevel)
284 end
285 texio.write_nl("")
286 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.

```

287 function ltx.__tag.trace.show_struct_data (num)
288 if ltx.__tag and ltx.__tag.struct and ltx.__tag.struct[num] then
289 for k,v in ipairs(ltx.__tag.struct[num]) do
290 __tag_log ("struct "..num..": "..tostring(k)..=>"..tostring(v),1)
291 end
292 else
293 __tag_log ("struct "..num.." not found ",1)
294 end
295 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.

```

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

```

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

```

310 ltx.__tag.tables.role_tag_attribute = {}
311 ltx.__tag.tables.role_attribute_tag = {}

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

```

ltx.__tag.func.alloctag

```

312 local __tag_alloctag =
313 function (tag)
314     if not ltx.__tag.tables.role_tag_attribute[tag] then
315         table.insert(ltx.__tag.tables.role_attribute_tag,tag)
316         ltx.__tag.tables.role_tag_attribute[tag]=#ltx.__tag.tables.role_attribute_tag
317         __tag_log ("Add "..tag.." "..ltx.__tag.tables.role_tag_attribute[tag],3)
318     end
319 end
320 ltx.__tag.func.alloctag = __tag_alloctag

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

```

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

```

321 local __tag_get_num_from =
322 function (tag)
323     if ltx.__tag.tables.role_tag_attribute[tag] then
324         a= ltx.__tag.tables.role_tag_attribute[tag]
325     else
326         a= -1
327     end
328     return a
329 end
330
331 ltx.__tag.func.get_num_from = __tag_get_num_from
332
333 function ltx.__tag.func.output_num_from (tag)
334     local num = __tag_get_num_from (tag)
335     tex.sprint(catlatex,num)
336     if num == -1 then
337         __tag_log ("Unknown tag "..tag.." used")
338     end
339 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 These functions are the opposites to the previous function: they take as argument a
ltx.__tag.func.get_tag_from number (the attribute value) and return the string **tag**. The first function outputs the
ltx.__tag.func.output_tag_from string for lua, while the **output** function outputs to tex.

```

340 local __tag_get_tag_from =
341 function (num)
342     if ltx.__tag.tables.role_attribute_tag[num] then
343         a = ltx.__tag.tables.role_attribute_tag[num]

```

```

344     else
345         a= "UNKNOWN"
346     end
347     return a
348 end

```

```

349
350 ltx.__tag.func.get_tag_from = __tag_get_tag_from
351
352 function ltx.__tag.func.output_tag_from (num)
353     tex.sprint(catlatex,__tag_get_tag_from (num))
354 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.

```

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

```

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

```

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

```

370 function ltx.__tag.func.mc_num_of_kids (mcnum)
371     local num = 0
372     if ltx.__tag.mc[mcnum] and ltx.__tag.mc[mcnum]["kids"] then

```

```

373     num = #ltx.__tag.mc[mcnum]["kids"]
374 end
375 __tag_log ("INFO MC-KID-NUMBERS: " .. mcnum .. "has " .. num .. "KIDS",4)
376 return num
377 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 This insert the emc node. We support also dvips and dvipdfmx backend
__tag_insert_emc_node

```

378 local __tag_backend_create_emc_node
379 if tex.outputmode == 0 then
380   if token.get_macro("c_sys_backend_str") == "dvipdfmx" then
381     function __tag_backend_create_emc_node ()
382       local emcnode = nodenew("whatsit","special")
383       emcnode.data = "pdf:code EMC"
384       return emcnode
385     end
386   else -- assume a dvips variant
387     function __tag_backend_create_emc_node ()
388       local emcnode = nodenew("whatsit","special")
389       emcnode.data = "ps:SDict begin mark /EMC pdfmark end"
390       return emcnode
391     end
392   end
393   else -- pdf mode
394     function __tag_backend_create_emc_node ()
395       local emcnode = nodenew("whatsit","pdf_literal")
396       emcnode.data = "EMC"
397       emcnode.mode=1
398       return emcnode
399     end
400   end
401 end
402 local function __tag_insert_emc_node (head,current)
403   local emcnode= __tag_backend_create_emc_node()
404   head = node.insert_before(head,current,emcnode)
405   return head
406 end

```

(End of definition for __tag_backend_create_emc_node and __tag_insert_emc_node.)

__tag_backend_create_bmc_node This inserts a simple bmc node
__tag_insert_bmc_node

```

407 local __tag_backend_create_bmc_node
408 if tex.outputmode == 0 then
409   if token.get_macro("c_sys_backend_str") == "dvipdfmx" then
410     function __tag_backend_create_bmc_node (tag)
411       local bmcnode = nodenew("whatsit","special")
412       bmcnode.data = "pdf:code /"..tag.." BMC"
413       return bmcnode
414     end

```



```

415 else -- assume a dvips variant
416   function __tag_backend_create_bmc_node (tag)
417     local bmcnode = nodenew("whatsit","special")
418     bmcnode.data = "ps:SDict begin mark/"..tag.." /BMC pdfmark end"
419     return bmcnode
420   end
421 end
422 else -- pdf mode
423   function __tag_backend_create_bmc_node (tag)
424     local bmcnode = nodenew("whatsit","pdf_literal")
425     bmcnode.data = "/"..tag.." BMC"
426     bmcnode.mode=1
427     return bmcnode
428   end
429 end
430
431 local function __tag_insert_bmc_node (head,current,tag)
432   local bmcnode = __tag_backend_create_bmc_node (tag)
433   head = node.insert_before(head,current,bmcnode)
434   return head
435 end

```

(End of definition for __tag_backend_create_bmc_node and __tag_insert_bmc_node.)

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

```

436 local __tag_backend_create_bdc_node
437
438 if tex.outputmode == 0 then
439   if token.get_macro("c_sys_backend_str") == "dvipdfmx" then
440     function __tag_backend_create_bdc_node (tag,dict)
441       local bdcnode = nodenew("whatsit","special")
442       bdcnode.data = "pdf:code /"..tag.."<<..dict..>> BDC"
443       return bdcnode
444     end
445   else -- assume a dvips variant
446     function __tag_backend_create_bdc_node (tag,dict)
447       local bdcnode = nodenew("whatsit","special")
448       bdcnode.data = "ps:SDict begin mark/"..tag.."<<..dict..>> /BDC pdfmark end"
449       return bdcnode
450     end
451   end
452 else -- pdf mode
453   function __tag_backend_create_bdc_node (tag,dict)
454     local bdcnode = nodenew("whatsit","pdf_literal")
455     bdcnode.data = "/"..tag.."<<..dict..>> BDC"
456     bdcnode.mode=1
457     return bdcnode
458   end
459 end
460
461 local function __tag_insert_bdc_node (head,current,tag,dict)
462   bdcnode= __tag_backend_create_bdc_node (tag,dict)

```

```

463 head = node.insert_before(head,current,bdcnode)
464 return head
465 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.

```

466 local function __tag_pdf_object_ref (name,index)
467     local object
468     if ltx.pdf.object_id then
469         object = ltx.pdf.object_id (name,index) ..' 0 R'
470     else
471         local tokenname = 'c__pdf_object_'..name..'/'..index..'_int'
472         object = token.create(tokenname).mode ..' 0 R'
473     end
474     return object
475 end
476 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.

```

477 local function __tag_show_spacemark (head,current,color,height)
478     local markcolor = color or "1 0 0"
479     local markheight = height or 10
480     local pdfstring
481     if tex.outputmode == 0 then
482         -- ignore dvi mode for now
483     else
484         pdfstring = node.new("whatsit","pdf_literal")
485         pdfstring.data =
486             string.format("q " ..markcolor.." RG " ..markcolor.." rg 0.4 w 0 %g m 0 %g l S Q",-
487                 3,markheight)
488         head = node.insert_after(head,current,pdfstring)
489     end
490 end

```

(End of definition for `__tag_show_spacemark`.)

`__tag_fakespace` This is used to define a lua version of `\pdf-fakespace`

`ltx.__tag.func.fakespace`

```

491 local function __tag_fakespace()
492     tex.setattribute(iwspaceattributeid,1)
493     tex.setattribute(iwfontattributeid,font.current())
494 end
495 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.

```
496 --[[ a function to mark up places where real space chars should be inserted
497      it only sets an attribute.
498 --]]
499
500 local function __tag_mark_spaces (head)
501     local inside_math = false
502     for n in nodetraverse(head) do
503         local id = n.id
504         if id == GLYPH then
505             local glyph = n
506             default_currfontid = glyph.font
507             if glyph.next and (glyph.next.id == GLUE)
508                and not inside_math and (glyph.next.width > 0)
509             then
510                 nodesetattribute(glyph.next,iwspaceattributeid,1)
511                 nodesetattribute(glyph.next,iwfontattributeid,glyph.font)
512             -- for debugging
513                 if ltx.__tag.trace.showspace then
514                     __tag_show_spacemark (head,glyph)
515                 end
516             elseif glyph.next and (glyph.next.id==KERN) and not inside_math then
517                 local kern = glyph.next
518                 if kern.next and (kern.next.id== GLUE) and (kern.next.width > 0)
519                 then
520                     nodesetattribute(kern.next,iwspaceattributeid,1)
521                     nodesetattribute(kern.next,iwfontattributeid,glyph.font)
522                 end
523             end
524             -- look also back
525             if glyph.prev and (glyph.prev.id == GLUE)
526                and not inside_math
527                and (glyph.prev.width > 0)
528                and not nodehasattribute(glyph.prev,iwspaceattributeid)
529             then
530                 nodesetattribute(glyph.prev,iwspaceattributeid,1)
531                 nodesetattribute(glyph.prev,iwfontattributeid,glyph.font)
532             -- for debugging
533                 if ltx.__tag.trace.showspace then
534                     __tag_show_spacemark (head,glyph)
535                 end
536             end
537             elseif id == PENALTY then
538                 local glyph = n
539                 -- __tag_log ("PENALTY ".. n.subtype.."VALUE"..n.penalty,3)
540                 if glyph.next and (glyph.next.id == GLUE)
541                    and not inside_math and (glyph.next.width > 0) and n.subtype==0
542                 then
543                     nodesetattribute(glyph.next,iwspaceattributeid,1)
```

```

544         -- changed 2024-01-18, issue #72
545         nodesetattribute(glyph.next,iwfontattributeid,default_currfontid)
546     -- for debugging
547     if ltx.__tag.trace.showspaces then
548         __tag_show_spacemark (head,glyph)
549     end
550 end
551 elseif id == MATH then
552     inside_math = (n.subtype == 0)
553 end
554 end
555 return head
556 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

```

```

557 local function __tag_activate_mark_space ()
558     if not luatexbase.in_callback ("pre_linebreak_filter","markspaces") then
559         luatexbase.add_to_callback("pre_linebreak_filter",__tag_mark_spaces,"markspaces")
560         luatexbase.add_to_callback("hpack_filter",__tag_mark_spaces,"markspaces")
561     end
562 end
563
564 ltx.__tag.func.markspaceon=__tag_activate_mark_space
565
566 local function __tag_deactivate_mark_space ()
567     if luatexbase.in_callback ("pre_linebreak_filter","markspaces") then
568         luatexbase.remove_from_callback("pre_linebreak_filter","markspaces")
569         luatexbase.remove_from_callback("hpack_filter","markspaces")
570     end
571 end
572
573 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.

```

574 local default_space_char = nodenew(GLYPH)
575 local default_fontid      = fontid("TU/lmr/m/n/10")
576 local default_currfontid = fontid("TU/lmr/m/n/10")
577 default_space_char.char   = 32
578 default_space_char.font   = default_fontid

```

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

```

579 local function __tag_font_has_space (fontid)
580     t= fonts.hashes.identifiers[fontid]
581     if luaotfload.aux.slot_of_name(fontid,"space")
582         or t.characters and t.characters[32] and t.characters[32]["unicode"]==32
583     then
584         return true

```

```

585 else
586     return false
587 end
588 end

```

`--tag_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.

`ltx.__tag.func.space_chars_shipout`

```

589 local function __tag_space_chars_shipout (box)
590     local head = box.head
591     if head then
592         for n in node.traverse(head) do
593             local spaceattr = -1
594             if not nodehasattribute(n,iwspaceOffattributeid) then
595                 spaceattr = nodegetattribute(n,iwspaceattributeid) or -1
596             end
597             if n.id == HLIST then -- enter the hlist
598                 __tag_space_chars_shipout (n)
599             elseif n.id == VLIST then -- enter the vlist
600                 __tag_space_chars_shipout (n)
601             elseif n.id == GLUE then
602                 if ltx.__tag.trace.showspace and spaceattr==1 then
603                     __tag_show_spacemark (head,n,"0 1 0")
604                 end
605                 if spaceattr==1 then
606                     local space
607                     local space_char = node.copy(default_space_char)
608                     local curfont = nodegetattribute(n,iwfontattributeid)
609                     __tag_log ("INFO SPACE-FUNCTION-FONT: ".. tostring(curfont),3)
610                     if curfont and
611                         -- luaotfload.aux.slot_of_name(curfont,"space")
612                         __tag_font_has_space (curfont)
613                     then
614                         space_char.font=curfont
615                     end
616                     head, space = node.insert_before(head, n, space_char) --
617                     n.width = n.width - space.width
618                     space.attr = n.attr
619                 end
620             end
621         end
622         box.head = head
623     end
624 end
625
626 function ltx.__tag.func.space_chars_shipout (box)
627     __tag_space_chars_shipout (box)
628 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.

```

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

```

(End of definition for ltx.__tag.func.mc_insert_kids.)

`ltx.__tag.func.store_struct_mcabs` This function is used in the tagpdf-mc-luacode. It store the absolute count of the mc into the current structure. This must be done ordered.

```

661 function ltx.__tag.func.store_struct_mcabs (structnum,mcnum)
662   ltx.__tag.struct[structnum]=ltx.__tag.struct[structnum] or { }
663   ltx.__tag.struct[structnum]["mc"]=ltx.__tag.struct[structnum]["mc"] or { }
664   -- a structure can contain more than on mc chunk, the content should be ordered
665   tableinsert(ltx.__tag.struct[structnum]["mc"],mcnum)
666   __tag_log("INFO TEX-MC-INTO-STRUCT: " ..
667         mcnum.." inserted in struct "..structnum,3)
668   -- but every mc can only be in one structure
669   ltx.__tag.mc[mcnum]= ltx.__tag.mc[mcnum] or { }

```

```

670 ltx.__tag.mc[mcnum]["parent"] = structnum
671 end
672

```

(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.

```

673 -- pay attention: lua counts arrays from 1, tex pages from one
674 -- mcid and arrays in pdf count from 0.
675 function ltx.__tag.func.store_mc_in_page (mcnum,mcpagecnt,page)
676   ltx.__tag.page[page] = ltx.__tag.page[page] or {}
677   ltx.__tag.page[page][mcpagecnt] = mcnum
678   __tag_log("INFO TAG-MC-INTO-PAGE: page " .. page ..
679             ": inserting MCID " .. mcpagecnt .. " => " .. mcnum,3)
680 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)

```

681 local function __tag_update_mc_attributes (head,mcnum,type)
682   for n in node.traverse(head) do
683     node.set_attribute(n,mccntattributeid,mcnum)
684     node.set_attribute(n,mctypeattributeid,type)
685     if n.id == HLIST or n.id == VLIST then
686       __tag_update_mc_attributes (n.list,mcnum,type)
687     end
688   end
689   return head
690 end
691 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.

```

692 --[[
693   Now follows the core function
694   It wades through the shipout box and checks the attributes
695   ARGUMENTS
696   box: is a box,
697   mcpagecnt: num, the current page cnt of mc (should start at -1 in shipout box), needed for r
698   mcntprev: num, the attribute cnt of the previous node/whatever - if different we have a chu
699   mcopen: num, records if some bdc/emc is open
700   These arguments are only needed for log messages, if not present are replaces by fix strings
701   name: string to describe the box
702   mctypeprev: num, the type attribute of the previous node/whatever
703
704   there are lots of logging messages currently. Should be cleaned up in due course.
705   One should also find ways to make the function shorter.

```

```

706 --]]
707
708 function ltx.__tag.func.mark_page_elements (box,mcpagecnt,mccntprev,mcopen,name,mctypeprev)
709     local name = name or ("SOMEBBOX")
710     local mctypeprev = mctypeprev or -1
711     local abspage = status.total_pages + 1 -- the real counter is increased
712                                           -- inside the box so one off
713                                           -- if the callback is not used. (???)
714     __tag_log ("INFO TAG-ABSPAGE: " .. abspage,3)
715     __tag_log ("INFO TAG-ARGS: pagecnt".. mcpagecnt..
716               " prev "..mccntprev ..
717               " type prev "..mctypeprev,4)
718     __tag_log ("INFO TAG-TRAVERSING-BOX: ".. tostring(name)..
719               " TYPE ".. node.type(node.getid(box)),3)
720     local head = box.head -- ShipoutBox is a vlist?
721     if head then
722         mccnthead, mctypehead,taghead = __tag_get_mc_cnt_type_tag (head)
723         __tag_log ("INFO TAG-HEAD: " ..
724                   node.type(node.getid(head))..
725                   " MC"..tostring(mccnthead)..
726                   " => TAG " .. tostring(mctypehead)..
727                   " => ".. tostring(taghead),3)
728     else
729         __tag_log ("INFO TAG-NO-HEAD: head is "..
730                   tostring(head),3)
731     end
732     for n in node.traverse(head) do
733         local mccnt, mctype, tag = __tag_get_mc_cnt_type_tag (n)
734         local spaceattr = nodegetattribute(n,iwspaceattributeid) or -1
735         __tag_log ("INFO TAG-NODE: "..
736                   node.type(node.getid(n))..
737                   " MC".. tostring(mccnt)..
738                   " => TAG ".. tostring(mctype)..
739                   " => " .. tostring(tag),3)
740         if n.id == HLIST
741         then -- enter the hlist
742             mcopen,mcpagecnt,mccntprev,mctypeprev=
743                 ltx.__tag.func.mark_page_elements (n,mcpagecnt,mccntprev,mcopen,"INTERNAL HLIST",mctypeprev)
744         elseif n.id == VLIST then -- enter the vlist
745             mcopen,mcpagecnt,mccntprev,mctypeprev=
746                 ltx.__tag.func.mark_page_elements (n,mcpagecnt,mccntprev,mcopen,"INTERNAL VLIST",mctypeprev)
747         elseif n.id == GLUE and not n.leader then -- at glue real space chars are inserted, but this
748                                                     -- been done if the previous shipout wandering, so here it is
749         elseif n.id == LOCAL_PAR then -- local_par is ignored
750         elseif n.id == PENALTY then -- penalty is ignored
751         elseif n.id == KERN then -- kern is ignored
752             __tag_log ("INFO TAG-KERN-SUBTYPE: "..
753                       node.type(node.getid(n)).." "..n.subtype,4)
754         else
755             -- math is currently only logged.
756             -- we could mark the whole as math
757             -- for inner processing the mlist_to_hlist callback is probably needed.
758             if n.id == MATH then
759                 __tag_log("INFO TAG-MATH-SUBTYPE: "..

```



```

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

```

```

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

```

(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.

```

849 function ltx.__tag.func.mark_shipout (box)
850     mcopen = ltx.__tag.func.mark_page_elements (box,-1,-100,0,"Shipout",-1)
851     if mcopen~=0 then -- there is a chunk open, close it (hope there is only one ...
852         local emcnode = __tag_backend_create_emc_node ()
853         local list = box.list
854         if list then
855             list = node.insert_after (list,node.tail(list),emcnode)
856             mcopen = mcopen - 1
857             __tag_log ("INFO SHIPOUT-INSERT-LAST-EMC: MCOPEN " .. mcopen,3)
858         else
859             __tag_log ("WARN SHIPOUT-UPS: this shouldn't happen",0)
860         end
861     end
862

```

```

861   if mcopen ~=0 then
862     __tag_log ("WARN SHIPOUT-MC-OPEN: " .. mcopen,1)
863   end
864 end
865 end

```

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

7 Parenttree

ltx.__tag.func.fill_parent_tree_line 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.

```

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

```

```

906     else
907         __tag_log ("INFO PARENTTREE-NO-DATA: page "..page,3)
908         numsentry = pdfpage.." ["
909     end
910     return numsentry
911 end
912
913 function ltx.__tag.func.output_parenttree (abspage)
914     for i=1,abspage do
915         line = ltx.__tag.func.fill_parent_tree_line (i) .. "^^J"
916         tex.sprint(catlatex,line)
917     end
918 end

```

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

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

```

919 do
920     local properties = node.get_properties_table()
921     local is_soft_hyphen_prop = 'tagpdf.rewrite-softhyphen.is_soft_hyphen'
922     local hyphen_char = 0x2D
923     local soft_hyphen_char = 0xAD

```

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

```

924     local softhyphen_fonts = setmetatable({}, {__index = function(t, fid)
925         local fdir = identifiers[fid]
926         local format = fdir and fdir.format
927         local result = (format == 'opentype' or format == 'truetype')
928         local characters = fdir and fdir.characters
929         result = result and (characters and characters[soft_hyphen_char]) ~= nil
930         t[fid] = result
931         return result
932     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.

```

933     local function process_softhyphen_pre(head, _context, _dir)
934         if softhyphenbool.mode ~= truebool.mode then return true end
935         for disc, sub in node.traverse_id(DISC, head) do
936             if sub == explicit_disc or sub == regular_disc then
937                 for n, _ch, _f in node.traverse_char(disc.pre) do
938                     local props = properties[n]
939                     if not props then
940                         props = {}
941                         properties[n] = props
942                     end
943                     props[is_soft_hyphen_prop] = true
944                 end
945             end
946         end

```

```

947     return true
948 end
949

```

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

```

950 local function process_softhyphen_post(head, _context, _dir)
951   if softhyphenbool.mode ~= truebool.mode then return true end
952   for disc, sub in node.traverse_id(DISC, head) do
953     for n, ch, fid in node.traverse_glyph(disc.pre) do
954       local props = properties[n]
955       if softhyphen_fonts[fid] and ch == hyphen_char and props and props[is_soft_hyphen_prop]
956         n.char = soft_hyphen_char
957         props.glyph_info = nil
958       end
959     end
960   end
961   return true
962 end
963
964 luatexbase.add_to_callback('pre_shaping_filter', process_softhyphen_pre, 'tagpdf.rewrite-
softhyphen')
965 luatexbase.add_to_callback('post_shaping_filter', process_softhyphen_post, 'tagpdf.rewrite-
softhyphen')
966 end

```

(End of definition for process_softhyphen_pre process_softhyphen_post. This function is documented on page ??.)

8 parent-child rules

role_get_parent_child_rule

`ltx.__tag.func.role_get_parent_child_rule`

```

967 local function role_get_parent_child_rule (parent,child)
968   local state=
969   ltx.__tag.role.matrix[ltx.__tag.role.index[parent]]
970   and ltx.__tag.role.matrix[ltx.__tag.role.index[parent]][ltx.__tag.role.index[child]] or 0
971   return state
972 end
973 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 ??.)

check_update_stashed

check_parent_child_rules

`ltx.__tag.func.check_parent_child_rules`

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.

```

974 function check_update_stashed (struct,loglevel,loop)
975   loop = (loop or 0) + 1
976   if loop > 10 then
977     __tag_log ('Warning: Too deeply nested stashed structures',0)
978     return
979   end
980   __tag_log ('updating parentrole for stashed structure '..struct,loglevel)
981   local parent = ltx.__tag.tables['g__tag_struct_ '..struct..'__prop']['parentnum']
982   if parent then
983     local ptag =
984       string.match(ltx.__tag.tables['g__tag_struct_ '..parent..'__prop']['parentrole'], "{(.-)
985     )}{(.-)}")
986     if ptag == 'STASHED' then
987       -- look at the parent and update it first
988       check_update_stashed (parent,loglevel,loop)
989     end
990     -- now copy the parent role from the parent
991     ltx.__tag.tables['g__tag_struct_ '..struct..'__prop']['parentrole']
992     =
993     ltx.__tag.tables['g__tag_struct_ '..parent..'__prop']['parentrole']
994     __tag_log
995     ('new parentrole: ' .. ltx.__tag.tables['g__tag_struct_ '..struct..'__prop']['parentrole'], 10)
996   else
997     __tag_log ('Warning: structure '..struct..' has no parent.',0)
998   end
999 end
1000 function check_parent_child_rules (loglevel)
1001   texio.write_nl('\n')
1002   __tag_log ('checking parent-child rules ...' ,0)
1003   for i=2,ltx.tag.get_struct_counter() do
1004     local t,tNS=
1005       string.match(ltx.__tag.tables['g__tag_struct_ '..i..'__prop']['tag'], "{(.-)
1006     )}{(.-)}")
1007     local r,rNS=
1008       string.match(ltx.__tag.tables['g__tag_struct_ '..i..'__prop']['rolemap'], "{(.-)
1009     )}{(.-)}")
1010     local p,pNS=
1011       string.match(ltx.__tag.tables['g__tag_struct_ '..i..'__prop']['parentrole'], "{(.-)
1012     )}{(.-)}")
1013     local parent=ltx.__tag.tables['g__tag_struct_ '..i..'__prop']['parentnum']
1014     if parent then
1015       __tag_log (i..'': '.. t..'': '..tNS,loglevel)
1016       __tag_log (i..'': '.. r..'': '..rNS,loglevel)
1017       __tag_log (i..'': '.. p..'': '..pNS,loglevel)
1018       __tag_log ('parent of ' ..i..'': '.. parent,loglevel )
1019       if p == 'STASHED' then
1020         check_update_stashed (i,loglevel,0)
1021       end
1022       p,pNS=
1023         string.match(ltx.__tag.tables['g__tag_struct_ '..i..'__prop']['parentrole'], "{(.-)
1024       )}{(.-)}")
1025     end
1026     local pt,ptNS=
1027       string.match(ltx.__tag.tables['g__tag_struct_ '..parent..'__prop']['tag'], "{(.-)

```

```

    )}{(.-)}")
1023     local pr,prNS=
1024     string.match(ltx.__tag.tables['g__tag_struct_'..parent..'__prop']['rolemap'], "{(.-)
    )}{(.-)}")
1025     local pp,ppNS=
1026     string.match(ltx.__tag.tables['g__tag_struct_'..parent..'__prop']['parentrole'], "{(.-)
    )}{(.-)}")
1027     if pp == 'STASHED' then
1028         check_update_stashed (parent,loglevel,0)
1029         pp,ppNS=
1030         string.match(ltx.__tag.tables['g__tag_struct_'..parent..'__prop']['parentrole'], "{(.-)
    )}{(.-)}")
1031     end
1032     __tag_log (parent..'': '.. pt..'': '..ptNS,loglevel)
1033     __tag_log (parent..'': '.. pr..'': '..prNS,loglevel)
1034     __tag_log (parent..'': '.. pp..'': '..ppNS,loglevel)
1035     -- now check the rule.
1036     -- at first rolemap of child against rolemap of parent.
1037     local state=ltx.__tag.func.role_get_parent_child_rule (pr,r)
1038     __tag_log ('rule of '..pr.."-">"..r.." is '..state,loglevel)
1039     -- if the state is 7 we check against parentrole of the parent
1040     if state == 7 then
1041         state=ltx.__tag.func.role_get_parent_child_rule (pp,r)
1042         __tag_log ('Parent-Child relation '..pp.."-">"..r.." is '..state,loglevel)
1043     end
1044     if state == 0 then
1045         __tag_log
1046         ('Warning: Parent-Child relation '
1047         '..ptNS..'': '..pt..' -> '..tNS..'': '..t..' is unknown',0)
1048         __tag_log
1049         ('Structure ' ..parent..' -> '..i,0)
1050     end
1051     if state == -1 then
1052         __tag_log
1053         ('Warning: Parent-Child relation '
1054         '..ptNS..'': '..pt..' -> '..tNS..'': '..t..' is not allowed',0)
1055         __tag_log
1056         ('Structure ' ..parent..' -> '..i,0)
1057     end
1058     -- check also for MC
1059     state =ltx.__tag.func.role_get_parent_child_rule ( r ,'MC')
1060     local curtag=r
1061     if state == 7 then
1062         state =ltx.__tag.func.role_get_parent_child_rule ( p ,'MC')
1063         local curtag=p
1064     end
1065     if state == -1 then
1066         if ltx.__tag.struct[i] and NEXT(ltx.__tag.struct[i]) then
1067             __tag_log
1068             ('Warning: Real content (MC) is not allowed in ' ..curtag,0)
1069         end
1070     end
1071     __tag_log('=====',loglevel)
1072 end

```

```

1073     end -- end for
1074 end
1075
1076 ltx.__tag.func.check_parent_child_rules=check_parent_child_rules
1077

```

(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.

```

1078 if luatexbase.callbacktypes['linksplit'] then
1079     luatexbase.add_to_callback('linksplit', function(start_link, position)
1080         if start_link == nil then return end
1081         local structnum =
1082             node.get_attribute(start_link, luatexbase.attributes.g__tag_structnum_attr)
1083         if structnum and structnum > -1 then
1084             local s = ltx.__tag.tables['g__tag_struct_'..structnum..'__prop']['rolemap']
1085             if s and (string.find(s, 'Link') or string.find(s, 'Reference')) then
1086                 local struct_insert_annot_shipout = token.create '__tag_struct_insert_annot_shipout:nn'
1087                 local parentnum = tex.count['c@g__tag_parenttree_obj_int']
1088                 start_link.link_attr =
1089                     start_link.link_attr ..
1090                     ' /LTEX_position /' .. position ..
1091                     '/StructParent ' .. parentnum
1092                 tex.sprint(catlatex, struct_insert_annot_shipout, '{'..
1093                     structnum..' }{'..
1094                     start_link.objnum..' 0 R }{'..
1095                     parentnum..' }')
1096                 -- the counter must be set explicitly as struct_insert_annot_shipout doesn't do it!
1097                 tex.setcount('global', 'c@g__tag_parenttree_obj_int', parentnum + 1)
1098                 __tag_log(position .. " link part has object id " .. start_link.objnum .. " and stru
1099             else
1100                 __tag_log('Warning: Link not in Link or Reference structure element', 0)
1101                 __tag_log('OBJR not created', 0)
1102                 __tag_log('', 0)
1103             end
1104         end
1105     end, 'tagpdf')
1106 end
1107 </lua>

```

The tagpdf-roles module
 Tags, roles and namespace code
 Part of the tagpdf package
 Ulrike Fischer
 Version 0.99u, released 2025-07-16

Part X

`add-new-tag` (setup-key)
`tag` (rolemap-key)
`namespace` (rolemap-key)
`role` (rolemap-key)
`role-namespace` (rolemap-key)

The `add-new-tag` key can be used in `\tagpdfsetup` to declare and rolemap new tags. It takes as value a key-value list or a simple `new-tag/old-tag`.

The key-value list knows the following keys:

tag This is the name of the new tag as it should then be used in `\tagstructbegin`.

namespace This is the namespace of the new tag. The value should be a shorthand of a namespace. The allowed values are currently `pdf`, `pdf2`, `mathml`, `latex`, `latex-book` and `user`. The default value (and recommended value for a new tag) is `user`. The public name of the user namespace is `tag/NS/user`. This can be used to reference the namespace e.g. in attributes.

role This is the tag the tag should be mapped too. In a PDF 1.7 or earlier this is normally a tag from the `pdf` set, in PDF 2.0 from the `pdf`, `pdf2` and `mathml` set. It can also be a user tag. The tag must be declared before, as the code retrieves the class of the new tag from it. The PDF format allows mapping to be done transitively. But tagpdf can't/won't check such unusual role mapping.

role-namespace If the role is a known tag the default value is the default namespace of this tag. With this key a specific namespace can be forced.

Namespaces are mostly a PDF 2.0 property, but it doesn't harm to set them also in a PDF 1.7 or earlier.

`\tag_check_child: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} {2025-07-16} {0.99u}
4 {part of tagpdf - code related to roles and structure names}
5 </header>
```

1 Code related to roles and structure names

6 <*package>

1.1 Variables

Tags are used in structures (`\tagstructbegin`) and mc-chunks (`\tagmcbegin`).

They have a name (a string), in lua a number (for the lua attribute), and in PDF 2.0 belong to one or more name spaces, with one being the default name space.

Tags of structures are classified, e.g. as grouping, inline or block level structure (and a few special classes like lists and tables), and must follow containments rules depending on their classification (for example a inline structure can not contain a block level structure). New tags inherit their classification from their rolemapping to the standard namespaces (**pdf** and/or **pdf2**). We store this classification as it will probably be needed for tests but currently the data is not much used. The classification for math (and the containment rules) is unclear currently and so not set.

The attribute number is only relevant in lua and only for the MC chunks (so tags with the same name from different names spaces can have the same number), and so only stored if luatex is detected.

Due to the namespaces the storing and processing of tags and there data are different in various places for PDF 2.0 and PDF <2.0, which makes things a bit difficult and leads to some duplications. Perhaps at some time there should be a clear split.

This are the main variables used by the code:

\g__tag_role_tags_NS_prop This is the core list of tag names. It uses tags as keys and the shorthand (e.g. pdf2, or mathml) of the default name space as value.

In pdf 2.0 the value is needed in the structure dictionaries.

\g__tag_role_tags_class_prop This contains for each tag a classification type. It is used in pdf <2.0.

\g__tag_role_NS_prop This contains the names spaces. The values are the object references. They are used in pdf 2.0.

\g__tag_role_rolemap_prop This contains for each tag the role to a standard tag. It is used in pdf<2.0 for tag checking and to fill at the end the RoleMap dictionary.

g_@@_role/RoleMap_dict This dictionary contains the standard rolemaps. It is relevant only for pdf <2.0.

\g__tag_role_NS_<ns>_prop This prop contains the tags of a name space and their role. The props are also use for remapping. As value they contain two brace groups: tag and namespace. In pdf <2.0 the namespace is empty.

\g__tag_role_NS_<ns>_class_prop This prop contains the tags of a name space and their type. The value is only needed for pdf 2.0.

\g__tag_role_index_prop This prop contains the standard tags (pdf in pdf<2.0, pdf, pdf2 + mathml in pdf 2.0) as keys, the values are a two-digit number. These numbers are used to get the containment rule of two tags from the intarray.

\g__tag_role_tags_NS_prop This is the core list of tag names. It uses tags as keys and the shorthand (e.g. pdf2, or mathml) of the default name space as value. We store the default name space also in pdf <2.0, even if not needed: it doesn't harm and simplifies the code. There is no need to access this from lua, so we use the standard prop commands.

7 **\prop_new:N** **\g__tag_role_tags_NS_prop**

(End of definition for **\g__tag_role_tags_NS_prop**.)

\g__tag_role_tags_class_prop With pdf 2.0 we store the class in the NS 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/dft>

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` %12 `\tl_new:N \l__tag_role_tag_tmpa_tl`

`\l__tag_role_role_tmpa_tl`13 `\tl_new:N \l__tag_role_tag_namespace_tmpa_tl`

`\l__tag_role_role_namespace_tmpa_tl`14 `\tl_new:N \l__tag_role_tag_namespace_tmpb_tl`

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

```
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}{#2}{#3}
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_role/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}{#2}
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}{#2}
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}{#2}
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}{#2}
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_rolemapping_prop {#2}\l__tag_tmpa_tl
145     {
146       \__tag_prop_gput:Nno \g__tag_role_rolemapping_prop {#1}{\l__tag_tmpa_tl}
147     }
148     {
149       \__tag_prop_gput:Nne \g__tag_role_rolemapping_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 #4
```

```

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 }
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:Nc #3 {\exp_last_unbraced:No\use_i:nn {\l__tag_get_tmpc_tl}}
239       \tl_set:Nc #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         }
278         \__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
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}
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` `\c__tag_role_rules_num_prop` These two properties map the rule strings to numbers and back. There are in tagpdf-data.dtx near the csv files for easier maintenance.

(End 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, #3
394     {
395         \prop_get:NnTF \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, #3
418     {
419         \prop_get:NnTF \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. 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:Nc#3
509     {
510         \lua_now:e{tex.print(\int_use:N\c_document_cctab,ltx.__tag.func.role_get_parent_child
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:Nc#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 t1 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_tmpp_tl
610 {
611   \tl_set:Nn \l__tag_tmpp_tl {#3}
612 }
613 {
614   \prop_get:NnNF \g__tag_role_rolemap_prop {#3}\l__tag_tmpp_tl
615   {
616     \tl_set:Nn \l__tag_tmpp_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_tmpp_tl }
622 {
623   \__tag_role_get_parent_child_rule:ooN
624   { \l__tag_tmpa_tl }
625   { \l__tag_tmpp_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, tl
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:nnTF

```

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 185.)

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.

```

tag (rolemap-key)
tag-namespace (rolemap-key)
role (rolemap-key)752 \keys_define:nn { __tag / tag-role }
role-namespace (rolemap-key)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}\l__
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 185.)

The tagpdf-space module

Code related to real space chars

Part of the tagpdf package

Ulrike Fischer

Version 0.99u, released 2025-07-16

Part XI

`activate/space` (setup-key)
`interwordspace` (deprecated)

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.

`show-spaces` (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} {2025-07-16} {0.99u}
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.

`activate/spaces` (setup-key)

`interwordspace` (deprecated)

`show-spaces` (deprecated)

```

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     \keys_define:nn { __tag / setup }
32     {
33       activate/spaces/true .code:n = { \AddToHook{shipout/firstpage}[tagpdf/space]{\pdfin
34       activate/spaces/false .code:n = { \RemoveFromHook{shipout/firstpage}[tagpdf/space] }
35       activate/spaces .default:n = true,
36     }
37   }
38   {
39     \keys_define:nn { __tag / setup }
40     {
41       activate/spaces .choices:nn = { true, false }
42       { \msg_warning:nnn {tag}{sys-no-interwordspace}{dvi} },
43       activate/spaces .default:n = true,
44     }
45   }
46 }
47
48
49 \sys_if_engine_luatex:T
50 {
51   \keys_define:nn { __tag / setup }
52   {
53     activate/spaces .choice:,
54     activate/spaces/true .code:n =
55     {
56       \bool_gset_true:N \g__tag_active_space_bool
57       \lua_now:e{!tx.__tag.func.markspaceon()}
58     },
59     activate/spaces/false .code:n =
60     {
61       \bool_gset_false:N \g__tag_active_space_bool
62       \lua_now:e{!tx.__tag.func.markspaceoff()}
63     },
64     activate/spaces .default:n = true,
65     debug/show/spaces .code:n =
66     { \lua_now:e{!tx.__tag.trace.showspace=true} },
67     debug/show/spacesOff .code:n =
68     { \lua_now:e{!tx.__tag.trace.showspace=nil} },
69   }
70 }

```

(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.

```

71 \sys_if_engine_luatex:T
72 {
73   \cs_new_protected:Nn \__tag_fakespace:

```

```

74     {
75       \group_begin:
76       \lua_now:e{!tx.__tag.func.fakespace()}
77       \skip_horizontal:n{\c_zero_skip}
78       \group_end:
79     }
80   }

```

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 what-sits 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.

```

\tag_spacechar_on:
\tag_spacechar_off: 81 \cs_new_protected:Npn \tag_spacechar_off: {}
82 \cs_new_protected:Npn \tag_spacechar_on: {}
83
84 \sys_if_engine luatex:T
85 {
86   \cs_set_protected:Npn \tag_spacechar_off:
87   {
88     \lua_now:e
89     {
90       tex.setattribute
91       (
92         "global",
93         luatexbase.attributes.g__tag_interwordspaceOff_attr,
94         1
95       )
96     }
97   }
98   \cs_set_protected:Npn \tag_spacechar_on:
99   {
100     \lua_now:e
101     {
102       tex.setattribute
103       (
104         "global",
105         luatexbase.attributes.g__tag_interwordspaceOff_attr,
106         -2147483647
107       )
108     }
109   }
110 }
111 \sys_if_engine pdftex:T
112 {
113   \sys_if_output_pdf:T
114   {
115     \cs_set_protected:Npn \tag_spacechar_off:
116     {
117       \pdfinterwordspaceoff
118     }

```

```

119         \cs_set_protected:Npn \tag_spacechar_on:
120         {
121             \pdfinterwordspaceon
122         }
123     }
124 }

125 \</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.

Symbols

\\ 10, 23, 27, 28, 44, 49, 50, 51, 56,
58, 60, 67, 70, 72, 78, 80, 93, 96, 97,
106, 107, 113, 114, 166, 539, 602, 610
_ 429, 440

A

activate_ (setup-key) 41, 285
activate-all (deprecated) (key) 1
activate-mc (deprecated) (key) 1
activate-struct (deprecated) (key) 1
activate-tree (deprecated) (key) 1
activate/all (key) 1, 218
activate/mc (key) 1, 218
activate/socket_ (setup-key) 285
activate/softhyphen (key) 1, 252
activate/space_ (setup-key) 209
activate/spaces (key) 1
activate/spaces_ (setup-key) 6
activate/struct (key) 1, 218
activate/struct-dest (key) 1, 218
activate/tagunmarked (key) 1, 249
activate/tree (key) 1, 218
actualtext (key) 1, 740
actualtext_ (mc-key) 83, 238, 384
add-new-tag_ (deprecated) 752
add-new-tag_ (setup-key) 185
\\AddToHook 13, 16, 33, 44, 57, 303,
387, 520, 522, 523, 527, 531, 538, 578
AF (key) 1, 938
AFinline (key) 1, 938
AFinline-o (key) 1, 938
AFref (key) 1, 938
alt (key) 1, 740
alt_ (mc-key) 83, 238, 384
artifact_ (mc-key) 83, 238, 384
artifact-bool internal commands:
 __artifact-bool 182
artifact-type internal commands:
 __artifact-type 182
\\AssignSocketPlug 597, 604
\\AssignTaggingSocketPlug 712, 713
\\AtBeginDocument 692
attr-unknown 21, 84
attribute (key) 1, 1551
attribute-class (key) 1, 1517

B

benchmark commands:
 \\benchmark_tic: 619, 621
 \\benchmark_toc: 622
bool commands:
 \\bool_gset_eq:NN
 616, 631, 643, 661, 720, 734
 \\bool_gset_false:N
 61, 221, 372, 617, 644, 721
 \\bool_gset_true:N
 36, 56, 87, 175, 313, 971
 \\bool_if:NTF 9, 13, 18, 31,
 40, 40, 68, 75, 80, 85, 91, 96, 111,
 135, 146, 196, 203, 208, 228, 248,
 264, 265, 272, 281, 314, 323, 335,
 339, 351, 389, 426, 437, 451, 453,
 470, 478, 484, 503, 510, 611, 626,
 638, 656, 715, 729, 1188, 1220, 1251
 \\bool_if:nTF 489
 \\bool_lazy_all:nTF 240
 \\bool_lazy_and:nnTF
 274, 284, 305, 309,
 355, 373, 548, 619, 697, 747, 753, 974
 \\bool_new:N 7, 16, 20,
 21, 35, 73, 82, 83, 84, 85, 85, 86, 88,
 90, 92, 93, 94, 255, 323, 324, 607, 970
 \\bool_set_false:N
 17, 163, 164, 165, 176, 187, 188,
 189, 222, 344, 398, 567, 610, 637, 714
 \\bool_set_true:N 16, 89, 91, 173, 174,
 175, 198, 199, 200, 256, 346, 397, 566
box commands:
 \\box_dp:N 180, 184
 \\box_ht:N 170
 \\box_new:N 77, 78
 \\box_set_dp:Nn 178, 180
 \\box_set_eq:NN 193
 \\box_set_ht:Nn 177, 179
 \\box_use_drop:N 182, 186
\\boxmaxdepth 96, 181

C

c@g internal commands:
 \\c@g__tag_MCID_abs_int
 11, 15, 28, 37, 50, 57, 60, 68, 74,
 132, 138, 178, 242, 245, 288, 295, 346
 \\c@g__tag_parenttree_obj_int 155, 518

\c@g__tag_struct_abs_int	6, 18, 40, 58, 91, 114, 115, 118, 123, 126, 149, 166, 259, 385, 568, 745, 758, 803, 815, 829, 845, 860, 868, 922, 933, 952, 955, 960, 996, 998, 1003, 1015, 1017, 1022, 1113, 1124, 1125, 1126, 1127, 1128, 1129, 1131, 1133, 1139, 1144, 1151, 1154, 1164, 1172, 1176, 1191, 1204, 1214, 1227, 1230, 1245, 1246, 1248, 1259, 1544, 1547, 1595	111, 117, 120, 122, 126, 137, 142, 147, 153, 163, 167, 169, 171, 172, 179, 189, 193, 209, 210, 211, 212, 213, 213, 214, 233, 245, 246, 255, 259, 260, 263, 266, 266, 277, 281, 282, 285, 286, 292, 292, 295, 298, 302, 309, 318, 318, 322, 322, 323, 332, 333, 337, 341, 345, 349, 349, 351, 353, 357, 370, 376, 378, 385, 393, 397, 411, 413, 416, 417, 423, 434, 435, 446, 480, 503, 509, 516, 521, 523, 530, 541, 550, 558, 565, 572, 573, 574, 575, 576, 577, 580, 592, 596, 608, 615, 616, 617, 622, 635, 640, 651, 652, 710, 726, 873, 881, 894, 907, 940, 968, 1113, 1114, 1115, 1312, 1353, 1405, 1418, 1438, 1442, 1446, 1450, 1456, 1475, 1499
catalog-supplemental-file (key) ...	1093	
cctab commands:		
\c_document_cctab	49, 54, 75, 155, 510	
\chapter	196, 357, 375	
check commands:		
check_parent_child_rules	974	
check_update_stashed	974	
clist commands:		
\clist_const:Nn	79, 80	\cs_set:Nn 676, 677, 741, 742
\clist_if_empty:NTF	1556	\cs_set:Npn 47, 52, 89, 109
\clist_map_inline:nn	105, 437, 919	\cs_set_eq:NN 14, 20,
\clist_new:N	75	66, 80, 81, 82, 138, 139, 140, 141,
\clist_set:Nn	1521, 1555	142, 143, 144, 145, 146, 147, 180,
color commands:		181, 192, 206, 215, 216, 224, 225,
\color_select:n	429, 440	235, 236, 237, 238, 378, 379, 380,
cs commands:		381, 621, 622, 669, 670, 671, 672,
\cs:w	1427, 1431	678, 679, 683, 684, 685, 686, 743, 744
\cs_end:	1427, 1431	\cs_set_protected:Nn
\cs_generate_variant:Nn		.. 169, 216, 241, 359, 365, 1269, 1270
44, 79, 97, 98, 99, 100, 101,		\cs_set_protected:Npn 9, 15,
102, 103, 104, 105, 106, 107, 107,		16, 22, 29, 35, 38, 40, 48, 49, 52, 58,
114, 115, 116, 118, 126, 134, 149,		63, 65, 71, 73, 78, 83, 83, 86, 94, 98,
150, 151, 152, 153, 153, 154, 155,		102, 112, 115, 119, 143, 159, 168,
163, 164, 168, 181, 212, 226, 232,		182, 193, 220, 228, 240, 246, 267,
253, 265, 265, 276, 281, 297, 317,		283, 299, 301, 305, 359, 363, 367,
322, 332, 589, 651, 699, 717, 939,		371, 1117, 1118, 1306, 1314, 1355, 1407
967, 988, 1403, 1415, 1455, 1484, 1505		\cs_to_str:N
\cs_gset_eq:NN	414	.. 12, 18, 25, 32, 38, 43, 61, 62, 68, 69
\cs_if_exist:NTF	231, 580, 619	
\cs_if_exist_p:N	357, 360, 375, 378	
\cs_if_exist_use:NTF	418, 1409	
\cs_if_free:NTF	48	
\cs_new:Nn	83,	
104, 109, 131, 136, 293, 427, 428, 429		
\cs_new:Npn	9, 15, 23, 27, 92,	
117, 122, 156, 167, 229, 229, 233,		
237, 383, 546, 554, 560, 566, 1399, 1485		
\cs_new_eq:NN	37	
\cs_new_protected:Nn		
73, 127, 167, 296, 430, 434		
\cs_new_protected:Npn	13, 17, 20,	
22, 23, 30, 31, 36, 42, 43, 45, 60, 61,		
63, 65, 67, 78, 79, 80, 81, 81, 82, 84,		
85, 86, 93, 95, 102, 107, 107, 108,		

D

debug/log (key)	1, 236
debug/show (key)	235
debug/structures_ (show-key)	42, 254
debug/uncompress (key)	236
\DebugSocketsOn	44
\DeclareOption	37, 38
dim commands:	
\c_max_dim	169, 194
\c_zero_dim	177, 178, 179
\documentclass	16
\DocumentMetadata	15

E

E (key)	1, 740, 915
---------	-------------

<code>\endinput</code>	22	int commands:	
<code>\ERRORusetaggingsocket</code>	106, 121	<code>\int_abs:n</code>	172
<code>exclude-header-footer_␣</code> (deprecated)	664	<code>\int_case:nnTF</code>	99, 114, 347
exp commands:		<code>\int_compare:nNnTF</code>	
<code>\exp_args:Ne</code>	122, 548	22, 58, 70, 98, 116, 124, 125, 132,	
<code>\exp_args:NNe</code>	86, 89, 195, 215	142, 148, 157, 170, 173, 173, 261,	
<code>\exp_args:Nne</code> ..	79, 337, 341, 427, 461	311, 341, 360, 387, 390, 418, 424,	
<code>\exp_args:NNno</code>	774	444, 450, 511, 512, 518, 525, 528,	
<code>\exp_args:No</code>	291, 326	532, 544, 552, 553, 560, 562, 567,	
<code>\exp_last_unbraced:Ne</code> ...	99, 102, 109	575, 578, 582, 601, 637, 734, 748, 1155	
<code>\exp_last_unbraced:No</code> ..	135, 138,	<code>\int_compare:nTF</code>	
152, 154, 157, 159, 224, 225, 238,		180, 458, 1537, 1539, 1541, 1565, 1591	
239, 588, 592, 612, 615, 623, 626, 1295		<code>\int_compare_p:nNn</code>	758
<code>\exp_not:n</code>	185, 204	<code>\int_decr:N</code>	170, 195
		<code>\int_eval:n</code>	118, 138, 166, 197,
		396, 603, 611, 755, 760, 763, 960,	
		1003, 1022, 1125, 1126, 1127, 1128,	
		1129, 1245, 1246, 1248, 1259, 1547	
		<code>\int_gincr:N</code> ...	178, 242, 288, 295,
		343, 347, 351, 355, 361, 365, 369,	
		373, 518, 946, 1082, 1099, 1113, 1124	
		<code>\int_gset:Nn</code>	7, 82, 158
		<code>\int_if_zero:nTF</code>	
		170, 171, 195, 196, 599, 607
		<code>\int_incr:N</code>	93, 162, 186, 442
		<code>\int_new:N</code>	6, 76, 78,
		81, 95, 155, 158, 326, 327, 328, 329, 938	
		<code>\int_rand:n</code> ..	61, 62, 64, 66, 68, 70, 71
		<code>\int_set:Nn</code>	237, 240, 243, 244, 245
		<code>\int_step_inline:nn</code>	473, 479
		<code>\int_step_inline:nnn</code>	25, 91, 259
		<code>\int_step_inline:nnnn</code>	
		149, 174, 177, 200, 443, 449
		<code>\int_to_arabic:n</code>	172, 174
		<code>\int_to_Hex:n</code> ..	61, 62, 64, 66, 68, 70, 71
		<code>\int_use:N</code>	11, 15, 18, 28, 37,
		40, 49, 50, 54, 57, 58, 60, 68, 74, 75,	
		100, 115, 123, 130, 132, 155, 161,	
		178, 185, 204, 234, 241, 245, 259,	
		261, 346, 385, 429, 440, 510, 533,	
		549, 550, 558, 559, 568, 745, 803,	
		815, 829, 845, 860, 868, 922, 933,	
		949, 952, 955, 996, 998, 1015, 1017,	
		1086, 1089, 1103, 1107, 1133, 1139,	
		1144, 1151, 1154, 1176, 1191, 1204,	
		1214, 1227, 1230, 1485, 1544, 1595	
		<code>\int_zero:N</code>	90, 105, 430
		intarray commands:	
		<code>\intarray_gset:Nnn</code>	
		397, 406, 421, 421, 425
		<code>\intarray_item:Nn</code>	423, 426, 554
		<code>\intarray_new:Nn</code>	390, 413
		interwordspace _␣ (deprecated)	209, 6
F			
file commands:			
<code>\file_if_exist:nTF</code>	322		
<code>\file_input:n</code>	268		
firstkid (key)	1, 740		
flag commands:			
<code>\flag_clear:n</code>	239		
<code>\flag_height:n</code>	136, 251		
<code>\flag_new:n</code>	134		
<code>\flag_raise:n</code>	252		
<code>\fontencoding</code>	6		
<code>\fontfamily</code>	6		
<code>\fontseries</code>	6		
<code>\fontshape</code>	6		
<code>\fontsize</code>	6		
G			
group commands:			
<code>\group_begin:</code>	67, 75, 173,		
311, 945, 1037, 1045, 1080, 1097, 1123			
<code>\group_end:</code>	74, 78, 213,		
350, 963, 1041, 1051, 1090, 1108, 1265			
H			
<code>\halign</code>	44		
hbox commands:			
<code>\hbox_set:Nn</code>	171, 172		
hook commands:			
<code>\hook_gput_code:nnn</code> .	7, 11, 33, 57,		
66, 80, 156, 239, 288, 289, 353, 371,			
387, 391, 759, 768, 777, 786, 795,			
804, 812, 821, 829, 838, 848, 861,			
871, 884, 894, 907, 917, 930, 939, 952			
<code>\hook_new:n</code>	348		
<code>\hook_use:n</code>	353		
I			
<code>\IfPDFManagementActiveF</code>	6		
<code>\ignorespaces</code>	41		

ior commands:		
\ior_close:N	330, 467	
\ior_map_inline:Nn	326, 438	
\ior_open:Nn	324, 433, 436	
\g_tmpa_ior	324, 326, 330, 433, 436, 438, 467	
iow commands:		
\iow_newline:	205, 303	
\iow_term:n	198, 211, 214, 220, 224, 241, 355, 359, 363, 367, 371, 375, 379	
K		
kernel internal commands:		
__kernel_pdffdict_name:n	45	
\g__kernel_pdfmanagement_end_-run_code_tl	972	
keys commands:		
\keys_define:nn	8, 31, 34, 39, 51, 131, 143, 182, 195, 203, 219, 238, 246, 255, 291, 385, 394, 403, 410, 416, 570, 664, 737, 740, 752, 760, 915, 983, 989, 1054, 1076, 1093, 1506, 1517, 1551	
\keys_set:nn	10, 18, 18, 19, 128, 187, 190, 296, 318, 321, 338, 342, 428, 978, 1087, 1149	
\keys_set_known:nnnN	765	
L		
label (key)	1, 740	
\label	12	
label_(mc-key)	83, 238, 384	
lang (key)	1, 740	
lang_(mc-key=	238	
legacy commands:		
\legacy_if:nTF	472, 475, 476	
\llap	429	
log (deprecated) (key)	236	
ltx. internal commands:		
ltx.__tag.func.alloctag	312	
ltx.__tag.func.check_parent_-child_rules	974	
ltx.__tag.func.fakespace	491	
ltx.__tag.func.fill_parent_tree_-line	866	
ltx.__tag.func.get_num_from	321	
ltx.__tag.func.get_tag_from	340	
ltx.__tag.func.mark_page_-elements	692	
ltx.__tag.func.mark_shipout	849	
ltx.__tag.func.markspaceoff	557	
ltx.__tag.func.markspaceon	557	
ltx.__tag.func.mc_insert_kids	629	
ltx.__tag.func.mc_num_of_kids	370	
ltx.__tag.func.output_num_from	321	
ltx.__tag.func.output_parenttree	866	
ltx.__tag.func.output_tag_from	340	
ltx.__tag.func.role_get_parent_-child_rule	967	
ltx.__tag.func.space_chars_-shipout	589	
ltx.__tag.func.store_mc_data	355	
ltx.__tag.func.store_mc_in_page	673	
ltx.__tag.func.store_mc_kid	364	
ltx.__tag.func.store_mc_label	360	
ltx.__tag.func.store_struct_-mcabs	661	
ltx.__tag.func.update_mc_-attributes	681	
ltx.__tag.tables.role_tag_-attribute	310	
ltx.__tag.trace.log	224	
ltx.__tag.trace.show_all_mc_data	281	
ltx.__tag.trace.show_mc_data	266	
ltx.__tag.trace.show_prop	241	
ltx.__tag.trace.show_seq	232	
ltx.__tag.trace.show_struct_data	287	
lua commands:		
\lua_escape:n	32	
\lua_now:n	8, 12, 15, 18, 25, 25, 26, 32, 35, 38, 42, 43, 49, 50, 54, 57, 59, 61, 62, 62, 66, 68, 68, 69, 73, 76, 86, 87, 87, 88, 96, 100, 109, 111, 120, 133, 137, 138, 152, 158, 161, 173, 181, 189, 230, 237, 244, 252, 268, 282, 303, 317, 327, 399, 408, 497, 510, 752	
M		
\MakeLinkTarget	152	
mathml (key)	1, 938	
\maxdimen	192	
mc-current	20, 16	
mc-current_(show-key)	42, 143	
mc-data_(show-key)	42, 131	
mc-label-unknown	20, 9	
mc-marks_(show-key)	42, 203	
mc-nested	20, 6	
mc-not-open	20, 13	
mc-popped	20, 14	
mc-pushed	20, 14	
mc-tag-missing	20, 8	
mc-used-twice	20, 12	
\MessageBreak	10, 14, 15	
msg commands:		
\msg_error:nn	299, 320, 491, 1161	
\msg_error:nnn	336, 347, 355, 366, 477, 1531, 1571	

\msg_error:nnnnn	546, 555	page/exclude-header-footer_(setup-key)	43, 664
\msg_info:nnn	134, 172, 313, 325, 333, 389, 393	page/tabsorder (key)	1, 254
\msg_info:nnnn	343, 362, 402	para-flattened_(deprecated)	394
\msg_line_context:	93, 97, 107, 114, 506, 507, 539, 543, 547, 603, 611	para-hook-count-wrong	21, 225
\g_msg_module_name_prop	24, 28	para/flattened_(tool-key)	394
\g_msg_module_type_prop	27	para/maintag_(setup-key)	394
\msg_new:nnn	7, 8, 9, 12, 13, 14, 15, 16, 22, 24, 25, 32, 35, 36, 38, 40, 42, 47, 54, 65, 74, 85, 86, 87, 88, 89, 90, 92, 94, 104, 111, 164, 213, 215, 216, 217, 218, 219, 220, 222, 506, 507, 537, 541, 545, 597, 605	para/maintag_(tool-key)	394
\msg_new:nnnn	225	para/tag_(setup-key)	394
\msg_note:nn	29, 199	para/tag_(tool-key)	394
\msg_note:nnn	161, 178, 527, 534, 569, 577	para/tagging_(setup-key)	43, 394
\msg_note:nnnn	128, 184, 203, 513, 520, 554, 562, 605	para/tagging_(tool-key)	394
\msg_note:nnnnn	519, 569	\PARALABEL	496
\msg_redirect_name:nnn	542	paratag_(deprecated)	394
\msg_show_item_unbraced:n	276	paratagging_(deprecated)	43, 394
\msg_show_item_unbraced:nn	267	paratagging-show_(deprecated)	43, 394
\msg_term:nnnnnn	261, 270	parent (key)	1, 740
\msg_warning:nn	24, 222	pdf commands:	
\msg_warning:nnn	12, 14, 42, 45, 54, 242, 248, 306, 321, 329, 374, 382, 407, 431, 666, 693, 891, 904, 1348, 1367, 1393	\pdf_activate_indexed_structure-destination:	311
\msg_warning:nnnn	458, 590, 762	\pdf_bdc:nn	237
\msg_warning:nnnnn	126, 175, 530, 580, 630, 708	\pdf_bdc_shipout:nn	238
\msg_warning:nnnnnn	146	\pdf_bmc:n	235
		\l_pdf_current_structure-destination_tl	309
N		\pdf_emc:	236
\n	1001	\pdf_name_from_unicode_e:n	122, 132, 137, 185, 190, 198, 278, 1049, 1502, 1525, 1561
namespace_(rolemap-key)	185	\pdf_object_if_exist:n	96
new-tag	21, 215	\pdf_object_if_exist:nTF	993, 1058
newattribute_(deprecated)	116, 1499	\pdf_object_new:n	115, 34, 36, 154, 262, 310, 321
\newcommand	563, 564	\pdf_object_new_indexed:nn	31, 1130
\newcounter	8	\pdf_object_ref:n	115, 56, 97, 131, 135, 159, 192, 318, 335, 996, 1060, 1107
\NewDocumentCommand	6, 23, 29, 34, 40, 46, 51, 56, 126, 316, 568	\pdf_object_ref_indexed:nn	57, 74, 96, 126, 211, 273, 289, 432, 453, 514, 542, 1401
\newmarks	13	\pdf_object_ref_last:	115, 104, 118, 124, 312, 1466, 1472, 1580
\NewSocketPlug	590, 598	\pdf_object_unnamed_write:nn	100, 111, 120, 304, 1458, 1575
\NewTaggingSocketPlug	696, 702	\pdf_object_write:nnn	257, 281, 311, 330, 337, 342
no-struct-dest (deprecated) (key)	1	\pdf_object_write_indexed:nnnn	139, 467
\nointerlineskip	185	\pdf_pageobject_ref:n	239, 504, 532
		\pdf_string_from_unicode:nnN	42, 819
P		\pdf_uncompress:	246, 248
\PackageError	8	\pdf_version_compare:NnTF	20, 81, 154, 154, 177, 227, 257, 324, 351, 431, 471, 590, 797, 817
\PackageWarning	22		

pdfannot commands:

- \pdfannot_dict_put:nnn 98, 855, 878, 901, 924, 946
- \pdfannot_link_ref_last: 865, 888, 911, 934, 956

pdfdict commands:

- \pdfdict_gput:nnn 38, 45, 53, 187, 276, 334, 822
- \pdfdict_if_empty:nTF 328
- \pdfdict_new:n 18, 35, 37
- \pdfdict_put:nnn 1038, 1039, 1046, 1047, 1048, 1081, 1098
- \pdfdict_use:n 283, 332, 339

\pdffakespace 42, 314

pdffile commands:

- \pdffile_embed_file:nnn 106, 1083, 1100
- \pdffile_embed_stream:nnN . 939, 947
- \pdffile_embed_stream:nnn 99

\pdfglyphtounicode 30

\pdfinterwordspaceoff 211, 117

\pdfinterwordspaceon 211, 33, 121

pdfmanagement commands:

- \pdfmanagement_add:nnn 52, 70, 71, 256, 258, 260, 393, 1104
- \pdfmanagement_remove:nn 262

phoneme (key) 740

prg commands:

- \prg_do_nothing: 37, 82, 102, 117, 378, 379, 380, 381, 414, 683, 684, 685, 686
- \prg_generate_conditional_variant:Nnn 96
- \prg_new_conditional:Nnn ... 68, 226
- \prg_new_conditional:Npnn 233, 257, 272, 282, 481, 487, 498
- \prg_new_eq_conditional:NNn . 82, 233
- \prg_new_protected_conditional:Npnn 719
- \prg_replicate:nn 171
- \prg_return_false: 78, 230, 234, 252, 263, 266, 279, 289, 484, 496, 502, 749
- \prg_return_true: .. 79, 229, 249, 262, 276, 286, 485, 495, 501, 719, 750
- \prg_set_conditional:Npnn 238
- \prg_set_protected_conditional:Npnn 721

process commands:

- process_softhyphen_preuuuprocess_softhyphen_post 919

\ProcessOptions 39

prop commands:

- \prop_clear:N 176
- \prop_count:N 203
- \prop_gclear:N 989
- \prop_get:NnN 127, 144, 145, 177, 198, 214, 268, 299, 456, 468, 470, 483, 484, 485, 546, 547, 603, 604
- \prop_get:NnNTF 44, 96, 130, 137, 144, 158, 183, 199, 205, 219, 236, 294, 295, 304, 324, 339, 358, 395, 405, 419, 450, 514, 564, 599, 604, 609, 614, 650, 677, 703, 719, 770, 780, 785, 883, 896, 1198, 1296, 1362, 1421, 1459, 1529, 1569, 1573
- \prop_gput:Nnn 24, 26, 27, 28, 31, 56, 88, 90, 91, 97, 98, 99, 100, 100, 101, 103, 110, 112, 113, 116, 119, 121, 122, 141, 145, 269, 272, 286, 291, 383, 448, 452, 454, 455, 469, 475, 481, 488, 490, 491, 744, 990, 992, 1247, 1258, 1333, 1378, 1501, 1533, 1580
- \prop_gremove:Nn 137, 165, 993
- \prop_gset_eq:NN 164, 1244
- \prop_gset_from_keyval:Nn 963
- \prop_if_exist:NnTF 174, 209, 234, 320, 448, 648, 675, 1318, 1359
- \prop_if_exist_p:N 755
- \prop_if_in:NnTF 94
- \prop_item:Nn 41, 98, 99, 102, 102, 109, 115, 145, 262, 551, 1255, 1578, 1585
- \prop_map_function:NN 265
- \prop_map_inline:Nn 267, 272, 293, 326, 363, 381, 416, 486, 495, 976
- \prop_map_tokens:Nn 344
- \prop_new:N .. 7, 8, 9, 10, 11, 11, 25, 33, 72, 138, 162, 962, 1126, 1494, 1497
- \prop_new_linked:N 17, 84, 89, 91, 139, 1495
- \prop_put:Nnn 101, 188
- \prop_show:N 67, 95, 147, 1241, 1262, 1547, 1574

property commands:

- \property_new:nnnn 121, 124, 128, 131, 135
- \property_record:nn 58, 110
- \property_ref:nn 114, 115
- \property_ref:nnn 42, 114, 119, 181, 190, 239, 240, 325, 460, 471, 505, 1319, 1323

\providecommand 62, 63, 64, 65, 66, 69, 70, 321

\ProvidesExplFile 3

\ProvidesExplPackage 3, 3, 3, 3, 3, 3, 3, 3, 7, 7, 20, 31, 1490

Q

\quad 233, 234
 quark commands:
 \q_no_value 606, 616, 662, 667, 689, 694
 \quark_if_no_value:NTF
 132, 178, 199, 215, 269, 300, 609, 620
 \quark_if_no_value_p:N
 549, 550, 620, 621, 698, 699
 \q_stop 259, 292, 328

R

raw_□(mc-key) 83, 238, 384
 ref (key) 1, 740, 915
 \RemoveFromHook 34, 525, 526
 \renewcommand 566, 567
 \RenewDocumentCommand 8
 \RequirePackage ... 40, 274, 277, 283, 286
 \rlap 440
 role_□(rolemap-key) 185, 752
 role commands:
 role_get_parent_child_rule 967
 role-MC-child-forbidden 104
 role-missing 21, 86
 role-namespace_□(rolemap-key) . 185, 752
 role-parent-child-check 90
 role-parent-child-forbidden 111
 role-parent-child-result 21, 92
 role-parent-child-unresolved 164
 role-remapping 21, 213
 role-struct-parent-child-forbidden . 94
 role-tag 21, 215
 role-unknown 21, 86
 role-unknown-NS 21, 86
 role-unknown-tag 21, 86
 role/new-attribute_□(setup-key) 116, 1499
 role/new-tag_□(setup-key) 752
 root-AF (key) 1, 1054
 root-supplemental-file (key) 1076

S

\selectfont 6
 seq commands:
 \seq_clear:N 337, 448
 \seq_const_from_clist:Nn 39, 52
 \seq_count:N 22, 25, 58,
 349, 462, 1537, 1539, 1541, 1565, 1591
 \seq_get:NN 723, 736
 \seq_get:NTF 487, 584, 1157, 1284, 1292
 \seq_gpop:NN 1277
 \seq_gpop:NTF 106, 1278
 \seq_gpop_left:NN 325
 \seq_gpush:Nn 13, 15, 89, 96, 1164, 1170
 \seq_gput_left:Nn .. 42, 143, 291, 329

\seq_gput_right:Nn
 . 37, 142, 146, 152, 254, 275, 314, 468
 \seq_gset_eq:NN 159, 221, 344
 \seq_if_empty:NTF 200, 456
 \seq_item:Nn
 . 59, 116, 118, 125, 129, 136, 140,
 144, 366, 373, 386, 491, 493, 500,
 712, 713, 728, 729, 775, 776, 779, 780
 \seq_log:N 175, 199, 249, 394, 555, 570
 \seq_map_function:NN 274
 \seq_map_indexed_inline:Nn 446, 459
 \seq_map_inline:Nn 289, 338, 1527, 1567
 \seq_new:N
 12, 14, 14, 15, 16, 17, 18, 19,
 21, 22, 24, 73, 74, 140, 163, 1129, 1498
 \seq_pop_left:NN 455, 457, 458
 \seq_put_right:Nn 339
 \seq_remove_all:Nn 342
 \seq_set_eq:NN 207, 208
 \seq_set_from_clist:NN ... 1522, 1558
 \seq_set_from_clist:Nn
 87, 90, 196, 216, 443, 454
 \seq_set_map_e:NNn 1523, 1559
 \seq_set_split:Nnn 51,
 103, 705, 709, 721, 725, 772, 774, 776
 \seq_show:N
 60, 146, 216, 217, 250, 340,
 341, 343, 478, 1174, 1242, 1263, 1273
 \seq_use:Nn
 50, 110, 111, 205, 233, 234, 401, 1538

Setup keys:

activate-all (deprecated) 1
 activate-mc (deprecated) 1
 activate-struct (deprecated) 1
 activate-tree (deprecated) 1
 activate/all 1, 218
 activate/mc 1, 218
 activate/softhyphen 1, 252
 activate/spaces 1
 activate/struct 1, 218
 activate/struct-dest 1, 218
 activate/tagunmarked 1, 249
 activate/tree 1, 218
 catalog-supplemental-file 1093
 debug/log 1, 236
 debug/show 235
 debug/uncompress 236
 log (deprecated) 236
 no-struct-dest (deprecated) 1
 page/tabsorder 1, 254
 root-AF 1, 1054
 root-supplemental-file 1076
 tabsorder (deprecated) 1, 254
 tagunmarked (deprecated) 1, 249

uncompress (deprecated)	236	AFinline-o	1 , 938
shipout commands:		AFref	1 , 938
\g_shipout_readonly_int		alt	1 , 740
.....	131 , 130 , 241 , 396 , 533	attribute	1 , 1551
show-kids	21 , 64	attribute-class	1 , 1517
show-spaces _□ (deprecated)	209 , 6	E	1 , 740 , 915
show-struct	21 , 64	firstkid	1 , 740
\ShowTagging	18 , 42 , 125	label	1 , 740
skip commands:		lang	1 , 740
\skip_horizontal:n	77	mathml	1 , 938
\c_zero_skip	77	parent	1 , 740
socket commands:		phoneme	740
\socket_assign_plug:nn		ref	1 , 740 , 915
.....	200 , 204 , 205 ,	stash	1 , 740
.....	209 , 210 , 518 , 519 , 535 , 739 , 813 , 814	tag	1 , 740
\socket_if_exist:nTF	694	texsource	1 , 938
\socket_new:nn .	183 , 184 , 446 , 447 , 700	title	1 , 740
\socket_new_plug:nnn		title-o	1 , 740
.....	185 , 449 , 468 , 501 , 701 , 717	\SuspendTagging	44
\socket_use:n	28 , 76 , 520 , 522 , 529 , 533	sys commands:	
\socket_use:nn		\c_sys_backend_str	46
..	81 , 205 , 341 , 793 , 1223 , 1340 , 1385	\c_sys_engine_str	12 , 14
\socket_use:nnn	86	\sys_if_engine luatex:TF	
\socket_use:nw	97	21 , 36 , 49 , 71 ,
\socket_use_expandable:n	92	83 , 84 , 105 , 187 , 266 , 361 , 391 , 493 , 501
\socket_use_expandable:nw ..	66 , 112	\sys_if_engine luatex_p:	748
stash (key)	1 , 740	\sys_if_engine pdftex:TF ..	26 , 111
stash _□ (mc-key)	84 , 182	\sys_if_output_pdf:TF	11 , 28 , 113
str commands:		sys-no-interwordspace	21 , 222
\str_case:nnTF	46 , 661 , 1181		
\str_const:Nn	59		
\str_if_eq:nnTF	117 , 127 , 500 , 586 , 673		
\str_if_eq_p:nn	310 , 491 , 493		
\str_if_exist:NTF	444 , 588		
\str_new:N	71		
\str_set_convert:Nnnn	104 , 261 , 296 ,		
.....	398 , 415 , 797 , 809 , 823 , 839 , 854 , 927		
\str_use:N	67 , 272 , 309		
\c_tilde_str	57 , 59		
\string	15 , 16		
struct-faulty-nesting	20 , 32		
struct-label-unknown	20 , 38		
struct-missing-tag	20 , 35		
struct-no-objnum	20 , 24		
struct-orphan	20 , 25		
struct-Ref-unknown	42		
struct-show-closing	20 , 40		
struct-stack _□ (show-key)	42 , 246		
struct-unknown	20 , 22		
struct-used-twice	20 , 36		
Structure keys:			
actualtext	1 , 740		
AF	1 , 938		
AFinline	1 , 938		

```

\tag_mc_begin:n ..... 10, 82, 25, 66, 114, 169,
    169, 295, 295, 299, 305, 428, 439,
    465, 497, 619, 647, 699, 765, 783,
    801, 818, 835, 854, 877, 900, 923, 945
\tag_mc_begin_pop:n ..... 82, 76,
    80, 81, 102, 628, 658, 731, 774, 792,
    810, 827, 844, 868, 891, 914, 937, 959
\tag_mc_end: ..... 82, 31, 75, 93,
    216, 216, 295, 296, 359, 365, 430,
    441, 507, 625, 654, 704, 772, 790,
    808, 825, 842, 866, 889, 912, 935, 957
\tag_mc_end_push: ..... 82, 65,
    80, 80, 83, 613, 640, 717, 763, 781,
    799, 816, 833, 852, 875, 898, 921, 943
\tag_mc_if_in: ..... 82, 233
\tag_mc_if_in:TF ..... 82, 42, 68, 226
\tag_mc_if_in_p: ..... 82, 68, 226
\tag_mc_new_stream:n 83, 17, 17, 67, 67
\tag_mc_reset_box:N 82, 79, 79, 228, 228
\tag_mc_use:n ..... 82, 36, 36, 36, 38
\l_tag_para_attr_class_tl . 389, 391
\tag_resume:n .....
    ... 7, 73, 157, 193, 206, 216, 624, 653
\tag_socket_use:n .. 44, 45, 62, 72, 73
\tag_socket_use:nn . 44, 45, 63, 72, 78
\tag_socket_use:nnn . 44, 45, 64, 72, 83
\tag_socket_use_expandable:n ...
    ..... 44, 45, 65, 72, 89
\tag_spacechar_off: ... 81, 81, 86, 115
\tag_spacechar_on: ... 81, 82, 98, 119
\tag_start: ..... 7, 157, 168, 181, 210
\tag_start:n .... 7, 157, 206, 214, 216
\tag_stop: ... 7, 54, 157, 159, 180, 209
\tag_stop:n ..... 7, 157, 192, 213, 215
\tag_struct_begin:n ..... 113, 48,
    456, 463, 481, 491, 646, 698, 723,
    764, 782, 800, 817, 834, 853, 876,
    899, 922, 944, 1113, 1113, 1117, 1118
\tag_struct_end: ..... 113,
    26, 53, 509, 513, 655, 705, 728, 773,
    791, 809, 826, 843, 867, 890, 913,
    936, 958, 1113, 1114, 1269, 1270, 1309
\tag_struct_end:n ... 113, 1115, 1306
\tag_struct_gput:nnn .....
    .... 113, 921, 1405, 1405, 1407, 1415
\tag_struct_gput_ref:nnn ..... 114
\tag_struct_insert_annot:nn .....
    ..... 113, 154, 865,
    888, 911, 934, 956, 1475, 1475, 1484
\tag_struct_object_ref:n .....
    . 113, 887, 900, 911, 1398, 1399, 1403

\tag_struct_parent_int: .....
    .... 113, 154, 858, 865, 881, 888,
    904, 911, 927, 934, 949, 956, 1475, 1485
\tag_struct_use:n .....
    ..... 113, 114, 58, 1312, 1312, 1314
\tag_struct_use_num:n .....
    ..... 113, 1353, 1353, 1355
\tag_suspend:n .....
    ... 7, 68, 157, 182, 192, 215, 620, 648
\tag_tool:n ..... 41, 13, 13, 14, 16, 20
tag internal commands:
__tag_activate_mark_space ..... 557
\g__tag_active_mc_bool .....
    ..... 40, 82, 221, 228, 243, 274
\l__tag_active_mc_bool .....
    .... 88, 164, 174, 188, 199, 246, 274
\l__tag_active_socket_bool ....
    ..... 75, 80, 85,
    88, 91, 96, 111, 165, 175, 189, 200, 293
\g__tag_active_space_bool .....
    ..... 13, 56, 61, 82
\g__tag_active_struct_bool ....
    .... 82, 223, 230, 242, 284, 307, 484
\l__tag_active_struct_bool ....
    .... 88, 163, 173, 187, 198, 245, 284
\g__tag_active_struct_dest_bool
    ..... 82, 227, 234, 306
\g__tag_active_tree_bool .....
    .... 9, 68, 82, 222, 229, 244, 351, 389
__tag_add_missing_mcs:Nn .....
    ..... 97, 98, 167, 167, 219
__tag_add_missing_mcs_to_-
    stream:Nn . 65, 65, 66, 189, 189, 225
\g__tag_attr_class_used_prop ...
    ..... 291, 293, 1493, 1533
\g__tag_attr_class_used_seq 289, 1498
\g__tag_attr_entries_prop .....
    295, 1493, 1501, 1529, 1569, 1574, 1578
__tag_attr_new_entry:nn .....
    ... 634, 1499, 1499, 1505, 1510, 1514
\g__tag_attr_objref_prop .....
    ..... 1493, 1573, 1580, 1585
\l__tag_attr_value_tl ..... 1493,
    1563, 1582, 1587, 1589, 1593, 1597
__tag_backend_create_bdc_node .. 436
__tag_backend_create_bmc_node .. 407
__tag_backend_create_emc_node .. 378
__tag_check_add_tag_role:nn ...
    ..... 129, 332, 332
__tag_check_add_tag_role:nnn ..
    ..... 169, 351
__tag_check_benchmark_tic: . 356,
    360, 364, 368, 372, 376, 380, 615, 621

```

__tag_check_benchmark_toc: .	358, 362, 366, 370, 374, 378, 382, 616, 622
__tag_check_forbidden_parent_-	
child:nnnn	<u>120</u> , 120, 134, 171
__tag_check_if_active_mc:	272
__tag_check_if_active_mc:TF	
.	85, 104, 171, 191, 218, <u>271</u> , 301, 307, 361, 367
__tag_check_if_active_struct: .	282
__tag_check_if_active_struct:TF	40, <u>271</u> , 1120, 1121, 1274, 1275, 1308, 1316, 1357, 1478
__tag_check_if_mc_in_galley: . .	481
__tag_check_if_mc_in_galley:TF	209, 230
__tag_check_if_mc_tmb_missing: .	487
__tag_check_if_mc_tmb_missing:TF	112, 218, 235, <u>487</u>
__tag_check_if_mc_tmb_missing_-	
p:	<u>487</u>
__tag_check_if_mc_tme_missing: .	498
__tag_check_if_mc_tme_missing:TF	155, 222, 239, <u>498</u>
__tag_check_if_mc_tme_missing_-	
p:	<u>498</u>
__tag_check_info_closing_-	
struct:n	309, 309, 317, 1280
__tag_check_init_mc_used:	
.	<u>411</u> , 411, 414, 420
__tag_check_mc_if_nested:	
.	174, 312, <u>370</u> , 370
__tag_check_mc_if_open:	220, <u>370</u> , 371, 378
__tag_check_mc_in_galley:TF . . .	<u>481</u>
__tag_check_mc_in_galley_p: . . .	<u>481</u>
__tag_check_mc_pushed_popped:nn	90, 97, 110, 113, 118, <u>385</u> , 385
__tag_check_mc_tag:N	
.	193, 330, <u>397</u> , 397
__tag_check_mc_used:n	
.	145, 268, <u>416</u> , 416
\g__tag_check_mc_used_intarray	<u>411</u> , 421, 423, 426
__tag_check_no_open_struct: . . .	
.	<u>318</u> , 318, 1282, 1290
__tag_check_para_begin_show:nn	423, 464, 496
__tag_check_para_end_show:nn	434, 508
__tag_check_show_MCID_by_page:	<u>435</u> , 435
__tag_check_struct_forbidden_-	
parent_child:nnn . . .	137, 163, 646
__tag_check_struct_used:n	
.	<u>322</u> , 322, 1321
__tag_check_structure_has_tag:n	<u>292</u> , 292, 1154
__tag_check_structure_tag:N	302, 302, 714, 737, 788
__tag_check_typeout_v:n	110, 111, 114, 149, 157, 164, 202, 211, <u>224</u> , 224, 241, 474, 490, 506
__tag_check_unresolved_parent_-	
child:nnnn	169, 169
\g__tag_css_bool	970, 971, 974, 985
\g__tag_css_prop	
.	962, 963, 976, 989, 990, 992, 993
__tag_debug_mc_begin_ignore:n	354, 516
__tag_debug_mc_begin_insert:n	309, 509
__tag_debug_mc_end_ignore: .	379, 530
__tag_debug_mc_end_insert: .	369, 523
__tag_debug_struct_begin_-	
ignore:n	558, 1267
__tag_debug_struct_begin_-	
insert:n	550, 1264
__tag_debug_struct_end_check:n	580, 1308
__tag_debug_struct_end_ignore:	573, 1303
__tag_debug_struct_end_insert:	565, 1301
__tag_exclude_headfoot_begin:	608, 669, 670
__tag_exclude_headfoot_end:	622, 671, 672
__tag_exclude_struct_headfoot_-	
begin:n	635, 676, 677
__tag_exclude_struct_headfoot_-	
end:	651, 678, 679
__tag_fakespace	<u>491</u>
__tag_fakespace:	<u>71</u> , 73, 318
__tag_finish_structure:	13, 16, <u>348</u> , 349
\l__tag_get_child_tmpa_tl	59, 587, 592, 659, 661, 671, 674, 685, 1322, 1326, 1328, 1334, 1344
\l__tag_get_child_tmpb_tl	59, 588, 593, 660, 672
\l__tag_get_child_tmpc_tl	59, 145, 157, 159
__tag_get_data_mc_counter:	<u>9</u> , 9
__tag_get_data_mc_tag:	<u>237</u> , 237, <u>293</u> , 293
__tag_get_data_struct_counter:	<u>565</u> , 566

__tag_get_data_struct_id:	554 , 554	388 , 391 , 418 , 511 , 512 , 518 , 525 ,
__tag_get_data_struct_num:	559 , 560	532 , 552 , 560 , 562 , 567 , 575 , 582 , 601
__tag_get_data_struct_tag:	546 , 546	_tag_mark_spaces
__tag_get_mathsubtype	302	496
__tag_get_mc_abs_cnt:		__tag_mc_artifact_begin_marks:n
..... 14 , 15 , 19 , 20 , 102 ,	 23 , 45 , 81 , 327
137 , 166 , 177 , 183 , 210 , 246 , 254 ,		\l__tag_mc_artifact_bool
272 , 286 , 307 , 321 , 331 , 374 , 382 , 402	 20 , 176 , 185 , 196 , 222 , 323
__tag_get_mc_cnt_type_tag	296	\l__tag_mc_artifact_type_tl
__tag_get_num_from	321 19 , 189 , 193 , 197 ,
\l__tag_get_parent_tmpa_tl		201 , 205 , 209 , 213 , 217 , 325 , 327 , 344
. 59 , 127 , 132 , 136 , 139 , 149 , 152 ,		__tag_mc_bdc:nn
162 , 165 , 175 , 582 , 590 , 603 , 609 ,		234 , 237 , 283
613 , 616 , 683 , 685 , 727 , 730 , 740 , 743		__tag_mc_bdc_mcid:n ...
\l__tag_get_parent_tmpb_tl		123 , 239 , 255
..... 59 , 150 ,		__tag_mc_bdc_mcid:nn
153 , 163 , 166 , 175 , 583 , 591 , 604 ,	 239 , 240 , 257 , 262
620 , 624 , 627 , 684 , 728 , 731 , 741 , 744		__tag_mc_bdc_shipout:nn ..
\l__tag_get_parent_tmpc_tl		238 , 248
..... 59 , 144 , 152 , 154		__tag_mc_begin_marks:nn
__tag_get_tag_from	340 23 , 23 , 44 , 80 , 334
\l__tag_get_tmpc_tl	59 ,	__tag_mc_bmc:n
199 , 204 , 222 , 224 , 225 , 236 , 238 ,		234 , 235 , 279
239 , 1201 , 1207 , 1424 , 1426 , 1430 , 1436		__tag_mc_bmc_artifact: 277 , 277 , 290
__tag_gincr_para_begin_int: ...		__tag_mc_bmc_artifact:n 277 , 281 , 291
..... 341 , 345 , 363 , 379 , 462 , 489		\l__tag_mc_botmarks_seq
__tag_gincr_para_end_int: 97 , 21 , 90 , 111 ,
..... 341 , 353 , 371 , 381 , 505		161 , 208 , 216 , 217 , 221 , 234 , 483 , 500
__tag_gincr_para_main_begin_int: ..		__tag_mc_check_parent_child:n .
... 341 , 341 , 359 , 378 , 455 , 480	 122 , 122 , 181 , 207 , 343
__tag_gincr_para_main_end_int:		__tag_mc_disable_marks:
..... 341 , 349 , 367 , 380 , 512		78 , 78
__tag_headfoot_tagged_begin:n .		__tag_mc_emc:
..... 710 , 741 , 742		158 , 234 , 236 , 374
__tag_headfoot_tagged_end:		__tag_mc_end_marks: ..
..... 726 , 743 , 744		23 , 63 , 82 , 375
__tag_hook_kernel_after_foot: .		\l__tag_mc_firstmarks_seq
.... 577 , 585 , 602 , 672 , 679 , 686 , 744	 97 , 21 , 87 , 110 , 196 , 199 ,
__tag_hook_kernel_after_head: .		200 , 207 , 208 , 216 , 233 , 483 , 491 , 493
.... 575 , 583 , 594 , 671 , 678 , 685 , 743		\g__tag_mc_footnote_marks_seq ...
__tag_hook_kernel_before_foot:		14
.... 576 , 584 , 600 , 670 , 677 , 684 , 742		__tag_mc_get_marks: .
__tag_hook_kernel_before_head:		84 , 84 , 208 , 229
.... 574 , 582 , 592 , 669 , 676 , 683 , 741		__tag_mc_handle_artifact:N
\g__tag_in_mc_bool	16 , 119 , 277 , 285 , 325
18 , 175 , 221 , 228 , 313 , 372 , 616 ,		__tag_mc_handle_mc_label:n
617 , 631 , 643 , 644 , 661 , 720 , 721 , 734	 27 , 27 , 200 , 337
__tag_insert_bdc_node	436	__tag_mc_handle_mcid:nn
__tag_insert_bmc_node	407 239 , 260 , 265 , 331
__tag_insert_emc_node	378	__tag_mc_handle_stash:n
__tag_log	224	50 , 140 ,
\l__tag_loglevel_int		142 , 143 , 168 , 210 , 266 , 266 , 276 , 346
..... 81 , 125 , 132 , 170 , 173 , 237 ,		__tag_mc_if_in:
240 , 243 , 244 , 245 , 311 , 341 , 360 ,		68 , 82 , 226 , 233
		__tag_mc_if_in:TF
		68 , 87 , 226 , 372 , 380
		__tag_mc_if_in_p:
		68 , 226
		__tag_mc_insert_extra_tmb:n ...
	 108 , 108 , 171
		__tag_mc_insert_extra_tme:n ...
	 108 , 153 , 172
		__tag_mc_insert_mcid_kids:n ...
	 131 , 131 , 150 , 327
		__tag_mc_insert_mcid_single_kids:n
	 131 , 136 , 328
		\l__tag_mc_key_label_tl
		. 23 , 198 , 200 , 316 , 334 , 335 , 337 , 424

\l__tag_mc_key_properties_tl ...	\l__tag_para_main_tag_tl
. 23, 177, 251, 266, 267, 281, 301, 322, 400, 407, 420, 458, 483
302, 333, 394, 403, 404, 409, 420, 421	\l__tag_para_show_bool
\l__tag_mc_key_stash_bool 322, 397, 398, 413, 426, 437
..... 20, 31, 40, 184, 203, 339	\l__tag_para_tag_default_tl 322
\g__tag_mc_key_tag_tl 19, 23,	\l__tag_para_tag_tl
180, 225, 237, 243, 293, 315, 373, 390 322, 399, 406, 414, 419, 463, 493
\l__tag_mc_key_tag_tl 23, 179, 193,	\l__tag_parent_child_check_tl ..
195, 224, 242, 314, 330, 332, 334, 389 156, 157, 169, 172, 500,
\l__tag_mc_lang_tl	636, 637, 644, 647, 733, 734, 746, 748
..... 22, 185, 190, 316, 321	__tag_parenttree_add_objr:nn ..
__tag_mc_lua_set_mc_type_attr:n 163, 163, 509, 537
..... 83, 83, 107, 195	\l__tag_parenttree_content_tl ..
__tag_mc_lua_unset_mc_type_- 170, 195, 207, 227, 235, 256, 259
attr:	\g__tag_parenttree_objr_tl
83, 109, 223 162, 165, 256
\g__tag_mc_main_marks_seq	__tag_pdf_name_e:n
14	122, 122
\g__tag_mc_marks	__tag_pdf_object_ref
13,	466
25, 34, 47, 54, 65, 71, 88, 91, 197, 217	__tag_prop_gput:Nnn
\g__tag_mc_multicol_marks_seq ... 14 9, 29, 89, 98, 111, 114,
\g__tag_mc_parenttree_prop	120, 121, 128, 132, 138, 141, 146,
..... 17, 18, 103, 184, 272	149, 149, 201, 205, 217, 220, 282,
\l__tag_mc_ref_abspage_tl	285, 314, 315, 384, 1327, 1463, 1470
11	__tag_prop_item:Nn ... 9, 52, 138, 145
__tag_mc_set_label_used:n 31, 31, 51	__tag_prop_new:N
\g__tag_mc_stack_seq	9, 9,
..... 18, 89, 96, 106, 394	11, 19, 24, 32, 125, 138, 138, 152, 1125
__tag_mc_store:nnn .. 93, 93, 107, 134	__tag_prop_new_linked:N
\l__tag_mc_tmpa_tl	15, 17, 138, 139
12	__tag_prop_show:N 9, 65, 138, 147, 155
g__tag_MCID_abs_int	\c__tag_property_mc_clist .. 79, 247
7	__tag_property_record:nn
\g__tag_mode_lua_bool 29, 107, 107, 116, 243, 495, 746
.. 35, 36, 135, 146, 248, 272, 281,	__tag_property_ref_lastpage:nn
314, 335, 611, 626, 638, 656, 715, 729	. 83, 117, 117, 160, 174, 177, 439, 453
__tag_new_output_prop_handler:n	\c__tag_property_struct_clist 79, 748
..... 92, 102, 126, 1127	\l__tag_Ref_tmpa_tl
__tag_pairs_prop	63
241	g__tag_role/RoleMap_dict
\l__tag_para_attr_class_tl	18
..... 322, 391, 494	\g__tag_role_add_mathml_bool ...
\g__tag_para_begin_int 73, 265, 762, 829
..... 322, 347, 365, 429, 553, 558	__tag_role_add_tag:nn
\l__tag_para_bool 127, 127, 153, 280, 365, 800
..... 322, 396, 405, 412, 418,	__tag_role_add_tag:nnnn
451, 470, 503, 566, 567, 610, 637, 714 167, 167, 226, 312, 805
\g__tag_para_end_int	__tag_role_alloctag:nnn 81,
..... 322, 355, 373, 440, 553, 559	85, 95, 107, 117, 126, 141, 184, 277, 308
\l__tag_para_flattened_bool	__tag_role_check_parent_-
.... 322, 401, 408, 421, 453, 478, 510	child:nnnnN
\l__tag_para_main_attr_class_tl	151,
..... 322, 484	164, 589, 590, 592, 640, 717, 729, 742
\g__tag_para_main_begin_int	\l__tag_role_debug_prop
..... 322, 343, 361, 544, 549	11
\g__tag_para_main_end_int	__tag_role_get:nnNN
..... 322, 351, 369, 544, 550	154,
__tag_para_main_store_struct: .	156, 164, 227, 229, 253, 730, 781, 1165
..... 383, 383, 460, 486	__tag_role_get_parent_child_-
\g__tag_para_main_struct_tl 322, 385	rule:nnN
 202, 500, 503, 541, 589, 623, 701

\g__tag_role_index_prop
 [186](#), [10](#), [448](#), [456](#), [468](#),
 [469](#), [470](#), [475](#), [481](#), [483](#), [484](#), [485](#),
 [488](#), [490](#), [491](#), [495](#), [546](#), [547](#), [599](#), [609](#)
 \g__tag_role_NS<ns>_class_prop [186](#)
 \g__tag_role_NS<ns>_prop [186](#)
 \g__tag_role_NS_mathml_prop [267](#), [486](#)
 __tag_role_NS_new:nnn
 . [188](#), [20](#), [22](#), [30](#), [74](#), [75](#), [76](#), [77](#), [78](#), [80](#)
 \g__tag_role_NS_prop
 [186](#), [9](#), [26](#), [56](#), [199](#), [326](#), [344](#), [786](#)
 \g__tag_role_parent_child_-
 intarray [390](#), [397](#), [406](#), [421](#), [425](#), [555](#)
 __tag_role_read_namespace:n [337](#),
 [337](#), [341](#), [342](#), [343](#), [345](#), [347](#), [349](#), [350](#)
 __tag_role_read_namespace:nn ..
 [318](#), [318](#), [339](#), [348](#)
 __tag_role_read_namespace_-
 line:nw [255](#), [259](#), [292](#), [328](#)
 \l__tag_role_role_namespace_-
 tmpa_tl [12](#),
 [757](#), [778](#), [783](#), [787](#), [790](#), [794](#), [809](#)
 \l__tag_role_role_tmpa_tl
 [12](#), [756](#), [776](#), [782](#), [802](#), [808](#)
 \g__tag_role_rolemap_prop
 [186](#), [18](#), [144](#), [146](#), [149](#), [158](#),
 [214](#), [217](#), [220](#), [269](#), [272](#), [385](#), [604](#), [614](#)
 \c__tag_role_rule_checkparent_tl
 [157](#), [173](#), [637](#), [734](#)
 \c__tag_role_rules_num_prop
 [391](#), [514](#), [564](#)
 \c__tag_role_rules_prop [391](#), [395](#), [419](#)
 \l__tag_role_tag_namespace_tmpa_-
 tl [12](#), [755](#), [807](#)
 \l__tag_role_tag_namespace_tmpb_-
 tl [14](#)
 \l__tag_role_tag_namespace_tmpb_-
 tluuuuu% [12](#)
 \l__tag_role_tag_tmpa_tl
 [12](#), [754](#), [775](#), [801](#), [806](#)
 \g__tag_role_tags_class_prop ...
 [186](#), [8](#), [90](#), [99](#), [112](#), [121](#), [137](#), [268](#)
 \g__tag_role_tags_NS_prop
 [186](#), [7](#), [88](#), [97](#), [110](#), [119](#), [130](#), [304](#),
 [339](#), [383](#), [405](#), [703](#), [719](#), [770](#), [781](#), [1296](#)
 \l__tag_role_tmpa_seq [12](#)
 \l__tag_role_update_bool
 [208](#), [255](#), [256](#), [264](#), [344](#), [346](#)
 \c__tag_role_userNS_id_str
 [187](#), [59](#), [80](#)
 \g__tag_root_default_tl [285](#)
 \g__tag_saved_in_mc_bool
 [607](#), [616](#), [631](#), [643](#), [661](#), [720](#), [734](#)
 __tag_seq_gput_left:Nn
 [9](#), [40](#), [143](#), [151](#), [286](#)
 __tag_seq_gput_right:Nn [9](#),
 [35](#), [138](#), [142](#), [150](#), [249](#), [259](#), [270](#), [309](#)
 __tag_seq_item:Nn ... [9](#), [47](#), [138](#), [144](#)
 __tag_seq_new:N
 [9](#), [9](#), [22](#), [127](#), [138](#), [140](#), [153](#), [1128](#)
 __tag_seq_show:N [9](#), [58](#), [138](#), [146](#), [154](#)
 __tag_show_spacemark [477](#)
 \l__tag_showspaces_bool ... [7](#), [16](#), [17](#)
 \g__tag_softthyphen_bool [94](#), [252](#)
 __tag_space_chars_shipout [589](#)
 __tag_start_para_ints:
 [176](#), [201](#), [357](#), [357](#)
 __tag_stop_para_ints:
 [166](#), [190](#), [357](#), [376](#)
 __tag_store_parent_child_-
 rule:nnn [391](#), [393](#), [417](#), [462](#)
 g__tag_struct_1_prop [124](#)
 __tag_struct_add_AF:nn
 [951](#), [968](#), [988](#), [995](#), [1015](#), [1060](#)
 __tag_struct_add_inline_AF:nn ..
 [940](#), [967](#), [1029](#), [1033](#), [1040](#), [1050](#)
 \l__tag_struct_addkid_tl [86](#), [790](#), [1238](#)
 \g__tag_struct_AFobj_int [938](#), [946](#), [949](#)
 __tag_struct_check_parent_-
 child:nn [596](#), [596](#), [651](#), [687](#), [696](#), [1225](#)
 __tag_struct_check_parent_-
 child_aux:nnnnN . [571](#), [572](#), [631](#), [639](#)
 \g__tag_struct_cont_mc_prop
 [11](#), [95](#), [96](#), [98](#), [101](#), [262](#)
 \g__tag_struct_dest_num_prop [88](#), [896](#)
 \l__tag_struct_elem_stash_bool ..
 [85](#), [750](#), [1188](#), [1221](#), [1251](#)
 __tag_struct_exchange_kid_-
 command:N [323](#), [323](#), [332](#), [363](#)
 __tag_struct_fill_kid_key:n ...
 [136](#), [333](#), [333](#), [465](#)
 __tag_struct_format_P:nnN [427](#)
 __tag_struct_format_parentnum:nnN
 [430](#), [430](#)
 __tag_struct_format_parentrole:nnN
 [427](#), [428](#)
 __tag_struct_format_Ref [139](#)
 __tag_struct_format_Ref:nnN [434](#), [434](#)
 __tag_struct_format_rolemap:nnN
 [427](#), [427](#)
 __tag_struct_format_tag:nnN [427](#), [429](#)
 __tag_struct_get_dict_content:nn
 [138](#), [413](#), [413](#), [466](#)
 __tag_struct_get_id:n
 . [96](#), [101](#), [114](#), [115](#), [166](#), [167](#), [472](#), [556](#)

```

__tag_struct_get_role:nnNN ....
    ..... 146, 159, 213, 213,
    232, 579, 584, 656, 668, 680, 724, 737
__tag_struct_gput_data_attribute:nn
    ..... 1456, 1456
__tag_struct_gput_data_ref:nn .
    ..... 1438, 1455
__tag_struct_gput_data_ref_-
    aux:nnn .....
    .. 1417, 1418, 1440, 1444, 1448, 1452
__tag_struct_gput_data_ref_-
    dest:nn ..... 1446
__tag_struct_gput_data_ref_-
    label:nn ..... 1442
__tag_struct_gput_data_ref_-
    num:nn ..... 1450
__tag_struct_insert_annot:nn ..
    ..... 480, 480, 1480
__tag_struct_insert_annot_-
    shipout:nnn ..... 521, 521
__tag_struct_kid_mc_gput_-
    right:nn ... 233, 245, 246, 265, 269
__tag_struct_kid_OBJR_gput_-
    right:nnn 298, 298, 301, 322, 496, 524
__tag_struct_kid_struct_gput_-
    left:nn ..... 282, 282, 283, 297
__tag_struct_kid_struct_gput_-
    right:nn .....
    ..... 266, 266, 267, 281, 1324, 1369
g__tag_struct_kids_1_seq ..... 124
g__tag_struct_label_num_prop ..
    ..... 84, 744, 883
l__tag_struct_lang_tl .....
    ..... 572, 1111, 1136, 1141
__tag_struct_mcid_dict:n .....
    ..... 98, 101, 233, 252
c__tag_struct_null_tl ..... 10, 367
g__tag_struct_objR_seq ..... 8
__tag_struct_output_prop_aux:nn
    ..... 92, 92, 106
l__tag_struct_parenttag_NS_tl .
    ..... 76, 780, 783, 787, 1194
l__tag_struct_parenttag_tl ....
    ..... 76, 779, 782, 786, 788, 1194
__tag_struct_prop_gput:nnn . 110,
    111, 112, 118, 129, 134, 139, 144,
    149, 156, 182, 186, 195, 201, 206,
    369, 382, 396, 802, 814, 828, 844,
    859, 867, 932, 954, 997, 1016, 1061,
    1132, 1138, 1143, 1175, 1190, 1203,
    1213, 1229, 1372, 1433, 1543, 1594
g__tag_struct_ref_by_dest_prop . 91
__tag_struct_Ref_dest:nN . 873, 894
__tag_struct_Ref_label:nN 873, 881

__tag_struct_Ref_num:nN .. 873, 907
__tag_struct_Ref_obj:nN .. 873, 873
g__tag_struct_roletag_NS_tl .... 76
l__tag_struct_roletag_NS_tl ...
    ..... 79, 1169, 1179, 1217
l__tag_struct_roletag_tl .....
    ..... 76, 1168, 1171, 1179, 1181, 1217
__tag_struct_set_attribute: ...
    ..... 23, 37, 1173, 1287
__tag_struct_set_tag_info:nnn .
    ..... 177, 179, 193, 212, 1150
g__tag_struct_stack_current_tl
    .. 16, 29, 31, 38, 69, 75, 121, 148,
    154, 162, 208, 270, 274, 310, 344,
    551, 556, 562, 1172, 1236, 1240,
    1241, 1262, 1280, 1286, 1325, 1331,
    1337, 1343, 1370, 1376, 1382, 1388
l__tag_struct_stack_parent_-
    tmpa_tl .. 16, 489, 498, 515, 760,
    1148, 1155, 1159, 1199, 1226, 1233,
    1237, 1239, 1242, 1254, 1255, 1263
g__tag_struct_stack_seq .....
    ..... 12, 22, 25, 488, 723,
    736, 1158, 1164, 1174, 1273, 1278, 1284
c__tag_struct_StructElem_-
    entries_seq ..... 39
c__tag_struct_StructTreeRoot_-
    entries_seq ..... 39
g__tag_struct_tag_NS_tl 76, 713,
    729, 732, 736, 1153, 1167, 1261, 1298
g__tag_struct_tag_stack_seq ...
    ..... 14, 50, 249,
    250, 555, 570, 584, 1170, 1277, 1292
g__tag_struct_tag_tl ..... 76,
    179, 180, 183, 314, 315, 401, 402,
    712, 714, 728, 731, 735, 737, 1152,
    1166, 1171, 1294, 1296, 1338, 1383
__tag_struct_use_check_parent_-
    child:nn . 652, 652, 699, 1342, 1387
__tag_struct_write_obj ..... 139
__tag_struct_write_obj:n .....
    ..... 151, 446, 446
l__tag_tag_stop_int 157, 161, 162,
    170, 171, 178, 185, 186, 195, 196, 204
g__tag_tagunmarked_bool 93, 249, 251
l__tag_tmp_unused_tl 62, 130, 297,
    304, 395, 398, 402, 405, 419, 422,
    703, 706, 719, 722, 770, 773, 788, 1569
l__tag_tmp_unused_tl_uuuu\l__-
    tag_Ref_tmpa_tl ..... 59
l__tag_tmpa_box .....
    ..... 59, 171, 177, 178, 182, 193, 194
l__tag_tmpa_clist .....
    ..... 59, 1521, 1522, 1555, 1556, 1558

```

\l__tag_tmpa_int	59, 90, 93, 98, 101, 105, 114, 430, 442, 444
\l__tag_tmpa_prop	59, 176, 189, 203, 205
\l__tag_tmpa_seq	51, 58, 59, 59, 337, 339, 341, 342, 343, 344, 443, 446, 448, 454, 455, 457, 458, 459, 468, 478, 705, 709, 712, 713, 721, 725, 728, 729, 772, 774, 775, 776, 776, 779, 780, 1523, 1527, 1537, 1538, 1539, 1541, 1559, 1565, 1567, 1591
\l__tag_tmpa_str	42, 43, 48, 59, 262, 267, 272, 297, 302, 309, 399, 404, 416, 421, 798, 805, 810, 817, 819, 820, 824, 825, 831, 840, 847, 855, 862, 928, 935
\l__tag_tmpa_tl	42, 43, 47, 49, 50, 51, 56, 59, 86, 88, 93, 94, 96, 98, 102, 106, 106, 108, 109, 113, 114, 116, 118, 119, 137, 138, 139, 141, 143, 144, 146, 177, 178, 180, 183, 184, 186, 191, 198, 199, 205, 205, 206, 209, 211, 214, 215, 220, 268, 269, 271, 275, 277, 288, 297, 299, 300, 302, 306, 308, 308, 314, 325, 326, 327, 329, 339, 358, 365, 367, 437, 445, 455, 456, 457, 458, 466, 468, 469, 470, 471, 475, 481, 483, 488, 514, 516, 525, 546, 549, 556, 564, 566, 575, 584, 588, 592, 599, 601, 604, 606, 620, 624, 645, 653, 655, 656, 658, 662, 667, 698, 702, 723, 725, 733, 735, 736, 738, 771, 772, 774, 784, 786, 950, 953, 1277, 1278, 1284, 1286, 1292, 1295, 1296, 1298, 1365, 1459, 1461, 1462, 1466, 1529, 1535, 1546, 1573
\l__tag_tmpb_box	59, 172, 179, 180, 184, 186
\l__tag_tmpb_seq	59, 1522, 1523, 1558, 1559
\l__tag_tmpb_tl	199, 59, 89, 104, 118, 120, 295, 301, 450, 456, 462, 484, 490, 518, 547, 550, 556, 568, 609, 611, 614, 616, 621, 625, 672, 680, 682, 683, 685, 689, 694, 699, 703, 734, 736, 785, 787, 883, 887, 896, 900
\l__tag_tmpc_tl	59, 485, 491
__tag_tree_fill_parenttree:	171, 172, 253
__tag_tree_final_checks:	20, 20, 354
\g__tag_tree_id_pad_int	78, 82, 172
__tag_tree_lua_fill_parenttree:	233, 233, 250
\g__tag_tree_openaction_struct-	32, 38, 57
__tag_tree_parenttree_rerun-	171, 220, 255
__tag_tree_update_openaction:	42, 75
__tag_tree_write_classmap:	286, 286, 369
__tag_tree_write_idtree:	86, 361
__tag_tree_write_namespaces:	322, 322, 373
__tag_tree_write_parenttree:	246, 246, 357
__tag_tree_write_rolemap:	263, 263, 365
__tag_tree_write_structelements:	147, 147, 377
__tag_tree_write_structtreeroot:	126, 126, 381
\g__tag_unique_cnt_int	95, 1082, 1086, 1089, 1099, 1103, 1107
__tag_whatsits:	36, 43, 48, 49, 52, 295, 296
tag-namespace_(rolemap-key)	752
tag/check/parent-child	183
tag/check/parent-child-end	183
tag/struct/1 internal commands:	
__tag/struct/1	31
tag/tree/namespaces internal commands:	
__tag/tree/namespaces	321
tag/tree/parenttree internal commands:	
__tag/tree/parenttree	154
tag/tree/rolemap internal commands:	
__tag/tree/rolemap	262
tagabspage	8, 121
tagmcabs	8, 121
\tagmcbegin	41, 185, 22
\tagmcend	41, 22
tagmcid	8, 121
\tagmcifinTF	41, 39
\tagmcuse	41, 22
\tagpdfparaOff	43, 563
\tagpdfparaOn	43, 563
\tagpdfsetup	41, 68, 115, 116, 185, 6
\tagpdfsuppressmarks	43, 568
\tagstart	7, 181, 212
\tagstop	7, 180, 211
tagstruct	8, 121
\tagstructbegin	41, 152, 185, 45, 288
\tagstructend	41, 45, 289
tagstructobj	8, 121
\tagstructuse	41, 45
\tagtool	41, 13
tagunmarked (deprecated) (key)	1, 249

test/lang _U (setup-key)	570	\tl_if_empty_p:n	310, 750
TeX and L ^A T _E X 2 _ε commands:		\tl_if_eq:NNTF	367, 483, 685
\@M	168	\tl_if_eq:NnTF	108
\@bsphack	109	\tl_if_eq:nnTF	212, 274, 278
\@esphack	111	\tl_if_exist:NTF	259, 338, 389, 971
\@gobble	31, 55	\tl_if_head_eq_charcode:nNTF	49
\@ifpackageloaded	22	\tl_if_in:nnTF	185
\@kernel@after@foot	585	\tl_new:N	11, 12, 12, 13, 14, 15, 16, 17, 19, 20, 22, 23, 24, 25, 26, 32, 33, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 76, 77, 78, 79, 80, 82, 86, 162, 170, 285, 330, 332, 334, 336, 339, 340, 500, 981, 1111, 1496
\@kernel@after@head	583	\tl_put_left:Nn	583, 585
\@kernel@before@foot	584	\tl_put_right:Nn	94, 104, 118, 195, 207, 226, 251, 256, 266, 267, 281, 297, 301, 302, 394, 403, 404, 409, 420, 421, 423, 432, 436, 441, 582, 584, 875, 885, 898, 909, 1426, 1582, 1589
\@kernel@before@head	580, 582	\tl_remove_once:Nn	1461, 1462
\@maxdepth	181	\tl_replace_once:Nnn	326
\@secondoftwo	31, 55	\tl_set:Nn	42, 81, 83, 86, 87, 118, 139, 160, 162, 180, 183, 189, 193, 197, 201, 205, 209, 213, 217, 224, 224, 225, 235, 238, 239, 242, 243, 244, 249, 250, 271, 275, 302, 306, 309, 316, 333, 335, 337, 365, 389, 391, 401, 437, 508, 516, 518, 552, 560, 566, 568, 601, 606, 611, 616, 629, 645, 655, 658, 662, 667, 672, 682, 685, 689, 694, 707, 760, 775, 776, 779, 780, 786, 787, 790, 790, 794, 1148, 1322, 1430, 1535, 1563
\c@chapter	360, 378	\tl_set_eq:NN	179, 314
\on@line	475, 490, 506	\tl_show:N	1236, 1237, 1587, 1593
tex commands:		\tl_tail:n	549
\tex_botmarks:D	91	\tl_to_str:n	33, 48, 149, 202, 217, 506, 539
\tex_firstmarks:D	88	\tl_trim_spaces:n	49
\tex_kern:D	184	\tl_use:N	261, 959, 1002, 1021, 1066
\tex_marks:D	25, 34, 47, 54, 65, 71	tree-mcid-index-wrong	21, 220
\tex_special:D	52	tree-statistic	20, 54
\tex_splitbotmarks:D	217	tree-struct-still-open	20, 47
\tex_splitfirstmarks:D	197		
texsource (key)	1, 938		
\tiny	429, 440		
title (key)	1, 740		
title-o (key)	1, 740		
tl commands:			
\c_empty_tl	365, 385		
\c_space_tl	55, 56, 58, 60, 98, 100, 104, 116, 167, 191, 197, 198, 216, 236, 238, 240, 259, 299, 406, 423, 443, 471, 877, 887, 900, 911, 978, 1254, 1337, 1382, 1466, 1538, 1584		
\tl_clear:N	88, 89, 106, 177, 228, 229, 288, 415		
\tl_const:Nn	10		
\tl_count:n	79, 83, 172		
\tl_gput_left:Nn	972		
\tl_gput_right:Nn	165, 976		
\tl_gset:Nn	18, 33, 38, 121, 225, 243, 286, 298, 331, 373, 385, 390, 712, 713, 728, 729, 735, 736, 983, 1172, 1286, 1294, 1298		
\tl_gset_eq:NN	180, 315		
\tl_head:N	655, 682		
\tl_if_empty:NTF	43, 43, 109, 185, 198, 289, 307, 316, 335, 399, 656, 683, 772, 778, 820, 1136		
\tl_if_empty:nTF	51, 69, 77, 89, 142, 196, 210, 259, 262, 266, 279, 294, 295, 297, 334, 353, 413, 440, 603, 611, 643, 670, 821, 837, 852, 943, 988, 1013		

U

uncompress (deprecated) (key)	236
unitag _U (deprecated)	394
\unskip	41
use commands:	
\use:N	67, 229, 569, 1238
\use:n	41, 366

\use_i:nn	\UseTaggingSocket	44 , 45 , 69 , 72
. 99, 102, 109, 138, 154, 159, 224,		
238, 365, 385, 588, 592, 615, 626, 1295		
\use_ii:nn	V	
104, 119,	\vbadness	168, 192
135, 152, 157, 225, 239, 344, 612, 623	vbox commands:	
\use_none:n	\vbox_set_split_to_ht:NNn	194
81, 103, 118, 224	\vbox_set_to_ht:Nnn	170
\use_none:nn	\vbox_unpack_drop:N	183
80, 1411	\vfuzz	169
\UseExpandableTaggingSocket .	viewer/startstructure_␣(setup-key) ..	34
44 , 70 , 72		
\UseSocket		44