

tagpdf – L^AT_EX kernel code for PDF tagging*

Ulrike Fischer[†]

Released 2025-01-12

Contents

I	The tagpdf main module	
	Part of the tagpdf package	7
1	Initialization and test if pdfmanagement is active.	8
2	base package	9
3	Package options	9
4	Packages	9
	4.1 Patches related to Ref improvement	10
	4.2 a LastPage label	10
5	Variables	11
6	Variants of l3 commands	12
7	Label and Reference commands	12
8	Setup label attributes	13
9	Commands to fill seq and prop	13
10	General tagging commands	14
11	Keys for tagpdfsetup	15
12	loading of engine/more dependent code	17
II	The tagpdf-checks module	
	Messages and check code	
	Part of the tagpdf package	18

*This file describes v0.99l, last revised 2025-01-12.

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

1	Commands	18
2	Description of log messages	18
2.1	\ShowTagging command	18
2.2	Messages in checks and commands	19
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	22
3.1	Messages related to mc-chunks	22
3.2	Messages related to structures	23
3.3	Attributes	25
3.4	Roles	25
3.5	Miscellaneous	26
4	Retrieving data	26
5	User conditionals	26
6	Internal checks	27
6.1	checks for active tagging	27
6.2	Checks related to structures	28
6.3	Checks related to roles	29
6.4	Check related to mc-chunks	30
6.5	Checks related to the state of MC on a page or in a split stream	33
6.6	Benchmarks	36
III The tagpdf-user module		
Code related to L^AT_EX₂ε user commands and document commands		
Part of the tagpdf package		37
1	Setup commands	37
2	Commands related to mc-chunks	37
3	Commands related to structures	38
4	Debugging	38
5	Extension commands	39
5.1	Fake space	39
5.2	Tagging of paragraphs	39
5.3	Header and footer	40
5.4	Link tagging	40
6	Socket support	40

7	User commands and extensions of document commands	41
8	Setup and preamble commands	41
9	Commands for the mc-chunks	42
10	Commands for the structure	42
11	Socket support	43
12	Debugging	44
13	Commands to extend document commands	48
	13.1 Document structure	48
	13.2 Structure destinations	49
	13.3 Fake space	49
	13.4 Paratagging	49
	13.5 Language support	57
	13.6 Header and footer	57
	13.7 Links	59

IV The tagpdf-tree module
Commands trees and main dictionaries
Part of the tagpdf package **61**

1	Trees, pdfmanagement and finalization code	61
	1.1 Check structure	61
	1.2 Catalog: MarkInfo and StructTreeRoot and OpenAction	62
	1.3 Writing the IDtree	63
	1.4 Writing structure elements	64
	1.5 ParentTree	65
	1.6 Rolemap dictionary	68
	1.7 Classmap dictionary	68
	1.8 Namespaces	69
	1.9 Finishing the structure	70
	1.10 StructParents entry for Page	71

V The tagpdf-mc-shared module
Code related to Marked Content (mc-chunks), code shared by
all modes
Part of the tagpdf package **72**

1	Public Commands	72
2	Public keys	73

3	Marked content code – shared	74
3.1	Variables and counters	74
3.2	Functions	75
3.3	Keys	78
VI	The tagpdf-mc-generic module	
	Code related to Marked Content (mc-chunks), generic mode	
	Part of the tagpdf package	80
1	Marked content code – generic mode	80
1.1	Variables	80
1.2	Functions	81
1.3	Looking at MC marks in boxes	84
1.4	Keys	92
VII	The tagpdf-mc-luacode module	
	Code related to Marked Content (mc-chunks), luamode-specific	
	Part of the tagpdf package	94
1	Marked content code – luamode code	94
1.1	Commands	96
1.2	Key definitions	100
VIII	The tagpdf-struct module	
	Commands to create the structure	
	Part of the tagpdf package	103
1	Public Commands	103
2	Public keys	104
2.1	Keys for the structure commands	104
2.2	Setup keys	106
3	Variables	106
3.1	Variables used by the keys	108
3.2	Variables used by tagging code of basic elements	109
4	Commands	109
4.1	Initialization of the StructTreeRoot	110
4.2	Adding the /ID key	111
4.3	Filling in the tag info	112
4.4	Handlings kids	113
4.5	Output of the object	117
5	Keys	121
6	User commands	127

7	Attributes and attribute classes	136
7.1	Variables	136
7.2	Commands and keys	137
IX	The tagpdf-luatex.def	
	Driver for luatex	
	Part of the tagpdf package	140
1	Loading the lua	140
2	User commands to access data	144
3	Logging functions	145
4	Helper functions	147
4.1	Retrieve data functions	147
4.2	Functions to insert the pdf literals	149
5	Function for the real space chars	151
6	Function for the tagging	155
7	Parenttree	160
X	The tagpdf-roles module	
	Tags, roles and namespace code	
	Part of the tagpdf package	163
1	Code related to roles and structure names	163
1.1	Variables	164
1.2	Namespaces	166
1.3	Adding a new tag	167
1.3.1	pdf 1.7 and earlier	168
1.3.2	The pdf 2.0 version	170
1.4	Helper command to read the data from files	172
1.5	Reading the default data	174
1.6	Parent-child rules	175
1.6.1	Reading in the csv-files	175
1.6.2	Retrieving the parent-child rule	177
1.7	Remapping of tags	182
1.8	Key-val user interface	182
XI	The tagpdf-space module	
	Code related to real space chars	
	Part of the tagpdf package	185
1	Code for interword spaces	185

Part I

The tagpdf main module

Part of the tagpdf package

<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, e.g. for debugging.

`activate/tree` (*setup key*) This key activates the code that finalize the various trees. It should be used only in 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. 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 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) or `none`. Currently this is set more or less globally. More finer control can be added if needed.

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

These are attributes used by the label/ref system.

1 Initialization and test if pdfmanagement is active.

```

1 <@=tag>
2 <*package>
3 \ProvidesExplPackage {tagpdf} {2025-01-12} {0.991}
4   { LaTeX kernel code for PDF tagging }
5
6 \bool_if:nF
7   {
8     \bool_lazy_and_p:nn
9       {\cs_if_exist_p:N \pdfmanagement_if_active_p:}
10      { \pdfmanagement_if_active_p: }
11   }
12 { %error for now, perhaps warning later.
13   \PackageError{tagpdf}
14     {
15       PDF-resource-management~is~no~active!\MessageBreak
16       tagpdf~will~no~work.
17     }
18     {
19       Activate-it~with \MessageBreak
20       \string\RequirePackage{pdfmanagement-testphase}\MessageBreak
21       \string\DocumentMetadata{<options>}\MessageBreak
22       before~\string\documentclass
23     }
24   }
25 </package>
26 <*debug>
27 \ProvidesExplPackage {tagpdf-debug} {2025-01-12} {0.991}
28   { debug code for tagpdf }
29 \@ifpackageloaded{tagpdf}{\PackageWarning{tagpdf-debug}{tagpdf~not~loaded,~quitting}\ending
30 </debug> We map the internal module name “tag” to “tagpdf” in messages.
31 <*package>
32 \prop_gput:Nnn \g_msg_module_name_prop { tag }{ tagpdf }
33 </package>
34 Debug mode has its special mapping:
35 <*debug>
36 \prop_gput:Nnn \g_msg_module_type_prop { tag / debug} {}
37 \prop_gput:Nnn \g_msg_module_name_prop { tag / debug }{tagpdf~DEBUG}
38 </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} {2025-01-12} {0.991}
38   {part of tagpdf - provide base, no-op versions of the user commands }
39 </base>
```

3 Package options

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

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

4 Packages

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

```
54 \RequirePackage{tagpdf-base}
55 </package>
```

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

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

4.1 Patches related to Ref improvement

2024-09-09: Temporary code. Can be removed when the latex-lab-footnote and latex-lab-toc code have been adapted to the better Ref handling.

```
70 <*package>
71 \AddToHook{package/latex-lab-testphase-new-or-2/after}
72 {
73   \cs_set_protected:Npn \__fnote_gput_ref:nn #1 #2 %#1 the structure number receiving the r
74   {
75     \tag_struct_gput:nnn {#1}{ref_num}{#2}
76   }
77 }
78 \AddToHook{package/latex-lab-testphase-toc/after}
79 {
80   \cs_set_protected:Npn \g__tag_struct_ref_by_dest:
81   {
82     \prop_map_inline:Nn\g__tag_struct_ref_by_dest_prop
83     {
84       \tag_struct_gput:nnn {##1}{ref_dest}{##2}
85     }
86   }
87 }
88 </package>
```

4.2 a LastPage label

See also issue #2 in Accessible-xref

__tag_lastpagelabel:

```
89 <*package>
90 \cs_new_protected:Npn \__tag_lastpagelabel:
91 {
92   \legacy_if:nT { @filesw }
93   {
94     \exp_args:NNne \exp_args:NNe\iow_now:Nn \@auxout
95     {
96       \token_to_str:N \new@label@record
97       { @tag@LastPage }
98       {
99         { abspage } { \int_use:N \g_shipout_readonly_int }
100        { tagmcabs } { \int_use:N \c@g__tag_MCID_abs_int }
101        { tagstruct } { \int_use:N \c@g__tag_struct_abs_int }
102      }
103    }
104  }
105 }
106
107 \AddToHook{enddocument/afterlastpage}
108 { \__tag_lastpagelabel: }
(End of definition for \__tag_lastpagelabel:.)
```

5 Variables

```

\l__tag_tmpa_tl A few temporary variables
\l__tag_tmpb_tl
\l__tag_Ref_tmpa_tl
\l__tag_get_tmpc_tl
\l__tag_get_parent_tmpb_tl \l__tag_tmpa_str
\l__tag_tmpa_prop
\l__tag_tmpa_seq
\l__tag_tmpb_seq
\l__tag_tmpa_clist
\l__tag_tmpa_int
\l__tag_tmpa_box
\l__tag_tmpb_box
109 \tl_new:N \l__tag_tmpa_tl
110 \tl_new:N \l__tag_tmpb_tl
111 \tl_new:N \l__tag_Ref_tmpa_tl
112 \tl_new:N \l__tag_get_tmpc_tl
113 \tl_new:N \l__tag_get_parent_tmpa_tl
114 \tl_new:N \l__tag_get_parent_tmpb_tl
115 \str_new:N \l__tag_tmpa_str
116 \prop_new:N \l__tag_tmpa_prop
117 \seq_new:N \l__tag_tmpa_seq
118 \seq_new:N \l__tag_tmpb_seq
119 \clist_new:N \l__tag_tmpa_clist
120 \int_new:N \l__tag_tmpa_int
121 \box_new:N \l__tag_tmpa_box
122 \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
123 \clist_const:Nn \c__tag_property_mc_clist {tagabspace,tagmcabs,tagmcid}
124 \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.

```

125 \int_new:N \l__tag_loglevel_int

```

(End of definition for \l__tag_loglevel_int.)

```

\g__tag_active_space_bool
\g__tag_active_mc_bool
\g__tag_active_tree_bool
\g__tag_active_struct_bool
\g__tag_active_struct_dest_bool

```

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

```

126 \bool_new:N \g__tag_active_space_bool
127 \bool_new:N \g__tag_active_mc_bool
128 \bool_new:N \g__tag_active_tree_bool
129 \bool_new:N \g__tag_active_struct_bool
130 \bool_new:N \g__tag_active_struct_dest_bool
131 \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` TODO: check if they are used everywhere as needed and as wanted.

`\l__tag_active_socket_bool`

```

132 \bool_new:N \l__tag_active_mc_bool
133 \bool_set_true:N \l__tag_active_mc_bool
134 \bool_new:N \l__tag_active_struct_bool
135 \bool_set_true:N \l__tag_active_struct_bool
136 \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 empty artifact mc-chunks.

```

137 \bool_new:N \g__tag_tagunmarked_bool

```

(End of definition for `\g__tag_tagunmarked_bool`.)

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

```

138 \bool_new:N \g__tag_softhyphen_bool

```

(End of definition for `\g__tag_softhyphen_bool`.)

6 Variants of l3 commands

```

139 \prg_generate_conditional_variant:Nnn \pdf_object_if_exist:n {e}{T,F,TF}
140 \cs_generate_variant:Nn \pdf_object_ref:n {e}
141 \cs_generate_variant:Nn \pdfannot_dict_put:nnn {nne}
142 \cs_generate_variant:Nn \pdffile_embed_stream:nnn {nee,oee}
143 \cs_generate_variant:Nn \prop_gput:Nnn {Nee,Nen} %** unneeded
144 \cs_generate_variant:Nn \prop_put:Nnn {Nee} %** unneeded
145 \cs_generate_variant:Nn \prop_item:Nn {No,Ne} %** unneeded
146 \cs_generate_variant:Nn \seq_set_split:Nnn{Nne} %** unneeded
147 \cs_generate_variant:Nn \str_set_convert:Nnnn {Nonn, Noon, Nnon }
148 \cs_generate_variant:Nn \clist_map_inline:nn {on}

```

7 Label and Reference commands

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

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

```

149 \cs_new_protected:Npn \__tag_property_record:nn #1#2
150 {
151   \@bsphack
152   \property_record:nn{#1}{#2}
153   \@esphack
154 }

```

155

And a few variants

```

156 \cs_generate_variant:Nn \property_ref:nnn {enn}
157 \cs_generate_variant:Nn \property_ref:nn {en}
158 \cs_generate_variant:Nn \__tag_property_record:nn {en,eV}

```

(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.

```

159 \cs_new:Npn \__tag_property_ref_lastpage:nn #1 #2
160 {
161   \property_ref:nnn {@tag@LastPage}{#1}{#2}
162 }

```

(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`.

```

163 \property_new:nnnn
164 { tagstruct } { now }
165 {1} { \int_use:N \c@g__tag_struct_abs_int }
166 \property_new:nnnn { tagstructobj } { now } {}
167 {
168   \pdf_object_ref_indexed:nn { __tag/struct } { \c@g__tag_struct_abs_int }
169 }
170 \property_new:nnnn
171 { tagabspage } { shipout }
172 {0} { \int_use:N \g_shipout_readonly_int }
173 \property_new:nnnn { tagmcabs } { now }
174 {0} { \int_use:N \c@g__tag_MCID_abs_int }
175
176 \flag_new:n { __tag/mcid }
177 \property_new:nnnn {tagmcid} { shipout }
178 {0} { \flag_height:n { __tag/mcid } }
179

```

(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 180 \cs_set_eq:NN \_tag_prop_new:N \prop_new:N
    \_tag_seq_new:N      181 \cs_set_eq:NN \_tag_prop_new_linked:N \prop_new_linked:N
    \_tag_prop_gput:Nnn  182 \cs_set_eq:NN \_tag_seq_new:N \seq_new:N
\_tag_seq_gput_right:Nn 183 \cs_set_eq:NN \_tag_prop_gput:Nnn \prop_gput:Nnn
    \_tag_seq_item:cn    184 \cs_set_eq:NN \_tag_seq_gput_right:Nn \seq_gput_right:Nn
    \_tag_prop_item:cn   185 \cs_set_eq:NN \_tag_seq_gput_left:Nn \seq_gput_left:Nn
    \_tag_seq_show:N     186 \cs_set_eq:NN \_tag_seq_item:cn \seq_item:cn
    \_tag_prop_show:N    187 \cs_set_eq:NN \_tag_prop_item:cn \prop_item:cn
    \_tag_prop_show:N    188 \cs_set_eq:NN \_tag_seq_show:N \seq_show:N
    \_tag_prop_show:N    189 \cs_set_eq:NN \_tag_prop_show:N \prop_show:N
190 % cnx temporary needed for latex-lab-graphic code
191 \cs_generate_variant:Nn \_tag_prop_gput:Nnn { Nen , Nee, Nne , cmn, cen, cne, cno, cnx}
192 \cs_generate_variant:Nn \_tag_seq_gput_right:Nn { Ne , No, cn, ce }
193 \cs_generate_variant:Nn \_tag_seq_gput_left:Nn { ce }
194 \cs_generate_variant:Nn \_tag_prop_new:N { c }
195 \cs_generate_variant:Nn \_tag_seq_new:N { c }
196 \cs_generate_variant:Nn \_tag_seq_show:N { c }
197 \cs_generate_variant:Nn \_tag_prop_show:N { c }
198 \end{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.

```

199 \*package | debug)
200 \package\int_new:N \l__tag_tag_stop_int
\l__tag_tag_stop_int
201 \cs_set_protected:Npn \tag_stop:
202 {
203 \debug \msg_note:nne {tag / debug }{tag-suspend}{ \int_use:N \l__tag_tag_stop_int }
204 \int_incr:N \l__tag_tag_stop_int
205 \bool_set_false:N \l__tag_active_struct_bool
206 \bool_set_false:N \l__tag_active_mc_bool
207 \bool_set_false:N \l__tag_active_socket_bool
208 \_tag_stop_para_ints:
209 }
210 \cs_set_protected:Npn \tag_start:
211 {
212 \int_if_zero:nF { \l__tag_tag_stop_int } { \int_decr:N \l__tag_tag_stop_int }
213 \int_if_zero:nT { \l__tag_tag_stop_int }
214 {

```

```

215         \bool_set_true:N \l__tag_active_struct_bool
216         \bool_set_true:N \l__tag_active_mc_bool
217         \bool_set_true:N \l__tag_active_socket_bool
218         \__tag_start_para_ints:
219     }
220 <debug>     \msg_note:nne {tag / debug }{tag-resume}{ \int_use:N \l__tag_tag_stop_int }
221 }
222 \cs_set_eq:NN\tagstop\tag_stop:
223 \cs_set_eq:NN\tagstart\tag_start:
224 \cs_set_protected:Npn \tag_suspend:n #1
225 {
226 <debug>     \msg_note:nnee {tag / debug }{tag-suspend}
227 <debug>     { \int_use:N \l__tag_tag_stop_int }{\exp_not:n{#1}}
228     \int_incr:N \l__tag_tag_stop_int
229     \bool_set_false:N \l__tag_active_struct_bool
230     \bool_set_false:N \l__tag_active_mc_bool
231     \bool_set_false:N \l__tag_active_socket_bool
232     \__tag_stop_para_ints:
233 }
234 \cs_set_eq:NN \tag_stop:n \tag_suspend:n
235 \cs_set_protected:Npn \tag_resume:n #1
236 {
237     \int_if_zero:nF { \l__tag_tag_stop_int } { \int_decr:N \l__tag_tag_stop_int }
238     \int_if_zero:nT { \l__tag_tag_stop_int }
239     {
240         \bool_set_true:N \l__tag_active_struct_bool
241         \bool_set_true:N \l__tag_active_mc_bool
242         \bool_set_true:N \l__tag_active_socket_bool
243         \__tag_start_para_ints:
244     }
245 <debug>     \msg_note:nnee {tag / debug }{tag-resume}
246 <debug>     { \int_use:N \l__tag_tag_stop_int }{\exp_not:n{#1}}
247 }
248 \cs_set_eq:NN \tag_start:n \tag_resume:n
249 </package | debug>
250 <*base>
251 \cs_new_protected:Npn \tag_stop:{}
252 \cs_new_protected:Npn \tag_start:{}
253 \cs_new_protected:Npn \tagstop{}
254 \cs_new_protected:Npn \tagstart{}
255 \cs_new_protected:Npn \tag_stop:n #1 {}
256 \cs_new_protected:Npn \tag_start:n #1 {}

```

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

```

257 \cs_set_eq:NN \tag_suspend:n \tag_stop:n
258 \cs_set_eq:NN \tag_resume:n \tag_start:n
259 </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) 260 <{*package}
activate/struct-dest (setup key) 261 \keys_define:nn { __tag / setup }
262 {
263   activate/mc      .bool_gset:N = \g__tag_active_mc_bool,
264   activate/tree    .bool_gset:N = \g__tag_active_tree_bool,
265   activate/struct .bool_gset:N = \g__tag_active_struct_bool,
266   activate/all     .meta:n =
267     {activate/mc={#1},activate/tree={#1},activate/struct={#1}},
268   activate/all     .default:n = true,
269   activate/struct-dest .bool_gset:N = \g__tag_active_struct_dest_bool,

```

old, deprecated names

```

270   activate-mc      .bool_gset:N = \g__tag_active_mc_bool,
271   activate-tree    .bool_gset:N = \g__tag_active_tree_bool,
272   activate-struct .bool_gset:N = \g__tag_active_struct_bool,
273   activate-all     .meta:n =
274     {activate/mc={#1},activate/tree={#1},activate/struct={#1}},
275   activate-all     .default:n = true,
276   no-struct-dest .bool_gset:N = \g__tag_active_struct_dest_bool,

```

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

```

277   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) 278   debug/log      .choice:,
log (deprecated) (setup key) 279   debug/log / none .code:n = {\int_set:Nn \l__tag_loglevel_int { 0 }},
uncompress (deprecated) (setup key) 280   debug/log / v   .code:n =
281     {
282       \int_set:Nn \l__tag_loglevel_int { 1 }
283       \cs_set_protected:Nn \__tag_check_typeout_v:n { \iow_term:e {##1} }
284     },
285   debug/log / vv  .code:n = {\int_set:Nn \l__tag_loglevel_int { 2 }},
286   debug/log / vvv .code:n = {\int_set:Nn \l__tag_loglevel_int { 3 }},
287   debug/log / all .code:n = {\int_set:Nn \l__tag_loglevel_int { 10 }},
288   debug/uncompress .code:n = { \pdf_uncompress: },

```

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

```

289   log      .meta:n = {debug/log={#1}},
290   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) 291   activate/tagunmarked .bool_gset:N = \g__tag_tagunmarked_bool,
292   activate/tagunmarked .initial:n = true,

```

deprecated name

```

293   tagunmarked .bool_gset:N = \g__tag_tagunmarked_bool,

```

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

```

294   activate/softhyphen .bool_gset:N = \g__tag_softhyphen_bool,
295   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.

```

296   page/tabsorder      .choice:,
297   page/tabsorder / row      .code:n =
298     \pdfmanagement_add:nnn { Page } {Tabs}{/R},
299   page/tabsorder / column  .code:n =
300     \pdfmanagement_add:nnn { Page } {Tabs}{/C},
301   page/tabsorder / structure .code:n =
302     \pdfmanagement_add:nnn { Page } {Tabs}{/S},
303   page/tabsorder / none    .code:n =
304     \pdfmanagement_remove:nn {Page} {Tabs},
305   page/tabsorder      .initial:n = structure,

```

deprecated name

```

306   tabsorder .meta:n = {page/tabsorder={#1}},
307   }

```

12 loading of engine/more dependent code

```

308 \sys_if_engine luatex:T
309 {
310   \file_input:n {tagpdf-luatex.def}
311 }
312 </package>
313 <*mcloding>
314 \bool_if:NTF \g__tag_mode_lua_bool
315 {
316   \RequirePackage {tagpdf-mc-code-lua}
317 }
318 {
319   \RequirePackage {tagpdf-mc-code-generic} %
320 }
321 </mcloding>
322 <*debug>
323 \bool_if:NTF \g__tag_mode_lua_bool
324 {
325   \RequirePackage {tagpdf-debug-lua}
326 }
327 {
328   \RequirePackage {tagpdf-debug-generic} %
329 }
330 </debug>

```

Part II

The tagpdf-checks module

Messages and check code

Part of the tagpdf package

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` $\langle keyword \rangle$

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

`\tag_if_box_tagged_p:N` * `\tag_if_box_tagged:NTF` $\langle box \rangle$ $\langle true\ code \rangle$ $\langle false\ code \rangle$

`\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_t1` 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
\@@_check_structure_has_tag:n	struct-missing-tag	error
\@@_check_structure_tag:N	role-unknown-tag	warning
\@@_check_info_closing_struct:n	struct-show-closing	info
\@@_check_no_open_struct:	struct-faulty-nesting	error
\@@_check_struct_used:n	struct-used-twice	warning
\@@_check_add_tag_role:nn	role-missing, role-tag, role-unknown	warning, info (>0), warning
\@@_check_mc_if_nested:,	mc-nested	warning
\@@_check_mc_if_open:	mc-not-open	warning
\@@_check_mc_pushed_popped:nn	mc-pushed, mc-popped	info (2), info+seq_log (>2)
\@@_check_mc_tag:N	mc-tag-missing, role-unknown-tag	error (missing), warning (unknown).
\@@_check_mc_used:n	mc-used-twice	warning
\@@_check_show_MCID_by_page:		
\tag_mc_use:n	mc-label-unknown, mc-used-twice	warning
\role_add_tag:nn	new-tag	info (>0)
	sys-no-interwordspace	warning
\@@_struct_write_obj:n	struct-no-objnum	error
\@@_struct_write_obj:n	struct-orphan	warning
\tag_struct_begin:n	struct-faulty-nesting	error
\@@_struct_insert_annot:nn	struct-faulty-nesting	error
tag_struct_use:n	struct-label-unknown	warning
attribute-class, attribute	attr-unknown	error
\@@_tree_fill_parenttree:	tree-mcid-index-wrong	warning TODO: should trigger a standard rerun m
in enddocument/info-hook	para-hook-count-wrong	error (warning?)

2.3 Messages from the ptagging code

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

2.4 Warning messages from the lua-code

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

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

2.5 Info messages from the lua-code

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

message	log-level	remark
INFO SHIPOUT-INSERT-LAST-EMC	3	finish of shipout code
INFO SPACE-FUNCTION-FONT	3	interwordspace code
INFO TAG-ABSPAGE	3	
INFO TAG-ARGS	4	
INFO TAG-ENDHEAD	4	
INFO TAG-ENDHEAD	4	
INFO TAG-HEAD	3	
INFO TAG-INSERT-ARTIFACT	3	

message	log-level	remark
INFO TAG-INSERT-BDC	3	
INFO TAG-INSERT-EMC	3	
INFO TAG-INSERT-TAG	3	
INFO TAG-KERN-SUBTYPE	4	
INFO TAG-MATH-SUBTYPE	4	
INFO TAG-MC-COMPARE	4	
INFO TAG-MC-INTO-PAGE	3	
INFO TAG-NEW-MC-NODE	4	
INFO TAG-NODE	3	
INFO TAG-NO-HEAD	3	
INFO TAG-NOT-TAGGED	2	replaced by artifact
INFO TAG-QUITTING-BOX	4	
INFO TAG-STORE-MC-KID	4	
INFO TAG-TRAVERSING-BOX	3	
INFO TAG-USE-ACTUALTEXT	3	
INFO TAG-USE-ALT	3	
INFO TAG-USE-RAW	3	
INFO TEX-MC-INSERT-KID	3	
INFO TEX-MC-INSERT-KID-TEST	4	
INFO TEX-MC-INTO-STRUCT	3	
INFO TEX-STORE-MC-DATA	3	
INFO TEX-STORE-MC-KID	3	
INFO PARENTTREE-CHUNKS	3	
INFO PARENTTREE-NO-DATA	3	
INFO PARENTTREE-NUM	3	
INFO PARENTTREE-NUMENTRY	3	
INFO PARENTTREE-STRUCT-OBJREF	4	

2.6 Debug mode messages and code

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

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

2.7 Messages

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

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

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

`tree-statistic` Message issued at the end of the compilation showing the number of objects to write

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

`attr-unknown` Message if an attribute is unknown.

`role-missing` Messages related to role mapping.
`role-unknown`
`role-unknown-tag`
`role-unknown-NS`
`role-tag`
`new-tag`
`role-parent-child`
`role-remapping`

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

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

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

```
1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-checks-code} {2025-01-12} {0.991}
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} { required~tag~missing~--~mcid~#1 }
```

(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 21.)

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 21.)

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 21.)

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 21.)

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 21.)

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 21.)

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 21.)

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 21.)

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 21.)

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 21.)

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      86 \msg_new:nnn { tag } {role-missing}      { tag-#1-has-no-role-assigned }
role-unknown-tag  87 \msg_new:nnn { tag } {role-unknown}      { role-#1-is-not-known }
role-unknown-NS   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` This is info and warning message about the containment rules between child and parent tags.

```

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

```

(End of definition for `role-parent-child`. This function is documented on page 21.)

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

```

92 \msg_new:nnn { tag } {role-remapping}
93   { remapping-tag-to-#1 }

```

(End of definition for `role-remapping`. This function is documented on page 21.)

`role-tag` Info messages.

```

new-tag          94 \msg_new:nnn { tag } {role-tag}          { mapping-tag-#1-to-role-#2 }
                95 \msg_new:nnn { tag } {new-tag}          { adding-new-tag-#1 }
                96 \msg_new:nnn { tag } {read-namespace} { reading-namespace-definitions-tagpdf-
                  ns-#1.def }
                97 \msg_new:nnn { tag } {namespace-missing}{ namespace-definitions-tagpdf-ns-#1.def-not-found }
                98 \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.

```
99 \msg_new:nnn { tag } {tree-mcid-index-wrong}
100   {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 lualatex have some support for real spaces.

```
101 \msg_new:nnn { tag } {sys-no-interwordspace}
102   {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.

```
103 \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.

```
104 \msg_new:nnnn { tag } {para-hook-count-wrong}
105   {The~number~of~automatic~begin~(#1)~and~end~(#2)~#3~para~hooks~differ!}
106   {This~quite~probably~a~coding~error~and~the~structure~will~be~wrong!}
107 \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`.

```
108 <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

```
109 <*base>
110 \cs_if_exist:Nf\tag_if_active:T
111 {
112   \prg_new_conditional:Npnn \tag_if_active: { p , T , TF , F }
113     { \prg_return_false: }
114 }
115 </base>
116 <*package>
117 \prg_set_conditional:Npnn \tag_if_active: { p , T , TF , F }
118 {
119   \bool_lazy_all:nTF
120   {
```

```

121     {\g__tag_active_struct_bool}
122     {\g__tag_active_mc_bool}
123     {\g__tag_active_tree_bool}
124     {\l__tag_active_struct_bool}
125     {\l__tag_active_mc_bool}
126   }
127   {
128     \prg_return_true:
129   }
130   {
131     \prg_return_false:
132   }
133 }
134 \endpackage

```

(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.

```

135 \begin{base}
136 \prg_new_conditional:Npnn \tag_if_box_tagged:N #1 {p,T,F,TF}
137 {
138   \tl_if_exist:cTF {l_tag_box_\int_use:N #1_tl}
139   {
140     \int_compare:nNnTF {0\tl_use:c{l_tag_box_\int_use:N #1_tl}}>{0}
141     { \prg_return_true: }
142     { \prg_return_false: }
143   }
144   {
145     \prg_return_false:
146     % warning??
147   }
148 }
149 \end{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
150 \begin{package}
151 \prg_new_conditional:Npnn \__tag_check_if_active_mc: {T,F,TF}
152 {
153   \bool_lazy_and:nnTF { \g__tag_active_mc_bool } { \l__tag_active_mc_bool }
154   {
155     \prg_return_true:

```

```

156     }
157     {
158         \prg_return_false:
159     }
160 }
161 \prg_new_conditional:Npnn \__tag_check_if_active_struct: {T,F,TF}
162 {
163     \bool_lazy_and:nnTF { \g__tag_active_struct_bool } { \l__tag_active_struct_bool }
164     {
165         \prg_return_true:
166     }
167     {
168         \prg_return_false:
169     }
170 }

```

(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.

```

171 \cs_new_protected:Npn \__tag_check_structure_has_tag:n #1 %#1 struct num
172 {
173     \prop_if_in:cnF { g__tag_struct_#1_prop }
174     {S}
175     {
176         \msg_error:nn { tag } {struct-missing-tag}
177     }
178 }

```

(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.

```

179 \cs_new_protected:Npn \__tag_check_structure_tag:N #1
180 {
181     \prop_if_in:NoF \g__tag_role_tags_NS_prop {#1}
182     {
183         \msg_warning:nne { tag } {role-unknown-tag} {#1}
184     }
185 }

```

(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.

```

186 \cs_new_protected:Npn \__tag_check_info_closing_struct:n #1 %#1 struct num
187 {
188     \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
189     {
190         \msg_info:nnn { tag } {struct-show-closing} {#1}
191     }
192 }

```

```

193
194 \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.

```

195 \cs_new_protected:Npn \__tag_check_no_open_struct:
196 {
197   \msg_error:nn { tag } {struct-faulty-nesting}
198 }

```

(End of definition for __tag_check_no_open_struct:.)

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

```

199 \cs_new_protected:Npn \__tag_check_struct_used:n #1 %#1 label
200 {
201   \prop_get:cnNT
202     {g__tag_struct\_property_ref:enn{tagpdfstruct-#1}{tagstruct}{unknown}_prop}
203     {P}
204     \l__tag_tmpa_tl
205     {
206       \msg_warning:nnn { tag } {struct-used-twice} {#1}
207     }
208 }

```

(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.

```

209 \cs_new_protected:Npn \__tag_check_add_tag_role:nn #1 #2 %#1 tag, #2 role
210 {
211   \tl_if_empty:nTF {#2}
212   {
213     \msg_error:nnn { tag } {role-missing} {#1}
214   }
215   {
216     \prop_get:NnNTF \g__tag_role_tags_NS_prop {#2} \l__tag_tmpa_tl
217     {
218       \int_compare:nNtT {\l__tag_loglevel_int} > { 0 }
219       {
220         \msg_info:nnnn { tag } {role-tag} {#1} {#2}
221       }
222     }
223     {
224       \msg_error:nnn { tag } {role-unknown} {#2}
225     }
226   }
227 }

```

Similar with a namespace

```

228 \cs_new_protected:Npn \__tag_check_add_tag_role:nnn #1 #2 #3 %#1 tag/NS, #2 role #3 namespace
229 {
230   \tl_if_empty:nTF {#2}

```

```

231     {
232     \msg_error:nnn { tag } {role-missing} {#1}
233     }
234     {
235     \prop_get:cnNTF { g__tag_role_NS_#3_prop } {#2} \l__tag_tmpa_tl
236     {
237     \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
238     {
239     \msg_info:nnnn { tag } {role-tag} {#1} {#2/#3}
240     }
241     }
242     {
243     \msg_error:nnn { tag } {role-unknown} {#2/#3}
244     }
245     }
246 }

```

(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).

```

247 \cs_new_protected:Npn \__tag_check_mc_if_nested:
248 {
249   \__tag_mc_if_in:T
250   {
251     \msg_warning:nne { tag } {mc-nested} { \__tag_get_mc_abs_cnt: }
252   }
253 }
254
255 \cs_new_protected:Npn \__tag_check_mc_if_open:
256 {
257   \__tag_mc_if_in:F
258   {
259     \msg_warning:nne { tag } {mc-not-open} { \__tag_get_mc_abs_cnt: }
260   }
261 }

```

(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.

```

262 \cs_new_protected:Npn \__tag_check_mc_pushed_popped:nn #1 #2
263 {
264   \int_compare:nNnT
265     { \l__tag_loglevel_int } = { 2 }
266     { \msg_info:nne {tag}{mc-#1}{#2} }
267   \int_compare:nNnT
268     { \l__tag_loglevel_int } > { 2 }
269     {
270     \msg_info:nne {tag}{mc-#1}{#2}
271     \seq_log:N \g__tag_mc_stack_seq

```

```

272     }
273 }

```

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

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

```

274 \cs_new_protected:Npn \_tag_check_mc_tag:N #1 % #1 is var with a tag name in it
275 {
276   \tl_if_empty:NT #1
277   {
278     \msg_error:nne { tag } {mc-tag-missing} { \_tag_get_mc_abs_cnt: }
279   }
280   \prop_if_in:NoF \g__tag_role_tags_NS_prop {#1}
281   {
282     \msg_warning:nne { tag } {role-unknown-tag} {#1}
283   }
284 }

```

(End of definition for `_tag_check_mc_tag:N`.)

`\g__tag_check_mc_used_intarray`
`_tag_check_init_mc_used:`

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

```

285 \cs_new_protected:Npn \_tag_check_init_mc_used:
286 {
287   \intarray_new:Nn \g__tag_check_mc_used_intarray { 65536 }
288   \cs_gset_eq:NN \_tag_check_init_mc_used: \prg_do_nothing:
289 }

```

(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.

```

290 \cs_new_protected:Npn \_tag_check_mc_used:n #1 % #1 mcid abs cnt
291 {
292   \int_compare:nNnT {\l__tag_loglevel_int} > { 2 }
293   {
294     \_tag_check_init_mc_used:
295     \intarray_gset:Nnn \g__tag_check_mc_used_intarray
296       {#1}
297       { \intarray_item:Nn \g__tag_check_mc_used_intarray {#1} + 1 }
298     \int_compare:nNnT
299       {
300         \intarray_item:Nn \g__tag_check_mc_used_intarray {#1}
301       }
302       >
303       { 1 }
304       {
305         \msg_warning:nnn { tag } {mc-used-twice} {#1}
306       }
307   }
308 }

```

(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.

```
309 \cs_new_protected:Npn \_tag\_check\_show\_MCID\_by\_page:
310 {
311   \tl\_set:Ne \l\_tag\_tmpa\_tl
312   {
313     \_tag\_property\_ref\_lastpage:nn
314     {abspage}
315     {-1}
316   }
317   \int\_step\_inline:nnnn {1}{1}
318   {
319     \l\_tag\_tmpa\_tl
320   }
321   {
322     \seq\_clear:N \l\_tag\_tmpa\_seq
323     \int\_step\_inline:nnnn
324     {1}
325     {1}
326     {
327       \_tag\_property\_ref\_lastpage:nn
328       {tagmcabs}
329       {-1}
330     }
331     {
332       \int\_compare:nT
333       {
334         \property\_ref:enn
335         {mcid-####1}
336         {tagabspage}
337         {-1}
338         =
339         ##1
340       }
341       {
342         \seq\_gput\_right:Ne \l\_tag\_tmpa\_seq
343         {
344           Page##1-####1-
345           \property\_ref:enn
346           {mcid-####1}
347           {tagmcid}
348           {-1}
349         }
350       }
351     }
352     \seq\_show:N \l\_tag\_tmpa\_seq
353   }
354 }
```

(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)
`_tag_check_mc_in_galley:TF` 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.

```

355 \prg_new_conditional:Npnn \_tag\_check\_if\_mc\_in\_galley: { T,F,TF }
356 {
357   \tl\_if\_eq:NNTF \l\_tag\_mc\_firstmarks\_seq \l\_tag\_mc\_botmarks\_seq
358   { \prg\_return\_false: }
359   { \prg\_return\_true: }
360 }

```

(End of definition for _tag_check_if_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
`_tag_check_if_mc_tmb_missing:TF` the case if the firstmarks start with `e-` or `b+`. Like above we assume that the marks content is already in the seq's.

```

361 \prg_new_conditional:Npnn \_tag\_check\_if\_mc\_tmb\_missing: { T,F,TF }
362 {
363   \bool\_if:nTF
364   {
365     \str\_if\_eq\_p:ee {\seq\_item:Nn \l\_tag\_mc\_firstmarks\_seq {1}}{e-}
366     ||
367     \str\_if\_eq\_p:ee {\seq\_item:Nn \l\_tag\_mc\_firstmarks\_seq {1}}{b+}
368   }
369   { \prg\_return\_true: }
370   { \prg\_return\_false: }
371 }

```

(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.

```

372 \prg_new_conditional:Npnn \_tag\_check\_if\_mc\_tme\_missing: { T,F,TF }
373 {
374   \str\_if\_eq:eeTF {\seq\_item:Nn \l\_tag\_mc\_botmarks\_seq {1}}{b+}
375   { \prg\_return\_true: }
376   { \prg\_return\_false: }
377 }

```

(End of definition for _tag_check_if_mc_tme_missing:TF.)

```

378 </package>

```

```

379 <*debug>

```

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

```

380 \msg_new:nnn { tag / debug } {mc-begin} { MC-begin~#1-with-options:~\tl_to_str:n{#2}~[\msg_line_context:] }
381 \msg_new:nnn { tag / debug } {mc-end} { MC-end~#1~[\msg_line_context:] }
382
383 \cs_new_protected:Npn \__tag_debug_mc_begin_insert:n #1
384 {
385   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
386   {
387     \msg_note:nnnn { tag / debug } {mc-begin} {inserted} { #1 }
388   }
389 }
390 \cs_new_protected:Npn \__tag_debug_mc_begin_ignore:n #1
391 {
392   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
393   {
394     \msg_note:nnnn { tag / debug } {mc-begin} {ignored} { #1 }
395   }
396 }
397 \cs_new_protected:Npn \__tag_debug_mc_end_insert:
398 {
399   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
400   {
401     \msg_note:nnn { tag / debug } {mc-end} {inserted}
402   }
403 }
404 \cs_new_protected:Npn \__tag_debug_mc_end_ignore:
405 {
406   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
407   {
408     \msg_note:nnn { tag / debug } {mc-end} {ignored}
409   }
410 }

```

And now something for the structures

```

411 \msg_new:nnn { tag / debug } {struct-begin}
412 {
413   Struct~\tag_get:n{struct_num}~begin~#1~with~options:~\tl_to_str:n{#2}~\[\msg_line_context:]
414 }
415 \msg_new:nnn { tag / debug } {struct-end}
416 {
417   Struct~end~#1~[\msg_line_context:]
418 }
419 \msg_new:nnn { tag / debug } {struct-end-wrong}
420 {
421   Struct~end~'#1'~doesn't~fit~start~'#2'~[\msg_line_context:]
422 }
423
424 \cs_new_protected:Npn \__tag_debug_struct_begin_insert:n #1
425 {
426   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
427   {
428     \msg_note:nnnn { tag / debug } {struct-begin} {inserted} { #1 }
429     \seq_log:N \g__tag_struct_tag_stack_seq

```

```

430     }
431 }
432 \cs_new_protected:Npn \__tag_debug_struct_begin_ignore:n #1
433 {
434   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
435   {
436     \msg_note:nnnn { tag / debug } {struct-begin } {ignored} { #1 }
437   }
438 }
439 \cs_new_protected:Npn \__tag_debug_struct_end_insert:
440 {
441   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
442   {
443     \msg_note:nnn { tag / debug } {struct-end} {inserted}
444     \seq_log:N \g__tag_struct_tag_stack_seq
445   }
446 }
447 \cs_new_protected:Npn \__tag_debug_struct_end_ignore:
448 {
449   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
450   {
451     \msg_note:nnn { tag / debug } {struct-end } {ignored}
452   }
453 }
454 \cs_new_protected:Npn \__tag_debug_struct_end_check:n #1
455 {
456   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
457   {
458     \seq_get:NNT \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
459     {
460       \str_if_eq:eeF
461         {#1}
462         {\exp_last_unbraced:NV\use_i:nn \l__tag_tmpa_tl}
463         {
464           \msg_warning:nnee { tag/debug }{ struct-end-wrong }
465           {#1}
466           {\exp_last_unbraced:NV\use_i:nn \l__tag_tmpa_tl}
467         }
468     }
469   }
470 }

```

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.

```

471 \msg_new:nnn { tag / debug } {tag-suspend}
472 {
473   \int_if_zero:nTF
474     {#1}
475     {Tagging~suspended}
476     {Tagging~(not)~suspended~(already-inactive)}\
477     level:~#1~==>~\int_eval:n{#1+1}\tl_if_empty:nF{#2}{,~label:~#2}~[\msg_line_context:]
478 }
479 \msg_new:nnn { tag / debug } {tag-resume}
480 {

```

```

481 \int_if_zero:nTF
482   {#1}
483   {Tagging~resumed}
484   {Tagging~(not)~resumed}\\
485   level:~\int_eval:n{#1+1}~==>~#1\tl_if_empty:nF{#2}{~,~label:~#2}~[\msg_line_context:]
486 }
487 </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?

```

488 <*package>
489 \cs_new_protected:Npn \__tag_check_benchmark_tic:{}
490 \cs_new_protected:Npn \__tag_check_benchmark_toc:{}
491 \cs_new_protected:Npn \tag_check_benchmark_on:
492 {
493   \cs_if_exist:NT \benchmark_tic:
494   {
495     \cs_set_eq:NN \__tag_check_benchmark_tic: \benchmark_tic:
496     \cs_set_eq:NN \__tag_check_benchmark_toc: \benchmark_toc:
497   }
498 }
499 </package>

```

Part III

The `tagpdf-user` module

Code related to L^AT_EX2e user commands and document commands Part of the `tagpdf` package

1 Setup commands

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

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

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

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

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

2 Commands related to mc-chunks

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

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

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

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

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

3 Commands related to structures

<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

<code>\ShowTagging</code>	<code>\ShowTagging{⟨key-val⟩}</code>
---------------------------	--------------------------------------

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

<code>mc-data (show-key)</code>	<code>mc-data = ⟨number⟩</code>
---------------------------------	---------------------------------

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.

<code>mc-current (show-key)</code>	<code>mc-current</code>
------------------------------------	-------------------------

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.

<code>mc-marks (show-key)</code>	<code>mc-marks = show use</code>
----------------------------------	----------------------------------

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

<code>struct-stack (show-key)</code>	<code>struct-stack = log show</code>
--------------------------------------	--------------------------------------

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`.

<code>debug/structures (show-key)</code>	<code>debug/structures = ⟨structure number⟩</code>
--	--

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.

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

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

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

5.3 Header and footer

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

```
page/exclude-header-footer (setup-key) page/exclude-header-footer = true|false|pagination
```

5.4 Link tagging

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

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

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

6 Socket support

```
\tag_socket_use:n \tag_socket_use:n {<socket name>}  
\tag_socket_use:nn \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.,

```
\UseTaggingSocket{foo} → \UseSocket{tagsupport/foo}
```

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

Usually, these sockets have (beside the default plug defined for every socket) one additional plug defined and directly assigned. This plug is used when tagging is active.

There may be more plugs, e.g., tagging with special debugging or special behaviour depending on the class or PDF version etc., but right now it is usually just on or off.

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

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

For special cases like in some `\halign` contexts we need a fully expandable version of the 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} {2025-01-12} {0.991}
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 37.)

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

`\tagtool`

```

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 37.)

9 Commands for the mc-chunks

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

(End of definition for `\tagmcbegin`, `\tagmcbegin`, and `\tagmcbegin`. These functions are documented on page 37.)

`\tagmcbeginTF` 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 \tagmcbeginTF { m m }
41 {
42   \tag_mc_if_in:TF { #1 } { #2 }
43 }
44 < /package >
```

(End of definition for `\tagmcbeginTF`. This function is documented on page 37.)

10 Commands for the structure

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

```
\tagstructbegin 45 < *base >
\tagstructbegin 46 \NewDocumentCommand \tagstructbegin { m }
\tagstructbegin 47 {
\tagstructbegin 48   \tag_struct_begin:n {#1}
\tagstructbegin 49 }
\tagstructbegin 50
\tagstructbegin 51 \NewDocumentCommand \tagstructbegin { }
\tagstructbegin 52 {
\tagstructbegin 53   \tag_struct_end:
\tagstructbegin 54 }
```

```

55
56 \NewDocumentCommand \tagstructure { m }
57   {
58     \tag_struct_use:n {#1}
59   }
60 </base>

```

(End of definition for `\tagstructbegin`, `\tagstructend`, and `\tagstructure`. These functions are documented on page 38.)

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 72 <*package>
\tag_socket_use:nnn 73 \cs_set_protected:Npn \tag_socket_use:n #1
\UseTaggingSocket 74   {
\tag_socket_use_expandable:n 75   \bool_if:NT \l__tag_active_socket_bool
\UseExpandableTaggingSocket 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 40.)

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 38.)

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     \sys_if_engine luatex:T
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 38.)

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       \sys_if_engine luatex:T
149       {
150         \int_compare:nNnTF
151         { -2147483647 }
152         =
153         {
154           \lua_now:e
155           {
156             tex.print
157             (tex.getattribute
158              (luatexbase.attributes.g__tag_mc_cnt_attr))
159           }
160         }
161         {
162           \lua_now:e
163           {
164             ltx.__tag.trace.log
165             (
166               "mc-current:~no~MC~open,~current~absent
167               =\__tag_get_mc_abs_cnt:"
168               ,0
169             )
170             texio.write_nl("")
171           }
172         }
173         {
174           \lua_now:e
175           {
176             ltx.__tag.trace.log
177             (

```

```

178         "mc-current:~absent=~\__tag_get_mc_abs_cnt:=="
179         ..
180         tex.getattribute(luatexbase.attributes.g__tag_mc_cnt_attr)
181         ..
182         "~=>tag="
183         ..
184         tostring
185         (ltx.__tag.func.get_tag_from
186         (tex.getattribute
187         (luatexbase.attributes.g__tag_mc_type_attr)))
188         ..
189         "="
190         ..
191         tex.getattribute
192         (luatexbase.attributes.g__tag_mc_type_attr)
193         ,0
194         )
195         texio.write_nl("")
196     }
197 }
198 }
199 }
200 {
201     \msg_note:nn{ tag }{ mc-current }
202 }
203 }
204 }

```

(End of definition for mc-current (show-key). This function is documented on page 38.)

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).

```

205 \keys_define:nn { __tag / show }
206 {
207     mc-marks .choice: ,
208     mc-marks / show .code:n =
209     {
210         \__tag_mc_get_marks:
211         \__tag_check_if_mc_in_galley:TF
212         {
213             \iow_term:n {Marks~from~this~page:~}
214         }
215         {
216             \iow_term:n {Marks~from~a~previous~page:~}
217         }
218         \seq_show:N \l__tag_mc_firstmarks_seq
219         \seq_show:N \l__tag_mc_botmarks_seq
220         \__tag_check_if_mc_tmb_missing:T
221         {
222             \iow_term:n {BDC~missing~on~this~page!}
223         }
224         \__tag_check_if_mc_tme_missing:T
225         {
226             \iow_term:n {EMC~missing~on~this~page!}

```

```

227     }
228   },
229   mc-marks / use .code:n =
230   {
231     \__tag_mc_get_marks:
232     \__tag_check_if_mc_in_galley:TF
233     { Marks~from~this~page:~}
234     { Marks~from~a~previous~page:~}
235     \seq_use:Nn \l__tag_mc_firstmarks_seq {,~}\quad
236     \seq_use:Nn \l__tag_mc_botmarks_seq {,~}\quad
237     \__tag_check_if_mc_tmb_missing:T
238     {
239       BDC~missing~
240     }
241     \__tag_check_if_mc_tme_missing:T
242     {
243       EMC~missing
244     }
245   },
246   mc-marks .default:n = show
247 }

```

(End of definition for mc-marks (show-key). This function is documented on page 38.)

struct-stack (show-key)

```

248 \keys_define:nn { __tag / show }
249 {
250   struct-stack .choice:
251   ,struct-stack / log .code:n = \seq_log:N \g__tag_struct_tag_stack_seq
252   ,struct-stack / show .code:n = \seq_show:N \g__tag_struct_tag_stack_seq
253   ,struct-stack .default:n = show
254 }
255 </package>

```

(End of definition for struct-stack (show-key). This function is documented on page 38.)

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.

```

256 <*debug>
257 \keys_define:nn { __tag / show }
258 {
259   ,debug/structures .code:n =
260   {
261     \int_step_inline:nnn{#1}{\c@g__tag_struct_abs_int}
262     {
263       \msg_term:nneeee
264       { tag/debug } { show-struct }
265       { ##1 }
266       {
267         \prop_map_function:cN
268         {g__tag_struct_debug_##1_prop}
269         \msg_show_item_unbraced:nn
270       }
271     } { } { }

```

```

272         \msg_term:nneeee
273         { tag/debug } { show-kids }
274         { ##1 }
275         {
276         \seq_map_function:cN
277         {g__tag_struct_debug_kids_##1_seq}
278         \msg_show_item_unbraced:n
279         }
280         { } { }
281     }
282 }
283 ,debug/structures .default:n = 1
284 }
285 </debug>

```

(End of definition for `debug/structures (show-key)`. This function is documented on page 38.)

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.

```
286 (*package)
```

13.1 Document structure

```

\g__tag_root_default_tl
  activate (setup-key)
activate/socket (setup-key)
287 \tl_new:N\g__tag_root_default_tl
288 \tl_gset:Nn\g__tag_root_default_tl {Document}
289
290 \hook_gput_code:nnn{begindocument}{tagpdf}{\tagstructbegin{tag=\g__tag_root_default_tl}}
291 \hook_gput_code:nnn{tagpdf/finish/before}{tagpdf}{\tagstructend}
292
293 \keys_define:nn { __tag / setup}
294 {
295   activate/socket .bool_set:N = \l__tag_active_socket_bool,
296   activate .code:n =
297   {
298     \keys_set:nn { __tag / setup }
299     { activate/mc,activate/tree,activate/struct,activate/socket }
300     \tl_gset:Nn\g__tag_root_default_tl {#1}
301   },
302   activate .default:n = Document
303 }
304

```

(End of definition for `\g__tag_root_default_tl`, `activate (setup-key)`, and `activate/socket (setup-key)`. These functions are documented on page 37.)

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.

```

305 \AddToHook{begindocument/before}
306   {
307     \bool_lazy_and:nnT
308       { \g__tag_active_struct_dest_bool }
309       { \g__tag_active_struct_bool }
310     {
311       \tl_set:Nn \l_pdf_current_structure_destination_tl
312         { {__tag/struct}{\g__tag_struct_stack_current_tl } }
313       \pdf_activate_indexed_structure_destination:
314     }
315   }

```

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

```

316 \sys_if_engine_luatex:T
317   {
318     \NewDocumentCommand\pdffakespace { }
319     {
320       \__tag_fakespace:
321     }
322   }
323 \providecommand\pdffakespace{}

```

(End of definition for `\pdffakespace`. This function is documented on page 39.)

13.4 Paratagging

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

```

\l__tag_para_bool
\l__tag_para_flattened_bool
\l__tag_para_show_bool
\g__tag_para_begin_int
\g__tag_para_end_int
\g__tag_para_main_begin_int
\g__tag_para_main_end_int
\g__tag_para_main_struct_tl
\l__tag_para_tag_default_tl
\l__tag_para_tag_tl
\l__tag_para_main_tag_tl
\l__tag_para_attr_class_tl
\l__tag_para_main_attr_class_tl

```

At first some variables.

```

324 </package>
325 <base>\bool_new:N \l__tag_para_flattened_bool
326 <base>\bool_new:N \l__tag_para_bool
327 <*package>
328 \int_new:N \g__tag_para_begin_int
329 \int_new:N \g__tag_para_end_int
330 \int_new:N \g__tag_para_main_begin_int
331 \int_new:N \g__tag_para_main_end_int

```

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

```

332 \tl_new:N \g__tag_para_main_struct_tl
333 \tl_gset:Nn \g__tag_para_main_struct_tl {1}
334 \tl_new:N \l__tag_para_tag_default_tl
335 \tl_set:Nn \l__tag_para_tag_default_tl { text }
336 \tl_new:N \l__tag_para_tag_tl

```

```

337 \tl_set:Nn \l__tag_para_tag_tl { \l__tag_para_tag_default_tl }
338 \tl_new:N \l__tag_para_main_tag_tl
339 \tl_set:Nn \l__tag_para_main_tag_tl {text-unit}

```

this is perhaps already defined by the block code

```

340 \tl_if_exist:NF \l__tag_para_attr_class_tl
341 { \tl_new:N \l__tag_para_attr_class_tl }
342 \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_begin_int:`

`__tag_gincr_para_end_int:`

```

343 \cs_new_protected:Npn \__tag_gincr_para_main_begin_int:
344 {
345   \int_gincr:N \g__tag_para_main_begin_int
346 }
347 \cs_new_protected:Npn \__tag_gincr_para_begin_int:
348 {
349   \int_gincr:N \g__tag_para_begin_int
350 }
351 \cs_new_protected:Npn \__tag_gincr_para_main_end_int:
352 {
353   \int_gincr:N \g__tag_para_main_end_int
354 }
355 \cs_new_protected:Npn \__tag_gincr_para_end_int:
356 {
357   \int_gincr:N \g__tag_para_end_int
358 }

```

(End of definition for __tag_gincr_para_main_begin_int: and others.)

`__tag_start_para_ints:`

`__tag_stop_para_ints:`

```

359 \cs_new_protected:Npn \__tag_start_para_ints:
360 {
361   \cs_set_protected:Npn \__tag_gincr_para_main_begin_int:
362   {
363     \int_gincr:N \g__tag_para_main_begin_int
364   }
365   \cs_set_protected:Npn \__tag_gincr_para_begin_int:
366   {
367     \int_gincr:N \g__tag_para_begin_int
368   }
369   \cs_set_protected:Npn \__tag_gincr_para_main_end_int:
370   {
371     \int_gincr:N \g__tag_para_main_end_int
372   }
373   \cs_set_protected:Npn \__tag_gincr_para_end_int:
374   {
375     \int_gincr:N \g__tag_para_end_int
376   }
377 }
378 \cs_new_protected:Npn \__tag_stop_para_ints:
379 {
380   \cs_set_eq:NN \__tag_gincr_para_main_begin_int:\prg_do_nothing:

```

```

381 \cs_set_eq:NN \__tag_gincr_para_begin_int: \prg_do_nothing:
382 \cs_set_eq:NN \__tag_gincr_para_main_end_int: \prg_do_nothing:
383 \cs_set_eq:NN \__tag_gincr_para_end_int: \prg_do_nothing:
384 }

```

(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:

```

385 \cs_new:Npn \__tag_para_main_store_struct:
386 {
387   \tl_gset:Ne \g__tag_para_main_struct_tl {\int_use:N \c@g__tag_struct_abs_int }
388 }

```

(End of definition for __tag_para_main_store_struct:.)

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

```

389 \AddToHook{package/latex-lab-testphase-sec/after}
390 {
391   \cs_set_protected:Npn \@kernel@tag@hangfrom #1
392   {
393     \tagstructbegin{tag=\l__tag_para_tag_tl}
394     \__tag_gincr_para_begin_int:
395     \tagstructbegin{tag=Lbl}
396     \setbox\@tempboxa
397     \hbox
398     {
399       \bool_lazy_and:nnT
400       {\tag_if_active_p:}
401       {\g__tag_mode_lua_bool}
402       {\tagmcbegin{tag=Lbl}}
403       {#1}
404     }
405     \tag_suspend:n{hangfrom}
406     \hangindent \wd\@tempboxa\noindent
407     \tag_resume:n{hangfrom}
408     \tagmcbegin{}\box\@tempboxa\tagmcbegin\tagstructend\tagmcbegin{}
409   }
410 }

```

and one temporary adaptations for the block module:

```

411 \AddToHook{package/latex-lab-testphase-block/after}
412 {
413   \tl_if_exist:NT \l_tag_para_attr_class_tl
414   {
415     \tl_set:Nn \l__tag_para_attr_class_tl { \l_tag_para_attr_class_tl }
416   }
417 }
418

```

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/tag (setup-key)`

`para/maintag (setup-key)`

`para/tagging (tool-key)`

`para/tag (tool-key)`

`para/maintag (tool-key)`

`para/flattened (tool-key)`

`unittag (deprecated)`

`para-flattened (deprecated)`

paratagging (deprecated)

`paratagging-show (deprecated)`

`paratag (deprecated)`

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`

```

419 \keys_define:nn { __tag / setup }
420 {
421   para/tagging      .bool_set:N = \l__tag_para_bool,
422   debug/show/para  .code:n = {\bool_set_true:N \l__tag_para_show_bool},
423   debug/show/paraOff .code:n = {\bool_set_false:N \l__tag_para_show_bool},
424   para/tag         .tl_set:N = \l__tag_para_tag_tl,
425   para/maintag     .tl_set:N = \l__tag_para_main_tag_tl,
426   para/flattened   .bool_set:N = \l__tag_para_flattened_bool
427 }
428 \keys_define:nn { tag / tool}
429 {
430   para/tagging     .bool_set:N = \l__tag_para_bool,
431   para/tag         .tl_set:N = \l__tag_para_tag_tl,
432   para/maintag     .tl_set:N = \l__tag_para_main_tag_tl,
433   para/flattened   .bool_set:N = \l__tag_para_flattened_bool
434 }

```

the deprecated names

```

435 \keys_define:nn { __tag / setup }
436 {
437   paratagging      .bool_set:N = \l__tag_para_bool,
438   paratagging-show .bool_set:N = \l__tag_para_show_bool,
439   paratag         .tl_set:N = \l__tag_para_tag_tl
440 }
441 \keys_define:nn { tag / tool}
442 {
443   para      .bool_set:N = \l__tag_para_bool,
444   paratag   .tl_set:N = \l__tag_para_tag_tl,
445   unittag   .tl_set:N = \l__tag_para_main_tag_tl,
446   para-flattened .bool_set:N = \l__tag_para_flattened_bool
447 }

```

(End of definition for `para/tagging` (setup-key) and others. These functions are documented on page 39.)

Helper command for debugging:

```

448 \cs_new_protected:Npn \__tag_check_para_begin_show:nn #1 #2
449   %#1 color, #2 prefix
450   {
451     \bool_if:NT \l__tag_para_show_bool
452     {
453       \tag_mc_begin:n{artifact}
454       \llap{\color_select:n{#1}\tiny#2\int_use:N\g__tag_para_begin_int\ }
455       \tag_mc_end:
456     }
457   }
458
459 \cs_new_protected:Npn \__tag_check_para_end_show:nn #1 #2
460   %#1 color, #2 prefix
461   {
462     \bool_if:NT \l__tag_para_show_bool
463     {

```

```

464     \tag_mc_begin:n{artifact}
465     \rlap{\color_select:n{#1}\tiny\ #2\int_use:N\g__tag_para_end_int}
466     \tag_mc_end:
467   }
468 }

```

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.

```

469 \str_if_exist:cF { l__socket_tagsupport/para/begin_plug_str }
470 {
471   \socket_new:nn      {tagsupport/para/begin}{0}
472   \socket_new:nn      {tagsupport/para/end}{0}
473
474   \socket_new_plug:nnn{tagsupport/para/begin}{plain}
475   {
476     \bool_if:NT \l__tag_para_bool
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         }
485         \__tag_para_main_store_struct:
486       }
487       \__tag_gincr_para_begin_int:
488       \tag_struct_begin:n {tag=\l__tag_para_tag_tl}
489       \__tag_check_para_begin_show:nn {green}{ }
490       \tag_mc_begin:n { }
491     }
492   }
493   \socket_new_plug:nnn{tagsupport/para/begin}{block}
494   {
495     \bool_if:NT \l__tag_para_bool
496     {
497       \legacy_if:nF { @inlabel }
498       {
499         \__tag_check_typeout_v:n
500         {=>~ @endpe = \legacy_if:nTF { @endpe }{true}{false} \on@line }
501         \legacy_if:nF { @endpe }
502         {
503           \bool_if:NF \l__tag_para_flattened_bool
504           {
505             \__tag_gincr_para_main_begin_int:
506             \tag_struct_begin:n
507             {
508               tag=\l__tag_para_main_tag_tl,
509               attribute-class=\l__tag_para_main_attr_class_tl,
510             }
511             \__tag_para_main_store_struct:

```

```

512         }
513     }
514     \_tag_gincr_para_begin_int:
515     \_tag_check_typeout_v:n {==>~increment~ P \on@line }
516     \tag_struct_begin:n
517     {
518         tag=\l__tag_para_tag_tl
519         ,attribute-class=\l__tag_para_attr_class_tl
520     }
521     \_tag_check_para_begin_show:nn {green}{\PARALABEL}
522     \tag_mc_begin:n {}
523 }
524 }
525 }

```

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

```

526     \socket_new_plug:nnn{tagsupport/para/end}{plain}
527     {
528         \bool_if:NT \l__tag_para_bool
529         {
530             \_tag_gincr_para_end_int:
531             \_tag_check_typeout_v:n {==>~increment~ /P \on@line }
532             \tag_mc_end:
533             \_tag_check_para_end_show:nn {red}{}
534             \tag_struct_end:
535             \bool_if:NF \l__tag_para_flattened_bool
536             {
537                 \_tag_gincr_para_main_end_int:
538                 \tag_struct_end:
539             }
540         }
541     }
542 }

```

By default we assign the plain plug:

```

543 \socket_assign_plug:nn { tagsupport/para/begin}{plain}
544 \socket_assign_plug:nn { tagsupport/para/end}{plain}

```

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

```

545 \AddToHook{para/begin}{ \socket_use:n { tagsupport/para/begin }
546 }
547 \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.

```

548 \AddToHook{package/latex-lab-testphase-block/after}
549 {
550     \RemoveFromHook{para/begin}[tagpdf]
551     \RemoveFromHook{para/end}[latex-lab-testphase-block]
552     \AddToHook{para/begin}[tagpdf]
553     {
554         \socket_use:n { tagsupport/para/begin }
555     }

```

```

556 \AddToHook{para/end}[tagpdf]
557   {
558     \socket_use:n { tagsupport/para/end }
559   }
560 \socket_assign_plug:nn { tagsupport/para/begin}{block}
561 }
562

```

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.

```

563 \AddToHook{enddocument/info}
564   {
565     \tag_if_active:F
566     {
567       \msg_redirect_name:nnn { tag } { para-hook-count-wrong } { warning }
568     }
569     \int_compare:nNnF {\g__tag_para_main_begin_int}={\g__tag_para_main_end_int}
570     {
571       \msg_error:nneee
572         {tag}
573         {para-hook-count-wrong}
574         {\int_use:N\g__tag_para_main_begin_int}
575         {\int_use:N\g__tag_para_main_end_int}
576         {text-unit}
577     }
578     \int_compare:nNnF {\g__tag_para_begin_int}={\g__tag_para_end_int}
579     {
580       \msg_error:nneee
581         {tag}
582         {para-hook-count-wrong}
583         {\int_use:N\g__tag_para_begin_int}
584         {\int_use:N\g__tag_para_end_int}
585         {text}
586     }
587   }

```

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

```

588 \@ifpackageloaded{footmisc}
589   {\PackageWarning{tagpdf}{tagpdf-has-been-loaded-too-late!}} %
590   {\RequirePackage{latex-lab-testphase-new-or-1}}
591
592 \AddToHook{begindocument/before}
593   {
594     \providecommand\@kernel@tagsupport@makecol{}
595     \providecommand\@kernel@before@cclv{}
596     \bool_if:NF \g__tag_mode_lua_bool
597     {
598       \cs_if_exist:NT \@kernel@before@footins
599       {
600         \tl_put_right:Nn \@kernel@before@footins
601           { \tag_mc_add_missing_to_stream:Nn \footins {footnote} }
602         \tl_put_right:Nn \@kernel@before@cclv
603           {

```

```

604         \tag_mc_add_missing_to_stream:Nn \@cclv {main}
605     }
606 \tl_put_right:Nn \@kernel@taggsupport@makecol
607 {
608     \tag_mc_add_missing_to_stream:Nn \@outputbox {main}
609 }
610 \tl_if_exist:NT \@mult@ptagging@hook
611 {
612     \tl_put_right:Nn \@mult@ptagging@hook
613     {
614         \tag_mc_add_missing_to_stream:Nn \count@ {multicol}
615     }
616     \tag_mc_add_missing_to_stream:Nn \mult@rightbox {multicol}
617 }
618 }
619 }
620 }
621 }
622 }
623 }
624 }
625 }
626 }
627 </package>

```

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

`\tagpdfparaOff`

```

628 <base>\newcommand\tagpdfparaOn {}
629 <base>\newcommand\tagpdfparaOff{}
630 <*package>
631 \renewcommand\tagpdfparaOn {\bool_set_true:N \l__tag_para_bool}
632 \renewcommand\tagpdfparaOff{\bool_set_false:N \l__tag_para_bool}

```

(End of definition for `\tagpdfparaOn` and `\tagpdfparaOff`. These functions are documented on page 39.)

`\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{\tagmcbegin}\tagstructend}%

633 \NewDocumentCommand\tagpdfsuppressmarks{m}
634   {{\use:c{__tag_mc_disable_marks:} #1}}

```

(End of definition for `\tagpdfsuppressmarks`. This function is documented on page 39.)

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)

```
635 \keys_define:nn { __tag / setup }
636 {
637   text / lang .tl_set:N = \l__tag_struct_lang_tl
638 }
```

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

13.6 Header and footer

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

```
639 \cs_new_protected:Npn \__tag_hook_kernel_before_head: {}
640 \cs_new_protected:Npn \__tag_hook_kernel_after_head: {}
641 \cs_new_protected:Npn \__tag_hook_kernel_before_foot: {}
642 \cs_new_protected:Npn \__tag_hook_kernel_after_foot: {}
643
644 \AddToHook{begindocument}
645 {
646   \cs_if_exist:NT \@kernel@before@head
647   {
648     \tl_put_right:Nn \@kernel@before@head {\__tag_hook_kernel_before_head:}
649     \tl_put_left:Nn \@kernel@after@head {\__tag_hook_kernel_after_head:}
650     \tl_put_right:Nn \@kernel@before@foot {\__tag_hook_kernel_before_foot:}
651     \tl_put_left:Nn \@kernel@after@foot {\__tag_hook_kernel_after_foot:}
652   }
653 }
654
655 \bool_new:N \g__tag_saved_in_mc_bool
656 \cs_new_protected:Npn \__tag_exclude_headfoot_begin:
657 {
658   \bool_set_false:N \l__tag_para_bool
659   \bool_if:NTF \g__tag_mode_lua_bool
660   {
661     \tag_mc_end_push:
662   }
663   {
664     \bool_gset_eq:NN \g__tag_saved_in_mc_bool \g__tag_in_mc_bool
665     \bool_gset_false:N \g__tag_in_mc_bool
666   }
667   \tag_mc_begin:n {artifact}
668   \tag_suspend:n{headfoot}
669 }
670 \cs_new_protected:Npn \__tag_exclude_headfoot_end:
671 {
672   \tag_resume:n{headfoot}
673   \tag_mc_end:
674   \bool_if:NTF \g__tag_mode_lua_bool
```

```

675     {
676       \tag_mc_begin_pop:n{ }
677     }
678     {
679       \bool_gset_eq:NN \g__tag_in_mc_bool\g__tag_saved_in_mc_bool
680     }
681   }

```

This version allows to use an Artifact structure

```

682 \__tag_attr_new_entry:nn {__tag/attr/pagination}{/0/Artifact/Type/Pagination}
683 \cs_new_protected:Npn \__tag_exclude_struct_headfoot_begin:n #1
684 {
685   \bool_set_false:N \l__tag_para_bool
686   \bool_if:NTF \g__tag_mode_lua_bool
687   {
688     \tag_mc_end_push:
689   }
690   {
691     \bool_gset_eq:NN \g__tag_saved_in_mc_bool \g__tag_in_mc_bool
692     \bool_gset_false:N \g__tag_in_mc_bool
693   }
694   \tag_struct_begin:n{tag=Artifact,attribute-class=__tag/attr/#1}
695   \tag_mc_begin:n {artifact=#1}
696   \tag_suspend:n{headfoot}
697 }
698
699 \cs_new_protected:Npn \__tag_exclude_struct_headfoot_end:
700 {
701   \tag_resume:n{headfoot}
702   \tag_mc_end:
703   \tag_struct_end:
704   \bool_if:NTF \g__tag_mode_lua_bool
705   {
706     \tag_mc_begin_pop:n{ }
707   }
708   {
709     \bool_gset_eq:NN \g__tag_in_mc_bool\g__tag_saved_in_mc_bool
710   }
711 }

```

And now the keys

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

exclude-header-footer (deprecated)

```

712 \keys_define:nn { __tag / setup }
713 {
714   page/exclude-header-footer .choice:,
715   page/exclude-header-footer / true .code:n =
716   {
717     \cs_set_eq:NN \__tag_hook_kernel_before_head: \__tag_exclude_headfoot_begin:
718     \cs_set_eq:NN \__tag_hook_kernel_before_foot: \__tag_exclude_headfoot_begin:
719     \cs_set_eq:NN \__tag_hook_kernel_after_head: \__tag_exclude_headfoot_end:
720     \cs_set_eq:NN \__tag_hook_kernel_after_foot: \__tag_exclude_headfoot_end:
721   },
722   page/exclude-header-footer / pagination .code:n =
723   {

```

```

724     \cs_set:Nn \__tag_hook_kernel_before_head: { \__tag_exclude_struct_headfoot_begin:n {p
725     \cs_set:Nn \__tag_hook_kernel_before_foot: { \__tag_exclude_struct_headfoot_begin:n {p
726     \cs_set_eq:NN \__tag_hook_kernel_after_head: \__tag_exclude_struct_headfoot_end:
727     \cs_set_eq:NN \__tag_hook_kernel_after_foot: \__tag_exclude_struct_headfoot_end:
728   },
729   page/exclude-header-footer / false .code:n =
730   {
731     \cs_set_eq:NN \__tag_hook_kernel_before_head: \prg_do_nothing:
732     \cs_set_eq:NN \__tag_hook_kernel_before_foot: \prg_do_nothing:
733     \cs_set_eq:NN \__tag_hook_kernel_after_head: \prg_do_nothing:
734     \cs_set_eq:NN \__tag_hook_kernel_after_foot: \prg_do_nothing:
735   },
736   page/exclude-header-footer .default:n = true,
737   page/exclude-header-footer .initial:n = true,

```

deprecated name

```

738   exclude-header-footer .meta:n = { page/exclude-header-footer = {#1} }
739 }

```

(End of definition for page/exclude-header-footer (setup-key) and exclude-header-footer (deprecated). These functions are documented on page 40.)

13.7 Links

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

```

740 \hook_gput_code:nnn
741 {pdfannot/link/URI/before}
742 {tagpdf}
743 {
744   \tag_mc_end_push:
745   \tag_struct_begin:n { tag=Link }
746   \tag_mc_begin:n { tag=Link }
747   \pdfannot_dict_put:nne
748     { link/URI }
749     { StructParent }
750     { \tag_struct_parent_int: }
751 }
752
753 \hook_gput_code:nnn
754 {pdfannot/link/URI/after}
755 {tagpdf}
756 {
757   \tag_struct_insert_annot:ee {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
758   \tag_mc_end:
759   \tag_struct_end:
760   \tag_mc_begin_pop:n{ }
761 }
762
763 \hook_gput_code:nnn
764 {pdfannot/link/GoTo/before}
765 {tagpdf}
766 {

```

```

767 \tag_mc_end_push:
768 \tag_struct_begin:n{tag=Link}
769 \tag_mc_begin:n{tag=Link}
770 \pdfannot_dict_put:nne
771 { link/GoTo }
772 { StructParent }
773 { \tag_struct_parent_int: }
774 }
775
776 \hook_gput_code:nnn
777 {pdfannot/link/GoTo/after}
778 {tagpdf}
779 {
780 \tag_struct_insert_annot:ee {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
781 \tag_mc_end:
782 \tag_struct_end:
783 \tag_mc_begin_pop:n{ }
784
785 }
786
787 % "alternative descriptions " for PAX3. How to get better text here??
788 \pdfannot_dict_put:nnn
789 { link/URI }
790 { Contents }
791 { (url) }
792
793 \pdfannot_dict_put:nnn
794 { link/GoTo }
795 { Contents }
796 { (ref) }
797
</package>

```

Part IV

The tagpdf-tree module

Commands trees and main dictionaries

Part of the tagpdf package

```
1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-tree-code} {2025-01-12} {0.991}
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   \msg_note:nn {tag}{tree-statistic}
29 }
```

(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
30 \pdf_object_new_indexed:nm { __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).
31 \tl_new:N \g__tag_tree_openaction_struct_tl
32 \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.
33 \keys_define:nm { __tag / setup }
34 {
35   viewer/startstructure .code:n =
36   {
37     \tl_gset:Ne \g__tag_tree_openaction_struct_tl {#1}
38   }
39   ,viewer/startstructure .default:n = { \int_use:N \c@g__tag_struct_abs_int }
40 }

```

(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:

```

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

```

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.

```

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

```

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

```

57             \int_compare:nNnTF{ \seq_count:N \l__tag_tmpa_seq } > {1}
58             { /\seq_item:Nn\l__tag_tmpa_seq{2} }
59             { ] }
60
61             >>
62         }
63     }
64 }
65 \hook_gput_code:nnn{shipout/lastpage}{tagpdf}
66 {
67     \bool_if:NT \g__tag_active_tree_bool
68     {
69         \pdfmanagement_add:nnn { Catalog / MarkInfo } { Marked } { true }
70         \pdfmanagement_add:nne
71         { Catalog }
72         { StructTreeRoot }
73         { \pdf_object_ref_indexed:nn { __tag/struct } { 1 } }
74         \__tag_tree_update_openaction:
75     }
76 }

```

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

```

77 \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

```

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

```

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.

```

85 \cs_new_protected:Npn \__tag_tree_write_idtree:
86 {
87     \tl_clear:N \l__tag_tmpa_tl
88     \tl_clear:N \l__tag_tmpb_tl
89     \int_zero:N \l__tag_tmpa_int
90     \int_step_inline:nnn {2} {\c@g__tag_struct_abs_int}
91     {
92         \int_incr:N\l__tag_tmpa_int
93         \tl_put_right:Ne \l__tag_tmpa_tl

```

```

94     {
95     \_tag_struct_get_id:n{##1}~\pdf_object_ref_indexed:nn {\_tag/struct}{##1}~
96     }
97 \int_compare:nNnF {\_tag_tmpa_int}<{50} %
98     {
99     \pdf_object_unnamed_write:ne {dict}
100     { /Limits~[\_tag_struct_get_id:n{##1}~\_tag_tmpa_int+1}~\_tag_struct_get_id:
101     /Names~[\_tag_tmpa_tl]
102     }
103     \tl_put_right:Ne\_tag_tmpb_tl {\pdf_object_ref_last:\c_space_tl}
104     \int_zero:N \_tag_tmpa_int
105     \tl_clear:N \_tag_tmpa_tl
106     }
107 }
108 \tl_if_empty:NF \_tag_tmpa_tl
109 {
110 \pdf_object_unnamed_write:ne {dict}
111 {
112 /Limits~
113 [\_tag_struct_get_id:n{\c@g__tag_struct_abs_int~\_tag_tmpa_int+1}~
114 \_tag_struct_get_id:n{\c@g__tag_struct_abs_int}]
115 /Names~[\_tag_tmpa_tl]
116 }
117 \tl_put_right:Ne\_tag_tmpb_tl {\pdf_object_ref_last:}
118 }
119 \pdf_object_unnamed_write:ne {dict}{/Kids~[\_tag_tmpb_tl]}
120 \_tag_prop_gput:cne
121 { g__tag_struct_1_prop }
122 { IDTree }
123 { \pdf_object_ref_last: }
124 }

```

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.

```

125 \cs_new_protected:Npn \_tag_tree_write_structtreeroot:
126 {
127     \_tag_prop_gput:cne
128     { g__tag_struct_1_prop }
129     { ParentTree }
130     { \pdf_object_ref:n { \_tag/tree/parenttree } }
131     \_tag_prop_gput:cne
132     { g__tag_struct_1_prop }
133     { RoleMap }
134     { \pdf_object_ref:n { \_tag/tree/rolemap } }
135     \_tag_struct_fill_kid_key:n { 1 }
136     \prop_gremove:cn { g__tag_struct_1_prop } {S}
137     \_tag_struct_get_dict_content:nN { 1 } \_tag_tmpa_tl
138     \pdf_object_write_indexed:nnne
139     { \_tag/struct } { 1 }
140     {dict}
141     {

```



```

142         \l__tag_tmpa_tl
143     }

```

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

```

144     \prop_gput:cnn { g__tag_struct_1_prop } {S}{ /StructTreeRoot }
145 }

```

(End of definition for `__tag_tree_write_structtreeroot:`.)

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

```

146 \cs_new_protected:Npn \__tag_tree_write_structelems:
147 {
148     \int_step_inline:nnnn {2}{1}{\c@g__tag_struct_abs_int}
149     {
150         \__tag_struct_write_obj:n { ##1 }
151     }
152 }

```

(End of definition for `__tag_tree_write_structelems:`.)

1.5 ParentTree

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

```

153 \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`.

```

154 \newcounter { g__tag_parenttree_obj_int }
155 \hook_gput_code:nnn{begindocument}{tagpdf}
156 {
157     \int_gset:Nn
158         \c@g__tag_parenttree_obj_int
159         { \__tag_property_ref_lastpage:nn{abspage}{100} }
160 }

```

(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`

```

161 \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.

```

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

```

(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.

```

169 \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.

```

170 \cs_new_protected:Npn \_tag_tree_parenttree_rerun_msg: {}
171 \cs_new_protected:Npn \_tag_tree_fill_parenttree:
172 {
173   \int_step_inline:nnnn{1}{1}{\_tag_property_ref_lastpage:nn{abspage}{-1}} %not quite clear
174   { %page ##1
175     \prop_clear:N \l__tag_tmpa_prop
176     \int_step_inline:nnnn{1}{1}{\_tag_property_ref_lastpage:nn{tagmcabs}{-
177     1}}
177     {
178       %mcid###1
179       \int_compare:nT
180       {\property_ref:enn{mcid-###1}{tagabspace}{-1}=##1} %mcid is on current page
181       {% yes
182         \prop_put:Nee
183         \l__tag_tmpa_prop
184         {\property_ref:enn{mcid-###1}{tagmcid}{-1}}
185         {\prop_item:Nn \g__tag_mc_parenttree_prop {###1}}
186       }
187     }
188     \tl_put_right:Ne\l__tag_parenttree_content_tl
189     {
190       \int_eval:n {##1-1}\c_space_tl
191       [\c_space_tl %]
192     }
193     \int_step_inline:nnnn %###1
194     {0}
195     {1}
196     { \prop_count:N \l__tag_tmpa_prop -1 }
197     {
198       \prop_get:NnNTF \l__tag_tmpa_prop {###1} \l__tag_tmpa_tl
199       {% page#1:mcid##1:\l__tag_tmpa_tl :content
200         \tl_put_right:Ne \l__tag_parenttree_content_tl
201         {
202           \prop_if_exist:cTF { g__tag_struct_ \l__tag_tmpa_tl _prop }
203           {

```

```

204         \pdf_object_ref_indexed:nn { __tag/struct }{ \l__tag_tmpa_tl }
205     }
206     {
207         null
208     }
209     \c_space_tl
210 }
211 }
212 {
213     \cs_set_protected:Npn \__tag_tree_parenttree_rerun_msg:
214     {
215         \msg_warning:nn { tag } {tree-mcid-index-wrong}
216     }
217 }
218 }
219 \tl_put_right:Nn
220 \l__tag_parenttree_content_tl
221 {%[
222 ]^^J
223 }
224 }
225 }

```

(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.

```

226 \cs_new_protected:Npn \__tag_tree_lua_fill_parenttree:
227 {
228     \tl_set:Nn \l__tag_parenttree_content_tl
229     {
230         \lua_now:e
231         {
232             ltx.__tag.func.output_parenttree
233             (
234                 \int_use:N\g_shipout_readonly_int
235             )
236         }
237     }
238 }

```

(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?

```

239 \cs_new_protected:Npn \__tag_tree_write_parenttree:
240 {
241     \bool_if:NTF \g__tag_mode_lua_bool
242     {
243         \__tag_tree_lua_fill_parenttree:
244     }
245     {
246         \__tag_tree_fill_parenttree:
247     }
248     \__tag_tree_parenttree_rerun_msg:

```

```

249 \tl_put_right:NV \l__tag_parenttree_content_tl\g__tag_parenttree_objr_tl
250 \pdf_object_write:nne { __tag/tree/parenttree }{dict}
251 {
252   /Nums\c_space_tl [\l__tag_parenttree_content_tl]
253 }
254 }

```

(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.

```

255 \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.

```

256 \cs_new_protected:Npn \__tag_tree_write_rolemap:
257 {
258   \bool_if:NT \g__tag_role_add_mathml_bool
259   {
260     \prop_map_inline:Nn \g__tag_role_NS_mathml_prop
261     {
262       \prop_gput:Nnn \g__tag_role_rolemap_prop {##1}{Span}
263     }
264   }
265   \prop_map_inline:Nn\g__tag_role_rolemap_prop
266   {
267     \tl_if_eq:nnF {##1}{##2}
268     {
269       \pdfdict_gput:nne {g__tag_role/RoleMap_dict}
270       {##1}
271       {\pdf_name_from_unicode_e:n{##2}}
272     }
273   }
274   \pdf_object_write:nne { __tag/tree/rolemap }{dict}
275   {
276     \pdfdict_use:n{g__tag_role/RoleMap_dict}
277   }
278 }

```

(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:
```

```
279 \cs_new_protected:Npn __tag_tree_write_classmap:
280   {
281     \tl_clear:N \l__tag_tmpa_tl
282     \seq_map_inline:Nn \g__tag_attr_class_used_seq
283     {
284       \prop_gput:Nnn \g__tag_attr_class_used_prop {##1}{ }
285     }
286     \prop_map_inline:Nn \g__tag_attr_class_used_prop
287     {
288       \tl_put_right:Ne \l__tag_tmpa_tl
289       {
290         ##1\c_space_tl
291         <<
292         \prop_item:Nn
293         \g__tag_attr_entries_prop
294         {##1}
295         >>
296         \iow_newline:
297       }
298     }
299     \tl_if_empty:NF
300     \l__tag_tmpa_tl
301     {
302       \pdf_object_new:n { __tag/tree/classmap }
303       \pdf_object_write:nne
304       { __tag/tree/classmap }
305       {dict}
306       { \l__tag_tmpa_tl }
307       \__tag_prop_gput:cne
308       { g__tag_struct_1_prop }
309       { ClassMap }
310       { \pdf_object_ref:n { __tag/tree/classmap } }
311     }
312   }
```

(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
```

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

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

```
\__tag_tree_write_namespaces:
```

```
314 \cs_new_protected:Npn __tag_tree_write_namespaces:
315   {
316     \pdf_version_compare:NnF < {2.0}
```

```

317 {
318   \prop_map_inline:Nn \g__tag_role_NS_prop
319   {
320     \pdfdict_if_empty:nF {g__tag_role/RoleMapNS_##1_dict}
321     {
322       \pdf_object_write:nne {__tag/RoleMapNS/##1}{dict}
323       {
324         \pdfdict_use:n {g__tag_role/RoleMapNS_##1_dict}
325         }
326       \pdfdict_gput:nne{g__tag_role/namespace_##1_dict}
327       {RoleMapNS}{\pdf_object_ref:n {__tag/RoleMapNS/##1}}
328       }
329     \pdf_object_write:nne{tag/NS/##1}{dict}
330     {
331       \pdfdict_use:n {g__tag_role/namespace_##1_dict}
332       }
333     }
334     \pdf_object_write:nne {__tag/tree/namespaces}{array}
335     {
336       \prop_map_tokens:Nn \g__tag_role_NS_prop{\use_ii:n}
337     }
338   }
339 }

```

(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:

```

340 \hook_new:n {tagpdf/finish/before}
341 \cs_new_protected:Npn \__tag_finish_structure:
342 {
343   \bool_if:NT\g__tag_active_tree_bool
344   {
345     \hook_use:n {tagpdf/finish/before}
346     \__tag_tree_final_checks:
347     \iow_term:n{Package~tagpdf~Info:~writing~ParentTree}
348     \__tag_check_benchmark_tic:
349     \__tag_tree_write_parenttree:
350     \__tag_check_benchmark_toc:
351     \iow_term:n{Package~tagpdf~Info:~writing~IDTree}
352     \__tag_check_benchmark_tic:
353     \__tag_tree_write_idtree:
354     \__tag_check_benchmark_toc:
355     \iow_term:n{Package~tagpdf~Info:~writing~RoleMap}
356     \__tag_check_benchmark_tic:
357     \__tag_tree_write_rolemap:
358     \__tag_check_benchmark_toc:
359     \iow_term:n{Package~tagpdf~Info:~writing~ClassMap}
360     \__tag_check_benchmark_tic:
361     \__tag_tree_write_classmap:

```

```

362     \__tag_check_benchmark_toc:
363     \iow_term:n{Package~tagpdf~Info:~writing~NameSpaces}
364     \__tag_check_benchmark_tic:
365     \__tag_tree_write_namespaces:
366     \__tag_check_benchmark_toc:
367     \iow_term:n{Package~tagpdf~Info:~writing~StructElems}
368     \__tag_check_benchmark_tic:
369     \__tag_tree_write_structelements: %this is rather slow!!
370     \__tag_check_benchmark_toc:
371     \iow_term:n{Package~tagpdf~Info:~writing~Root}
372     \__tag_check_benchmark_tic:
373     \__tag_tree_write_structtreeroot:
374     \__tag_check_benchmark_toc:
375   }
376 }
377 </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.

```

378 <*package>
379 \hook_gput_code:nnn{begindocument}{tagpdf}
380 {
381   \bool_if:NT\g__tag_active_tree_bool
382   {
383     \hook_gput_code:nnn{shipout/before} { tagpdf/structparents }
384     {
385       \pdfmanagement_add:nne
386       { Page }
387       { StructParents }
388       { \int_eval:n { \g_shipout_readonly_int} }
389     }
390   }
391 }
392 </package>

```

Part V

The `tagpdf-mc-shared` module Code related to Marked Content (mc-chunks), code shared by all modes

Part of the `tagpdf` package

1 Public Commands

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

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

```
\tag_mc_use:n  \tag_mc_use:n {<label>}
```

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

```
\tag_mc_artifact_group_begin:n \tag_mc_artifact_group_begin:n {<name>}  
\tag_mc_artifact_group_end:    \tag_mc_artifact_group_end:
```

New: 2019-11-20

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

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

New: 2021-04-22

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

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

`\tag_mc_reset_box:N` ★ `\tag_mc_reset_box:N` $\langle box \rangle$

New: 2023-06-11 This resets in lua mode the mc attributes to the one currently in use. It does nothing in generic mode.

`\tag_mc_add_missing_to_stream:Nn` `\tag_mc_add_missing_to_stream:Nn` $\langle box \rangle$ $\{ \langle stream name \rangle \}$

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.

`\tag_mc_new_stream:n` `\tag_mc_new_stream:n` $\{ \langle stream name \rangle \}$

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`,

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

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

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

`alt` (mc-key) This key inserts an `/Alt` value in the property dictionary of the BDC operator. The value is handled as verbatim string, commands are not expanded. The value will be expanded first once. If it is empty, nothing will happen.

`actualtext` (mc-key) This key inserts an `/ActualText` value in the property dictionary of the BDC operator. The value is handled as verbatim string, commands are not expanded. The value will be expanded first once. If it is empty, nothing will happen.

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

stash (mc-key) This “stashes” an mc-chunk: it is not inserted into the current structure. It should be normally be used along with a label to be able to use the mc-chunk in another place.

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

3 Marked content code – shared

```

1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-mc-code-shared} {2025-01-12} {0.991}
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 `\c1@ckpt` and restored e.g. in `tabulars` and `align`. `\int_new:N \c@g_@@_MCID_abs_int` and `\tl_put_right:Nn\c1@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_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`

22 \tl_new:N \l__tag_mc_key_tag_tl

23 \tl_new:N \g__tag_mc_key_tag_tl

24 \tl_new:N \l__tag_mc_key_label_tl

25 \tl_new:N \l__tag_mc_key_properties_tl

(End of definition for \l__tag_mc_key_tag_tl and others.)

3.2 Functions

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

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

`tagmcabs`: the absolute mc-counter `\c@g_@@_MCID_abs_int`. The reference command is based on `l3ref`.

26 \cs_new:Npn __tag_mc_handle_mc_label:e #1

27 {

28 __tag_property_record:en{tagpdf-#1}{tagabspage,tagmcabs}

29 }

(End of definition for __tag_mc_handle_mc_label:e)

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

```

30 \cs_new_protected:Npn \_tag_mc_set_label_used:n #1 % #1 labelname
31 {
32   \tl_new:c { g__tag_mc_label_\tl_to_str:n{#1}_used_tl }
33 }
34 </shared>

```

(End of definition for `_tag_mc_set_label_used:n`.)

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

TODO: is testing for struct the right test?

```

35 <base>\cs_new_protected:Npn \tag_mc_use:n #1 { \_tag_whatsits: }
36 <*shared>
37 \cs_set_protected:Npn \tag_mc_use:n #1 % #1: label name
38 {
39   \_tag_check_if_active_struct:T
40   {
41     \tl_set:Nc \l__tag_tmpa_tl { \property_ref:nnn{tagpdf-#1}{tagmcabs}{ } }
42     \tl_if_empty:NTF\l__tag_tmpa_tl
43     {
44       \msg_warning:nnn {tag} {mc-label-unknown} {#1}
45     }
46     {
47       \cs_if_free:cTF { g__tag_mc_label_\tl_to_str:n{#1}_used_tl }
48       {
49         \_tag_mc_handle_stash:e { \l__tag_tmpa_tl }
50         \_tag_mc_set_label_used:n {#1}
51       }
52       {
53         \msg_warning:nnn {tag}{mc-used-twice}{#1}
54       }
55     }
56   }
57 }
58 </shared>

```

(End of definition for `\tag_mc_use:n`. This function is documented on page 72.)

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

```

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

```

```

69
70 \cs_set_protected:Npn \tag_mc_artifact_group_end:
71 {
72   \tag_resume:n{artifact-group}
73   \group_end:
74   \tag_mc_end:
75   \tag_mc_begin_pop:n{ }
76 }
77 </shared>

```

(End of definition for \tag_mc_artifact_group_begin:n and \tag_mc_artifact_group_end:. These functions are documented on page 72.)

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

```

78 <base>\cs_new_protected:Npn \tag_mc_reset_box:N #1 { }

```

(End of definition for \tag_mc_reset_box:N. This function is documented on page 73.)

\tag_mc_end_push:
\tag_mc_begin_pop:n

```

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

```

```

114         }
115     }
116     {
117         \__tag_check_mc_pushed_popped:nn {popped}{empty~stack,~nothing}
118     }
119 }
120 }

```

(End of definition for `\tag_mc_end_push:` and `\tag_mc_begin_pop:n`. These functions are documented on page 72.)

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.

`--artifact-bool`
`--artifact-type`

```

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

```

```
156     \tl_set:Nn \l__tag_mc_artifact_type_tl {}  
157     },  
158 }
```

(End of definition for `stash (mc-key)`, `__artifact-bool`, and `__artifact-type`. This function is documented on page 74.)

```
159 </shared>
```

Part VI

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

1 Marked content code – generic mode

```
1 <@@=tag>
2 <*generic>
3 \ProvidesExplPackage {tagpdf-mc-code-generic} {2025-01-12} {0.991}
4 {part of tagpdf - code related to marking chunks - generic mode}
5 </generic>
6 <*debug>
7 \ProvidesExplPackage {tagpdf-debug-generic} {2025-01-12} {0.991}
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 73.)

`\l__tag_mc_firstmarks_seq`
`\l__tag_mc_botmarks_seq`

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

```
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`
`__tag_mc_artifact_begin_marks:n`
`__tag_mc_end_marks:`

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

```
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:n`, `__tag_mc_artifact_begin_marks:n`, and `__tag_mc_end_marks:.`)

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

```

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

```

(End of definition for `__tag_mc_disable_marks:.`)

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

```

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

```

(End of definition for `__tag_mc_get_marks:.`)

`__tag_mc_store:nnn` This inserts the mc-chunk `<mc-num>` into the structure `struct-num` after the `<mc-prev>`. The structure must already exist. The additional mcid dictionary is stored in a property.

The item is retrieved when the kid entry is built. We test if there is already an addition and append if needed.

```

93 \cs_new_protected:Npn \__tag_mc_store:nnn #1 #2 #3 % #1 mc-prev, #2 mc-num #3 structure-
    num
94 {
95   %\prop_show:N \g__tag_struct_cont_mc_prop
96   \prop_get:NnNTF \g__tag_struct_cont_mc_prop {#1} \l__tag_tmpa_tl
97   {
98     \prop_gput:Nne \g__tag_struct_cont_mc_prop {#1}{ \l__tag_tmpa_tl \__tag_struct_mcid_c
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} } = {-
117       1}
118     {
119       \tl_set:Ne \l__tag_tmpa_tl { \seq_item:cn { g__tag_mc_#1_marks_seq } {4} }
120       \__tag_mc_handle_artifact:N \l__tag_tmpa_tl
121     }
122     {
123       \exp_args:Ne
124       \__tag_mc_bdc_mcid:n
125       {
126         \seq_item:cn { g__tag_mc_#1_marks_seq } {4}
127       }
128       \str_if_eq:eeTF
129       {
130         \seq_item:cn { g__tag_mc_#1_marks_seq } {5}
131       }

```

```

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

```

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

```

177   \box_set_ht:Nn \l__tag_tmpa_box \c_zero_dim
178   \box_set_dp:Nn \l__tag_tmpa_box \c_zero_dim

```

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

```

179   \box_set_ht:Nn \l__tag_tmpb_box \c_zero_dim
180   \box_set_dp:Nn \l__tag_tmpb_box { \box_dp:N #1 }

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

\tag_mc_add_missing_to_stream:Nn
\__tag_add_missing_mcs_to_stream:Nn

```

This is the main command to add mc to the stream. It is therefore guarded by the mc-boolean.

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

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

```

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

```

First set up a temp box for trial splitting.

```

192   \vbadness\maxdimen
193   \box_set_eq:NN \l__tag_tmpa_box #1

```

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

```

194   \vbox_set_split_to_ht:NNn \l__tag_tmpa_box \l__tag_tmpa_box \c_max_dim

```

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

```

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

```

Some debugging info:

```

198 %     \iow_term:n { First~ mark~ from~ this~ box: }
199 %     \seq_log:N \l__tag_mc_firstmarks_seq

```

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

```

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

```

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

```

208         \seq_set_eq:NN \l__tag_mc_botmarks_seq \l__tag_mc_firstmarks_seq
209     }

```

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

```

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

```

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

```

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

```

(End of definition for `\tag_mc_add_missing_to_stream:Nn` and `__tag_add_missing_mcs_to_stream:Nn`. This function is documented on page 73.)

`_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:TF`
`\tag_mc_if_in_p:`
`\tag_mc_if_in:TF`

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

```
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 72.)

`_tag_mc_bmc:n`
`_tag_mc_emc:`
`_tag_mc_bdc:nn`

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

```
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:VV`

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

```
239 \bool_if:NTF \g__tag_delayed_shipout_bool
240   {
241     \hook_gput_code:nnn {shipout/before}{tagpdf}{ \flag_clear:n { \_tag/mcid } }
242     \cs_set_protected:Npn \_tag_mc_bdc_mcid:nn #1 #2
243       {
244         \int_gincr:N \c@g__tag_MCID_abs_int
245         \_tag_property_record:eV
246         {
247           mcid-\int_use:N \c@g__tag_MCID_abs_int
248         }
249         \c__tag_property_mc_clist
250         \_tag_mc_bdc_shipout:ee
251         {#1}
252       }

```

```

253         /MCID~\flag_height:n { __tag/mcid }
254         \flag_raise:n { __tag/mcid }~ #2
255     }
256 }
257 }

```

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

```

258 {
259     \msg_new:nnn { tagpdf } { old-engine }
260     {
261         The~engine~or~the~PDF~management~is~too~old~or~\
262         delayed~shipout~has~been~disabled~\
263         Fast~numbering~of~MC~chunks~not~available~\
264         More~compilations~will~be~needed~in~generic~mode.
265     }
266     \msg_warning:nn { tagpdf } { old-engine }
267     \__tag_prop_new:N \g__tag_MCID_byabspage_prop
268     \int_new:N \g__tag_MCID_tmp_bypage_int
269     \cs_generate_variant:Nn \__tag_mc_bdc:nn {ne}

```

revert the attribute:

```

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

```



```

304         {1}
305     }
306     \__tag_property_record:eV
307     {
308         mcid-\int_use:N \c@g__tag_MCID_abs_int
309     }
310     \c__tag_property_mc_clist
311     \__tag_mc_bdc:ne
312     {#1}
313     { /MCID~\int_eval:n { \g__tag_MCID_tmp_bypage_int }~ \exp_not:n { #2 } }
314 }
315 }
316 \cs_new_protected:Npn \__tag_mc_bdc_mcid:n #1
317 {
318     \__tag_mc_bdc_mcid:nn {#1} {}
319 }
320
321 \cs_new_protected:Npn \__tag_mc_handle_mcid:nn #1 #2 %#1 tag, #2 properties
322 {
323     \__tag_mc_bdc_mcid:nn {#1} {#2}
324 }
325
326 \cs_generate_variant:Nn \__tag_mc_handle_mcid:nn {VV}

```

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

__tag_mc_handle_stash:n 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?

```

327 \cs_new_protected:Npn \__tag_mc_handle_stash:n #1 %1 mcidnum
328 {
329     \__tag_check_mc_used:n {#1}
330     \__tag_struct_kid_mc_gput_right:nn
331     { \g__tag_struct_stack_current_tl }
332     {#1}
333     \prop_gput:Nee \g__tag_mc_parenttree_prop
334     {#1}
335     { \g__tag_struct_stack_current_tl }
336 }
337 \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

```

338 \cs_new_protected:Npn \__tag_mc_bmc_artifact:
339 {
340     \__tag_mc_bmc:n {Artifact}
341 }
342 \cs_new_protected:Npn \__tag_mc_bmc_artifact:n #1

```

```

343 {
344   \__tag_mc_bdc:nn {Artifact}{/Type/#1}
345 }
346 \cs_new_protected:Npn \__tag_mc_handle_artifact:N #1
347   % #1 is a var containing the artifact type
348   {
349     \int_gincr:N \c@g__tag_MCID_abs_int
350     \tl_if_empty:NTF #1
351       { \__tag_mc_bmc_artifact: }
352       { \exp_args:NV\__tag_mc_bmc_artifact:n #1 }
353   }

```

(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.

```

354 \cs_new:Nn \__tag_get_data_mc_tag: { \g__tag_mc_key_tag_tl }
355 </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.

```

356 <(base)\cs_new_protected:Npn \tag_mc_begin:n #1 { \__tag_whatsits: \int_gincr:N \c@g__tag_MCID
357 <(base)\cs_new_protected:Nn \tag_mc_end:{ \__tag_whatsits: }
358 <*generic | debug>
359 <*generic>
360 \cs_set_protected:Npn \tag_mc_begin:n #1 %#1 keyval
361 {
362   \__tag_check_if_active_mc:T
363   {
364 </generic>
365 <*debug>
366 \cs_set_protected:Npn \tag_mc_begin:n #1 %#1 keyval
367 {
368   \__tag_check_if_active_mc:TF
369   {
370     \__tag_debug_mc_begin_insert:n { #1 }
371 </debug>
372     \group_begin: %hm
373     \__tag_check_mc_if_nested:
374     \bool_gset_true:N \g__tag_in_mc_bool

```

set default MC tags to structure:

```

375     \tl_set_eq:NN \l__tag_mc_key_tag_tl \g__tag_struct_tag_tl
376     \tl_gset_eq:NN\g__tag_mc_key_tag_tl \g__tag_struct_tag_tl
377     \keys_set:nn { __tag / mc } {#1}
378     \bool_if:NTF \l__tag_mc_artifact_bool
379     { %handle artifact
380       \__tag_mc_handle_artifact:N \l__tag_mc_artifact_type_tl
381       \exp_args:NV
382       \__tag_mc_artifact_begin_marks:n \l__tag_mc_artifact_type_tl
383     }

```

```

384     { %handle mcid type
385         \__tag_check_mc_tag:N \l__tag_mc_key_tag_tl
386         \__tag_mc_handle_mcid:VV
387             \l__tag_mc_key_tag_tl
388             \l__tag_mc_key_properties_tl
389         \__tag_mc_begin_marks:oo{\l__tag_mc_key_tag_tl}{\l__tag_mc_key_label_tl}
390         \tl_if_empty:NF {\l__tag_mc_key_label_tl}
391         {
392             \exp_args:NV
393             \__tag_mc_handle_mc_label:e \l__tag_mc_key_label_tl
394         }
395     \bool_if:NF \l__tag_mc_key_stash_bool
396     {
397         \exp_args:NV\__tag_struct_get_parentrole:nNN
398             \g__tag_struct_stack_current_tl
399             \l__tag_get_parent_tmpa_tl
400             \l__tag_get_parent_tmpb_tl
401         \__tag_check_parent_child:VVnnN
402             \l__tag_get_parent_tmpa_tl
403             \l__tag_get_parent_tmpb_tl
404             {MC}{ }
405             \l__tag_parent_child_check_tl
406         \int_compare:nNnT {\l__tag_parent_child_check_tl}<{0}
407         {
408             \prop_get:cnN
409             { g__tag_struct_ \g__tag_struct_stack_current_tl _prop }
410             { S }
411             \l__tag_tmpa_tl
412             \msg_warning:nneee
413             { tag }
414             { role-parent-child }
415             { \l__tag_get_parent_tmpa_tl/\l__tag_get_parent_tmpb_tl }
416             { MC~(real content) }
417             { not~allowed~
418               (struct~\g__tag_struct_stack_current_tl,~\l__tag_tmpa_tl)
419             }
420         }
421         \__tag_mc_handle_stash:e { \int_use:N \c@g__tag_MCID_abs_int }
422     }
423 }
424 \group_end:
425 }
426 < *debug >
427 {
428     \__tag_debug_mc_begin_ignore:n { #1 }
429 }
430 < /debug >
431 }
432 < *generic >
433 \cs_set_protected:Nn \tag_mc_end:
434 {
435     \__tag_check_if_active_mc:T
436     {
437 < /generic >

```

```

438 <*debug>
439 \cs_set_protected:Nn \tag_mc_end:
440 {
441   \__tag_check_if_active_mc:TF
442   {
443     \__tag_debug_mc_end_insert:
444 </debug>
445     \__tag_check_mc_if_open:
446     \bool_gset_false:N \g__tag_in_mc_bool
447     \tl_gset:Nn \g__tag_mc_key_tag_tl { }
448     \__tag_mc_emc:
449     \__tag_mc_end_marks:
450   }
451 <*debug>
452 {
453   \__tag_debug_mc_end_ignore:
454 }
455 </debug>
456 }
457 </generic | debug>

```

(End of definition for \tag_mc_begin:n and \tag_mc_end:. These functions are documented on page 72.)

1.4 Keys

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

```

tag (mc-key)
raw (mc-key)
alt (mc-key)
actualtext (mc-key)
label (mc-key)
artifact (mc-key)
458 <*generic>
459 \keys_define:nn { __tag / mc }
460 {
461   tag .code:n = % the name (H,P,Span) etc
462   {
463     \tl_set:Ne \l__tag_mc_key_tag_tl { #1 }
464     \tl_gset:Ne \g__tag_mc_key_tag_tl { #1 }
465   },
466   raw .code:n =
467   {
468     \tl_put_right:Ne \l__tag_mc_key_properties_tl { #1 }
469   },
470   alt .code:n = % Alt property
471   {
472     \str_set_convert:Noon
473     \l__tag_tmpa_str
474     { #1 }
475     { default }
476     { utf16/hex }
477     \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
478     \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
479   },
480   alttext .meta:n = {alt=#1},
481   actualtext .code:n = % ActualText property

```

```

482     {
483       \tl_if_empty:oF{#1}
484       {
485         \str_set_convert:Noon
486         \l__tag_tmpa_str
487         { #1 }
488         { default }
489         { utf16/hex }
490         \tl_put_right:Nn \l__tag_mc_key_properties_tl { /ActualText~< }
491         \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
492       }
493     },
494     label .tl_set:N      = \l__tag_mc_key_label_tl,
495     artifact .code:n    =
496     {
497       \exp_args:Nne
498       \keys_set:nn
499       { __tag / mc }
500       { __artifact-bool, __artifact-type=#1 }
501     },
502     artifact .default:n = {notype}
503   }
504 </generic>

```

(End of definition for tag (mc-key) and others. These functions are documented on page 73.)

Part VII

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

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} {2025-01-12} {0.991}
4   {tagpdf - mc code only for the luamode }
5 </luamode>
6 <*debug>
7 \ProvidesExplPackage {tagpdf-debug-lua} {2025-01-12} {0.991}
8   {part of tagpdf - debugging code related to marking chunks - lua mode}
9 </debug>
```

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

```

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

```

```

62     }
63   }
64 }

```

1.1 Commands

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

```

65 \cs_new_protected:Npn \_tag_add_missing_mcs_to_stream:Nn #1#2 {}
66 \cs_set_eq:NN \tag_mc_add_missing_to_stream:Nn \_tag_add_missing_mcs_to_stream:Nn

```

(End of definition for `_tag_add_missing_mcs_to_stream:Nn`.)

`\tag_mc_new_stream:n`

```

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

```

(End of definition for `\tag_mc_new_stream:n`. This function is documented on page 73.)

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

```

(End of definition for `_tag_mc_if_in:TF` and `\tag_mc_if_in:TF`. This function is documented on page 72.)

`_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:Ne\l__tag_tmpa_tl{\lua_now:e{ltx.__tag.func.output_num_from ("#1")} }
87   \lua_now:e
88   {
89     tex.setattribute
90     (
91       "global",
92       luatexbase.attributes.g__tag_mc_type_attr,
93       \l__tag_tmpa_tl
94     )
95   }
96   \lua_now:e
97   {

```



```

98         tex.setattribute
99         (
100            "global",
101            luatexbase.attributes.g__tag_mc_cnt_attr,
102            \__tag_get_mc_abs_cnt:
103        )
104     }
105 }
106
107 \cs_generate_variant:Nn\__tag_mc_lua_set_mc_type_attr:n { o }
108
109 \cs_new:Nn \__tag_mc_lua_unset_mc_type_attr:
110 {
111     \lua_now:e
112     {
113         tex.setattribute
114         (
115             "global",
116             luatexbase.attributes.g__tag_mc_type_attr,
117             -2147483647
118         )
119     }
120     \lua_now:e
121     {
122         tex.setattribute
123         (
124             "global",
125             luatexbase.attributes.g__tag_mc_cnt_attr,
126             -2147483647
127         )
128     }
129 }
130

```

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

__tag_mc_insert_mcid_kids:n 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     MC~#1%
157   }
158   \lua_now:e
159   {
160     ltx.__tag.func.store_struct_mcab
161     (
162       \g__tag_struct_stack_current_tl,#1
163     )
164   }
165   \prop_gput:Nee
166   \g__tag_mc_parenttree_prop
167   { #1 }
168   { \g__tag_struct_stack_current_tl }
169 }
170 </luamode | debug>
171 <*luamode>
172 \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.

```

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

```

set the default tag to the structure:

```

183     \tl_set_eq:NN \l__tag_mc_key_tag_tl \g__tag_struct_tag_tl
184     \tl_gset_eq:NN\g__tag_mc_key_tag_tl \g__tag_struct_tag_tl
185     \lua_now:e
186     {
187       ltx.__tag.func.store_mc_data(\__tag_get_mc_abs_cnt:,"tag","\g__tag_struct_tag_tl
188     }
189     \keys_set:nn { __tag / mc }{ label={}, #1 }

```

```

190 %check that a tag or artifact has been used
191 \__tag_check_mc_tag:N \l__tag_mc_key_tag_tl
192 %set the attributes:
193 \__tag_mc_lua_set_mc_type_attr:o { \l__tag_mc_key_tag_tl }
194 \bool_if:NF \l__tag_mc_artifact_bool
195 { % store the absolute num name in a label:
196   \tl_if_empty:NF {\l__tag_mc_key_label_tl}
197   {
198     \exp_args:NV
199     \__tag_mc_handle_mc_label:e \l__tag_mc_key_label_tl
200   }
201 % if not stashed record the absolute number
202 \bool_if:NF \l__tag_mc_key_stash_bool
203 {
204   \exp_args:NV\__tag_struct_get_parentrole:nNN
205   \g__tag_struct_stack_current_tl
206   \l__tag_get_parent_tmpa_tl
207   \l__tag_get_parent_tmpb_tl
208   \__tag_check_parent_child:VVnnN
209   \l__tag_get_parent_tmpa_tl
210   \l__tag_get_parent_tmpb_tl
211   {MC}{ }
212   \l__tag_parent_child_check_tl
213   \int_compare:nNnT {\l__tag_parent_child_check_tl}<{0}
214   {
215     \prop_get:cnN
216     { g__tag_struct_ \g__tag_struct_stack_current_tl _prop}
217     {S}
218     \l__tag_tmpa_tl
219     \msg_warning:nneee
220     { tag }
221     {role-parent-child}
222     { \l__tag_get_parent_tmpa_tl/\l__tag_get_parent_tmpb_tl }
223     { MC-(real content) }
224     {
225       not-allowed~
226       (struct~\g__tag_struct_stack_current_tl,~\l__tag_tmpa_tl)
227     }
228   }
229   \__tag_mc_handle_stash:e { \__tag_get_mc_abs_cnt: }
230 }
231 }
232 \group_end:
233 }
234 }

```

(End of definition for \tag_mc_begin:n. This function is documented on page 72.)

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

```

235 \cs_set_protected:Nn \tag_mc_end:
236 {
237   \__tag_check_if_active_mc:T
238   {
239     %\__tag_check_mc_if_open:

```

```

240     \bool_gset_false:N \g__tag_in_mc_bool
241     \bool_set_false:N\l__tag_mc_artifact_bool
242     \__tag_mc_lua_unset_mc_type_attr:
243     \tl_set:Nn \l__tag_mc_key_tag_tl { }
244     \tl_gset:Nn \g__tag_mc_key_tag_tl { }
245   }
246 }

```

(End of definition for `\tag_mc_end:`. This function is documented on page 72.)

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

```

247 \cs_set_protected:Npn \tag_mc_reset_box:N #1
248 {
249   \lua_now:e
250   {
251     local~type=tex.getattribute(luatexbase.attributes.g__tag_mc_type_attr)
252     local~mc=tex.getattribute(luatexbase.attributes.g__tag_mc_cnt_attr)
253     ltx.__tag.func.update_mc_attributes(tex.getbox(\int_use:N #1),mc,type)
254   }
255 }

```

(End of definition for `\tag_mc_reset_box:N`. This function is documented on page 73.)

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

```

256 \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)   257 \keys_define:nn { __tag / mc }
alt (mc-key)   258 {
actualtext (mc-key) 259   tag .code:n = %
label (mc-key)    260   {
artifact (mc-key)  261     \tl_set:Ne \l__tag_mc_key_tag_tl { #1 }
                  262     \tl_gset:Ne \g__tag_mc_key_tag_tl { #1 }
                  263     \lua_now:e
                  264     {
                  265       ltx.__tag.func.store_mc_data(\__tag_get_mc_abs_cnt:,"tag", "#1")
                  266     }
                  267   },
raw .code:n =   268   {
alt (mc-key)   269   {
                270     \tl_put_right:Ne \l__tag_mc_key_properties_tl { #1 }
                271     \lua_now:e
                272     {
                273       ltx.__tag.func.store_mc_data(\__tag_get_mc_abs_cnt:,"raw", "#1")
                274     }
                275   },
alt .code:n    276   = % Alt property
                277   {

```

```

278     \tl_if_empty:oF{#1}
279     {
280         \str_set_convert:Noon
281         \l__tag_tmpa_str
282         { #1 }
283         { default }
284         { utf16/hex }
285         \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
286         \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
287         \lua_now:e
288         {
289             ltx.__tag.func.store_mc_data
290             (
291                 \__tag_get_mc_abs_cnt:,"alt","/Alt~<\str_use:N \l__tag_tmpa_str>"
292             )
293         }
294     }
295 },
296 alttext .meta:n = {alt=#1},
297 actualtext .code:n = % Alt property
298 {
299     \tl_if_empty:oF{#1}
300     {
301         \str_set_convert:Noon
302         \l__tag_tmpa_str
303         { #1 }
304         { default }
305         { utf16/hex }
306         \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
307         \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
308         \lua_now:e
309         {
310             ltx.__tag.func.store_mc_data
311             (
312                 \__tag_get_mc_abs_cnt:,
313                 "actualtext",
314                 "/ActualText~<\str_use:N \l__tag_tmpa_str>"
315             )
316         }
317     }
318 },
319 label .code:n =
320 {
321     \tl_set:Nn\l__tag_mc_key_label_tl { #1 }
322     \lua_now:e
323     {
324         ltx.__tag.func.store_mc_data
325         (
326             \__tag_get_mc_abs_cnt:,"label","#1"
327         )
328     }
329 },
330 __artifact-store .code:n =
331 {

```

```

332     \lua_now:e
333     {
334         ltx.__tag.func.store_mc_data
335         (
336             \__tag_get_mc_abs_cnt:,"artifact","#1"
337         )
338     }
339 },
340 artifact .code:n      =
341 {
342     \exp_args:Nne
343     \keys_set:nn
344     { __tag / mc }
345     { __artifact-bool, __artifact-type=#1, tag=Artifact }
346     \exp_args:Nne
347     \keys_set:nn
348     { __tag / mc }
349     { __artifact-store=\l__tag_mc_artifact_type_tl }
350 },
351 artifact .default:n   = { notype }
352 }
353
354 </luamode>

```

(End of definition for tag (mc-key) and others. These functions are documented on page 73.)

Part VIII

The tagpdf-struct module

Commands to create the structure Part of the tagpdf package

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.

`\tag_struct_gput:nnn` `\tag_struct_gput:nnn` `{<structure number>}` `{<keyword>}` `{<value>}`

This is a command that allows to update the data of a structure. This often can't 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 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.

firstkey (*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 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*) **AF** = *<object name>*
- AFinline-o** (*struct key*) **AFref** = *<object reference>*
- texsource** (*struct key*) **AF-inline** = *<text content>*
- mathml** (*struct key*)
- The value *<object name>* should be the name of an object pointing to the `/Filespec` dictionary as expected by `\pdf_object_ref:n` from a current `l3kernel`.
- The value **AF-inline** is some text, which is embedded in the PDF as a text file with mime type `text/plain`. **AF-inline-o** is like **AF-inline** but expands the value once.
- Future versions will perhaps extend this to more mime types, but it is still a research task to find out what is really needed.
- texsource** is a special variant of **AF-inline-o** which embeds the content as `.tex` source with the `/AFrelationship` key set to `/Source`. It also sets the `/Desc` key to a (currently) fix text.
- mathml** is a special variant of **AF-inline-o** which embeds the content as `.xml` file with the `/AFrelationship` key set to `/Supplement`. It also sets the `/Desc` key to a (currently) fix text.
- The argument of **AF** is an object name referring an embedded file as declared for example with `\pdf_object_new:n` or with the `l3pdffile` module. **AF** expands its argument (this allows e.g. to use some variable for automatic numbering) and can be used more than once, to associate more than one file.
- The argument of **AFref** is an object reference to an embedded file or a variable expanding to such a object reference in the format as you would get e.g. from `\pdf_object_ref_last:` or `\pdf_object_ref:n` (and which is different for the various engines!). The key allows to make use of anonymous objects. Like **AF** the **AFref** key expands its argument and can be used more than once, to associate more than one file. *It does not check if the reference is valid!*
- The inline keys can be used only once per structure. Additional calls are ignored.
- attribute** (*struct key*) This key takes as argument a comma list of attribute names (use braces to protect the commas from the external key-val parser) and allows to add one or more attribute dictionary entries in the structure object. As an example

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

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

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

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

2.2 Setup keys

```
role/new-attribute (setup-key) role/new-attribute = {<name>}{<Content>}
newattribute (deprecated)
```

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

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

root-AF (*setup key*) `root-AF = <object name>`

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

```
1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-struct-code} {2025-01-12} {0.991}
4 {part of tagpdf - code related to storing structure}
5 </header>
```

3 Variables

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

```
6 <base>\newcounter { g__tag_struct_abs_int }
7 <base>\int_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`.
`\l__tag_struct_stack_parent_tmpa_tl` The local temporary variable will hold the parent when we fetch it from the stack.

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

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

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

The data of the StructTreeRoot and the StructElem are in properties: `\g__@@_struct_1_prop` for the root and `\g__@@_struct_N_prop`, $N \geq 2$ for the other.

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

All properties have at least the keys

Type StructTreeRoot or StructElem

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

`\c__tag_struct_StructTreeRoot_entries_seq`
`\c__tag_struct_StructElem_entries_seq`

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

```

21 \seq_const_from_clist:Nn \c__tag_struct_StructTreeRoot_entries_seq
22   {%p. 857/858
23     Type,           % always /StructTreeRoot
24     K,             % kid, dictionary or array of dictionaries
25     IDTree,        % currently unused
26     ParentTree,    % required,obj ref to the parent tree
27     ParentTreeNextKey, % optional
28     RoleMap,
29     ClassMap,
30     Namespaces,
31     AF             %pdf 2.0
32   }
33
34 \seq_const_from_clist:Nn \c__tag_struct_StructElem_entries_seq
35   {%p 858 f
36     Type,           %always /StructElem
37     S,             %tag/type
38     P,             %parent
39     ID,            %optional
40     Ref,           %optional, pdf 2.0 Use?
41     Pg,           %obj num of starting page, optional
42     K,            %kids
43     A,            %attributes, probably unused
44     C,            %class ""
45     %R,           %attribute revision number, irrelevant for us as we
46                 % don't update/change existing PDF and (probably)
47                 % deprecated in PDF 2.0
48     T,            %title, value in () or <>
49     Lang,         %language
50     Alt,          % value in () or <>
51     E,            % abbreviation
52     ActualText,
53     AF,           %pdf 2.0, array of dict, associated files
54     NS,           %pdf 2.0, dict, namespace
55     PhoneticAlphabet, %pdf 2.0
56     Phoneme       %pdf 2.0
57   }

```

(End of definition for `\c__tag_struct_StructTreeRoot_entries_seq` and `\c__tag_struct_StructElem_entries_seq`.)

3.1 Variables used by the keys

`\g__tag_struct_tag_tl`
`\g__tag_struct_tag_NS_tl`
`\l__tag_struct_roletag_tl`
`\g__tag_struct_roletag_NS_tl`

Use by the tag key to store the tag and the namespace. The role tag variables will hold locally rolemapping info needed for the parent-child checks

```

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

```

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

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

```

62 \prop_new_linked:N \g__tag_struct_label_num_prop

```

(End of definition for `\g__tag_struct_label_num_prop`.)

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

```

63 \bool_new:N \l__tag_struct_elem_stash_bool

```

(End of definition for `\l__tag_struct_elem_stash_bool`.)

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

```

64 \tl_new:N \l__tag_struct_addkid_tl
65 \tl_set:Nn \l__tag_struct_addkid_tl {right}

```

(End of definition for `\l__tag_struct_addkid_tl`.)

3.2 Variables used by tagging code of basic elements

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

```

66 \package
67 \prop_new_linked:N \g__tag_struct_dest_num_prop
68 \*package

```

(End of definition for `\g__tag_struct_dest_num_prop`.)

`\g__tag_struct_ref_by_dest_prop` This variable contains structures whose Ref key should be updated at the end to point to structured related with this destination. As this is probably need in other places too, it is not only a toc-variable. TODO: remove after 11/2024 release.

```

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

```

```

77     }
78   }
79
80 \cs_new_protected:Npn \__tag_new_output_prop_handler:n #1
81   {
82     \cs_new:cn { \__tag_struct_output_prop_#1:n }
83     {
84       \__tag_struct_output_prop_aux:nn {#1}{#1}
85     }
86   }
87 \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:

```

88 \begin{package} \debug
89 \package\cs_new_protected:Npn \__tag_struct_prop_gput:nnn #1 #2 #3
90 \debug\cs_set_protected:Npn \__tag_struct_prop_gput:nnn #1 #2 #3
91   {
92     \__tag_prop_gput:cnn
93     { g__tag_struct_#1_prop }{#2}{#3}
94 \debug\prop_gput:cnn { g__tag_struct_debug_#1_prop } {#2} {#3}
95   }
96 \cs_generate_variant:Nn \__tag_struct_prop_gput:nnn {onn,nne,nee,nno}
97 \endpackage \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.

```

98 \begin{package}
99 \tl_gset:Nn \g__tag_struct_stack_current_tl {1}

```

__tag_pdf_name_e:n

```

100 \cs_new:Npn \__tag_pdf_name_e:n #1{\pdf_name_from_unicode_e:n{#1}}
101 \endpackage

```

(End of definition for __tag_pdf_name_e:n.)

g__tag_struct_1_prop
g__tag_struct_kids_1_seq

```

102 \begin{package}
103 \__tag_prop_new:c { g__tag_struct_1_prop }
104 \__tag_new_output_prop_handler:n {1}
105 \__tag_seq_new:c { g__tag_struct_kids_1_seq }
106
107 \__tag_struct_prop_gput:nne
108   { 1 }
109   { Type }

```

```

110 { \pdf_name_from_unicode_e:n {StructTreeRoot} }
111
112 \__tag_struct_prop_gput:nne
113 { 1 }
114 { S }
115 { \pdf_name_from_unicode_e:n {StructTreeRoot} }
116
117 \__tag_struct_prop_gput:nne
118 { 1 }
119 { rolemap }
120 { {StructTreeRoot}{pdf} }
121
122 \__tag_struct_prop_gput:nne
123 { 1 }
124 { parentrole }
125 { {StructTreeRoot}{pdf} }
126

```

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

```

127 \pdf_version_compare:NnF < {2.0}
128 {
129   \__tag_struct_prop_gput:nne
130   { 1 }
131   { Namespaces }
132   { \pdf_object_ref:n { __tag/tree/namespaces } }
133 }
134 </package>

```

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

```

135 <debug>\prop_new:c { g__tag_struct_debug_1_prop }
136 <debug>\seq_new:c { g__tag_struct_debug_kids_1_seq }
137 <debug>\prop_gset_eq:cc { g__tag_struct_debug_1_prop }{ g__tag_struct_1_prop }
138 <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
139 <*package>
140 \cs_new:Npn \__tag_struct_get_id:n #1 %#1=struct num
141 {
142   (
143     ID.
144     \prg_replicate:nn
145     { \int_abs:n{\g__tag_tree_id_pad_int - \tl_count:e { \int_to_arabic:n { #1 } }} }
146     { 0 }
147     \int_to_arabic:n { #1 }
148   )
149 }

```

(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.

```

150 \pdf_version_compare:NnTF < {2.0}
151 {
152   \cs_new_protected:Npn \_tag_struct_set_tag_info:nnn #1 #2 #3
153     %#1 structure number, #2 tag, #3 NS
154     {
155       \_tag_struct_prop_gput:nne
156         { #1 }
157         { S }
158         { \pdf_name_from_unicode_e:n {#2} } %
159     }
160 }
161 {
162   \cs_new_protected:Npn \_tag_struct_set_tag_info:nnn #1 #2 #3
163     {
164       \_tag_struct_prop_gput:nne
165         { #1 }
166         { S }
167         { \pdf_name_from_unicode_e:n {#2} } %
168       \prop_get:NnNT \g__tag_role_NS_prop {#3} \l__tag_get_tmpc_tl
169       {
170         \_tag_struct_prop_gput:nne
171           { #1 }
172           { NS }
173           { \l__tag_get_tmpc_tl } %
174       }
175     }
176 }
177 \cs_generate_variant:Nn \_tag_struct_set_tag_info:nnn {eVV}

```

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

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

```

178 \cs_new_protected:Npn \_tag_struct_get_parentrole:nNN #1 #2 #3
179   %#1 struct num, #2 tlvvar for tag , #3 tlvvar for NS
180   {
181     \prop_get:cnNTF
182       { g__tag_struct_#1_prop }
183       { parentrole }
184       \l__tag_get_tmpc_tl
185       {
186         \tl_set:Ne #2{\exp_last_unbraced:NV\use_i:nn \l__tag_get_tmpc_tl}
187         \tl_set:Ne #3{\exp_last_unbraced:NV\use_ii:nn \l__tag_get_tmpc_tl}
188       }
189       {
190         \tl_clear:N#2
191         \tl_clear:N#3
192       }
193   }
194 \cs_generate_variant:Nn \_tag_struct_get_parentrole:nNN {eNN}

```


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

4.4 Handlings kids

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

`__tag_struct_kid_mc_gput_right:nn`
`__tag_struct_kid_mc_gput_right:ne`

The command to store an mc-chunk, this is a dictionary of type MCR. It would be possible to write out the content directly as unnamed object and to store only the object reference, but probably this would be slower, and the PDF is more readable like this. The code doesn't try to avoid the use of the /Pg key by checking page numbers. That imho only slows down without much gain. In generic mode the page break code will perhaps 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!

```
195 \cs_new:Npn \__tag_struct_mcid_dict:n #1 %#1 MCID absnum
196   {
197     <<
198     /Type \c_space_tl /MCR \c_space_tl
199     /Pg
200     \c_space_tl
201     \pdf_pageobject_ref:n { \property_ref:enn{mcid-#1}{tagabspage}{1} }
202     /MCID \c_space_tl \property_ref:enn{mcid-#1}{tagmcid}{1}
203     >>
204   }
205 (/package)
206 (*package | debug)
207 (package)\cs_new_protected:Npn \__tag_struct_kid_mc_gput_right:nn #1 #2
208 (debug)\cs_set_protected:Npn \__tag_struct_kid_mc_gput_right:nn #1 #2
209 %#1 structure num, #2 MCID absnum%
210   {
211     \__tag_seq_gput_right:ce
212     { g__tag_struct_kids_#1_seq }
213     {
214       \__tag_struct_mcid_dict:n {#2}
215     }
216 (debug)   \seq_gput_right:cn
217 (debug)   { g__tag_struct_debug_kids_#1_seq }
218 (debug)   {
219 (debug)     MC~#2
220 (debug)   }
221   \__tag_seq_gput_right:cn
222   { g__tag_struct_kids_#1_seq }
223   {
224     \prop_item:Nn \g__tag_struct_cont_mc_prop {#2}
225   }
226 }
227 (package)\cs_generate_variant:Nn \__tag_struct_kid_mc_gput_right:nn {ne}
```

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

`__tag_struct_kid_struct_gput_right:nn`
`__tag_struct_kid_struct_gput_right:ee`

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

```

228 <package>\cs_new_protected:Npn\__tag_struct_kid_struct_gput_right:nn #1 #2
229 <debug>\cs_set_protected:Npn\__tag_struct_kid_struct_gput_right:nn #1 #2
230 %%#1 num of parent struct, #2 kid struct
231 {
232   \__tag_seq_gput_right:ce
233   { g__tag_struct_kids_#1_seq }
234   {
235     \pdf_object_ref_indexed:nn { __tag/struct }{ #2 }
236   }
237 <debug>   \seq_gput_right:cn
238 <debug>   { g__tag_struct_debug_kids_#1_seq }
239 <debug>   {
240 <debug>     Struct~#2
241 <debug>   }
242 }
243 <package>\cs_generate_variant:Nn \__tag_struct_kid_struct_gput_right:nn {ee}

```

(End of definition for __tag_struct_kid_struct_gput_right:nn.)

__tag_struct_kid_struct_gput_left:nn
 __tag_struct_kid_struct_gput_left:ee

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

```

244 <package>\cs_new_protected:Npn\__tag_struct_kid_struct_gput_left:nn #1 #2
245 <debug>\cs_set_protected:Npn\__tag_struct_kid_struct_gput_left:nn #1 #2
246 %%#1 num of parent struct, #2 kid struct
247 {
248   \__tag_seq_gput_left:ce
249   { g__tag_struct_kids_#1_seq }
250   {
251     \pdf_object_ref_indexed:nn { __tag/struct }{ #2 }
252   }
253 <debug>   \seq_gput_left:cn
254 <debug>   { g__tag_struct_debug_kids_#1_seq }
255 <debug>   {
256 <debug>     Struct~#2
257 <debug>   }
258 }
259 <package>\cs_generate_variant:Nn \__tag_struct_kid_struct_gput_left:nn {ee}

```

(End of definition for __tag_struct_kid_struct_gput_left:nn.)

__tag_struct_kid_OBJR_gput_right:nnn
 __tag_struct_kid_OBJR_gput_right:eee

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

```

260 <package>\cs_new_protected:Npn\__tag_struct_kid_OBJR_gput_right:nnn #1 #2 #3
261 <package>
262 <package>
263 <debug>\cs_set_protected:Npn\__tag_struct_kid_OBJR_gput_right:nnn #1 #2 #3
264 %%#1 num of parent struct,#2 obj reference,#3 page object reference
265 {
266   \pdf_object_unnamed_write:nn
267   { dict }
268   {
269     /Type/OBJR/Obj~#2/Pg~#3
270   }

```

```

271 \__tag_seq_gput_right:ce
272 { g__tag_struct_kids_#1_seq }
273 {
274   \pdf_object_ref_last:
275 }
276 <debug> \seq_gput_right:ce
277 <debug> { g__tag_struct_debug_kids_#1_seq }
278 <debug> {
279 <debug>   OBJR~reference
280 <debug> }
281 }
282 </package | debug>
283 <*package>
284 \cs_generate_variant:Nn\__tag_struct_kid_OBJR_gput_right:nnn { eee }

```

(End of definition for __tag_struct_kid_OBJR_gput_right:nnn.)

__tag_struct_exchange_kid_command:N
 __tag_struct_exchange_kid_command:c

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

```

285 \cs_new_protected:Npn\__tag_struct_exchange_kid_command:N #1 % #1 = seq var
286 {
287   \seq_gpop_left:NN #1 \l__tag_tmpa_tl
288   \tl_replace_once:Nnn \l__tag_tmpa_tl
289   {\__tag_mc_insert_mcid_kids:n}
290   {\__tag_mc_insert_mcid_single_kids:n}
291   \seq_gput_left:NV #1 \l__tag_tmpa_tl
292 }
293
294 \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.

```

295 \cs_new_protected:Npn\__tag_struct_fill_kid_key:n #1 % #1 is the struct num
296 {
297   \bool_if:NF\g__tag_mode_lua_bool
298   {
299     \seq_clear:N \l__tag_tmpa_seq
300     \seq_map_inline:cn { g__tag_struct_kids_#1_seq }
301     { \seq_put_right:Ne \l__tag_tmpa_seq { ##1 } }
302     %\seq_show:c { g__tag_struct_kids_#1_seq }
303     %\seq_show:N \l__tag_tmpa_seq
304     \seq_remove_all:Nn \l__tag_tmpa_seq {}
305     %\seq_show:N \l__tag_tmpa_seq
306     \seq_gset_eq:cN { g__tag_struct_kids_#1_seq } \l__tag_tmpa_seq
307   }
308
309   \int_case:nnF
310   {
311     \seq_count:c
312     {

```

```

313         g__tag_struct_kids_#1_seq
314     }
315 }
316 {
317     { 0 }
318     { } %no kids, do nothing
319     { 1 } % 1 kid, insert
320     {
321         % in this case we need a special command in
322         % luamode to get the array right. See issue #13
323         \bool_if:NTF\g__tag_mode_lua_bool
324         {
325             \__tag_struct_exchange_kid_command:c
326             {g__tag_struct_kids_#1_seq}

```

check if we get null

```

327         \tl_set:Nc\l__tag_tmpa_tl
328         {\use:e{\seq_item:cn {g__tag_struct_kids_#1_seq} {1}}}
329         \tl_if_eq:NNF\l__tag_tmpa_tl \c__tag_struct_null_tl
330         {
331             \__tag_struct_prop_gput:nne
332             {#1}
333             {K}
334             {
335                 \seq_item:cn
336                 {
337                     g__tag_struct_kids_#1_seq
338                 }
339                 {1}
340             }
341         }
342     }
343     {
344         \__tag_struct_prop_gput:nne
345         {#1}
346         {K}
347         {
348             \seq_item:cn
349             {
350                 g__tag_struct_kids_#1_seq
351             }
352             {1}
353         }
354     }
355 } %
356 }
357 { %many kids, use an array
358     \__tag_struct_prop_gput:nne
359     {#1}
360     {K}
361     {
362         [
363             \seq_use:cn
364             {

```

```

365         g__tag_struct_kids_#1_seq
366         }
367         {
368         \c_space_tl
369         }
370     ]
371 }
372 }
373 }
374

```

(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.

```

375 \cs_new_protected:Npn \__tag_struct_get_dict_content:nN #1 #2 %#1: structure num
376 {
377     \tl_clear:N #2
378     \prop_map_inline:cn { g__tag_struct_#1_prop }
379     {

```

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.

```

380         \cs_if_exist_use:cTF {__tag_struct_format_##1:nnN}
381         {
382             {##1}{##2}#2
383         }
384         {
385             \tl_put_right:Ne #2 { \c_space_tl/##1-##2 }
386         }
387     }
388 }

```

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

`__tag_struct_format_rolemap:nnN` This two entries should not end in the PDF.
`__tag_struct_format_parentrole:nnN`

```

389 \cs_new:Nn\__tag_struct_format_rolemap:nnN{}
390 \cs_new:Nn\__tag_struct_format_parentrole:nnN{}

```

(End of definition for `__tag_struct_format_rolemap:nnN` and `__tag_struct_format_parentrole: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.

```

391 \cs_new_protected:Nn\__tag_struct_format_Ref:nnN
392 {
393     \tl_put_right:Nn #3 { ~/#1~[ ] %]
394     \clist_map_inline:nn{ #2 }
395     {
396         ##1 #3
397     }
398     \tl_put_right:Nn #3
399     { %[

```

```

400     \c_space_tl]
401   }
402 }

```

(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.

```

403 \cs_new_protected:Npn \_tag_struct_write_obj:n #1 % #1 is the struct num
404 {
405   \prop_if_exist:cTF { g__tag_struct_#1_prop }
406   {

```

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.

```

407     \prop_get:cnNF { g__tag_struct_#1_prop } {P}\l__tag_tmpb_tl
408     {
409       \prop_gput:cne { g__tag_struct_#1_prop } {P}
410       {\pdf_object_ref_indexed:nn { __tag/struct }{1}}
411       \prop_gput:cne { g__tag_struct_#1_prop } {S}{/Artifact}
412       \seq_if_empty:cF {g__tag_struct_kids_#1_seq}
413       {
414         \msg_warning:nnee
415         {tag}
416         {struct-orphan}
417         { #1 }
418         {\seq_count:c{g__tag_struct_kids_#1_seq}}
419       }
420     }
421     \_tag_struct_fill_kid_key:n { #1 }
422     \_tag_struct_get_dict_content:nN { #1 } \l__tag_tmpa_tl
423     \pdf_object_write_indexed:nnne
424     { __tag/struct }{ #1 }
425     {dict}
426     {
427       \l__tag_tmpa_tl\c_space_tl
428       /ID-\_tag_struct_get_id:n{#1}
429     }
430   }
431 }
432 {
433   \msg_error:nnn { tag } { struct-no-objnum } { #1}
434 }
435 }

```

(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:

436 \cs_new_protected:Npn \__tag_struct_insert_annot:nn #1 #2
437   % #1 object reference to the annotation/xform
438   % #2 structparent number
439   {
440     \bool_if:NT \g__tag_active_struct_bool
441     {
442       % get the number of the parent structure:
443       \seq_get:NNF
444         \g__tag_struct_stack_seq
445         \l__tag_struct_stack_parent_tmpa_tl
446         {
447           \msg_error:nn { tag } { struct-faulty-nesting }
448         }
449       % put the obj number of the annot in the kid entry, this also creates
450       % the OBJR object
451       \__tag_property_record:nn {@tag@objr@page@#2 }{ tagabspage }
452       \__tag_struct_kid_OBJR_gput_right:eee
453       {
454         \l__tag_struct_stack_parent_tmpa_tl
455       }
456       {
457         #1 %
458       }
459       {
460         \pdf_pageobject_ref:n
461         { \property_ref:nnn {@tag@objr@page@#2 }{ tagabspage }{1} }
462       }
463       % add the parent obj number to the parent tree:
464       \exp_args:Nne
465       \__tag_parenttree_add_objr:nn
466       {
467         #2
468       }
469       {
470         \pdf_object_ref_indexed:nn
471         { __tag/struct }{ \l__tag_struct_stack_parent_tmpa_tl }
472       }
473       % increase the int:
474       \int_gincr:N \c@g__tag_parenttree_obj_int
475     }
476   }
```

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

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

```
477 \cs_new:Npn \__tag_get_data_struct_tag:
478   {
479     \exp_args:Ne
480     \tl_tail:n
481     {
482       \prop_item:cn {g__tag_struct_}g__tag_struct_stack_current_tl _prop}{S}
483     }
484   }
```

(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`.

```
485 \cs_new:Npn \__tag_get_data_struct_id:
486   {
487     \__tag_struct_get_id:n {\g__tag_struct_stack_current_tl}
488   }
489 \</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

```
490 \<base>
491 \cs_new:Npn \__tag_get_data_struct_num:
492   {
493     \g__tag_struct_stack_current_tl
494   }
495 \</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.

```
496 \<base>
497 \cs_new:Npn \__tag_get_data_struct_counter:
498   {
499     \int_use:N \c@g__tag_struct_abs_int
500   }
501 \</base>
```

(End of definition for `__tag_get_data_struct_counter:.`)

5 Keys

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

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

```

502 (*package)
503 \socket_new:nn { tag/struct/tag }{1}
504 \socket_new_plug:nnn { tag/struct/tag }{ latex-tags }
505 {
506   \seq_set_split:Nne \l__tag_tmpa_seq { / }
507   {#1/\prop_item:Ne\g__tag_role_tags_NS_prop{#1}}
508   \tl_gset:Ne \g__tag_struct_tag_tl { \seq_item:Nn\l__tag_tmpa_seq {1} }
509   \tl_gset:Ne \g__tag_struct_tag_NS_tl{ \seq_item:Nn\l__tag_tmpa_seq {2} }
510   \__tag_check_structure_tag:N \g__tag_struct_tag_tl
511 }
512
513 \socket_new_plug:nnn { tag/struct/tag }{ pdf-tags }
514 {
515   \seq_set_split:Nne \l__tag_tmpa_seq { / }
516   {#1/\prop_item:Ne\g__tag_role_tags_NS_prop{#1}}
517   \tl_gset:Ne \g__tag_struct_tag_tl { \seq_item:Nn\l__tag_tmpa_seq {1} }
518   \tl_gset:Ne \g__tag_struct_tag_NS_tl{ \seq_item:Nn\l__tag_tmpa_seq {2} }
519   \__tag_role_get:VVNN
520   \g__tag_struct_tag_tl \g__tag_struct_tag_NS_tl \l__tag_tmpa_tl \l__tag_tmpb_tl
521   \tl_gset:Ne \g__tag_struct_tag_tl {\l__tag_tmpa_tl}
522   \tl_gset:Ne \g__tag_struct_tag_NS_tl{\l__tag_tmpb_tl}
523   \__tag_check_structure_tag:N \g__tag_struct_tag_tl
524 }
525 \socket_assign_plug:nn { tag/struct/tag } {latex-tags}

label (struct key)
stash (struct key) 526 \keys_define:nn { __tag / struct }
parent (struct key) 527 {
firstkid (struct key) 528   label .code:n      =
tag (struct key) 529   {
title (struct key) 530     \prop_gput:Nee\g__tag_struct_label_num_prop
title-o (struct key) 531     {#1}{\int_use:N \c@g__tag_struct_abs_int}
alt (struct key) 532     \__tag_property_record:eV
actualtext (struct key) 533     {tagpdfstruct-#1}
lang (struct key) 534     \c__tag_property_struct_clist
ref (struct key) 535   },
E (struct key) 536   stash .bool_set:N    = \l__tag_struct_elem_stash_bool,
537   parent .code:n      =
538   {
539     \bool_lazy_and:nnTF
540     {
541       \prop_if_exist_p:c { g__tag_struct_\int_eval:n {#1}_prop }
542     }
543     {
544       \int_compare_p:nNn {#1}<{\c@g__tag_struct_abs_int}
545     }
546     { \tl_set:Ne \l__tag_struct_stack_parent_tmpa_tl { \int_eval:n {#1} } }

```

```

547     {
548         \msg_warning:nnee { tag } { struct-unknown }
549         { \int_eval:n {#1} }
550         { parent~key~ignored }
551     }
552 },
553 parent .default:n = {-1},
554 firstkid .code:n = { \tl_set:Nn \l__tag_struct_addkid_tl {left} },
555 tag .code:n = % S property
556 {
557     \socket_use:nn { tag/struct/tag }{#1}
558 },
559 title .code:n = % T property
560 {
561     \str_set_convert:Nnnn
562     \l__tag_tmpa_str
563     { #1 }
564     { default }
565     { utf16/hex }
566     \__tag_struct_prop_gput:nne
567     { \int_use:N \c@g__tag_struct_abs_int }
568     { T }
569     { <\l__tag_tmpa_str> }
570 },
571 title-o .code:n = % T property
572 {
573     \str_set_convert:Nonn
574     \l__tag_tmpa_str
575     { #1 }
576     { default }
577     { utf16/hex }
578     \__tag_struct_prop_gput:nne
579     { \int_use:N \c@g__tag_struct_abs_int }
580     { T }
581     { <\l__tag_tmpa_str> }
582 },
583 alt .code:n = % Alt property
584 {
585     \tl_if_empty:oF{#1}
586     {
587         \str_set_convert:Noon
588         \l__tag_tmpa_str
589         { #1 }
590         { default }
591         { utf16/hex }
592         \__tag_struct_prop_gput:nne
593         { \int_use:N \c@g__tag_struct_abs_int }
594         { Alt }
595         { <\l__tag_tmpa_str> }
596     }
597 },
598 alttext .meta:n = {alt=#1},
599 actualtext .code:n = % ActualText property
600 {

```

```

601     \tl_if_empty:oF{#1}
602     {
603         \str_set_convert:Noon
604         \l__tag_tmpa_str
605         { #1 }
606         { default }
607         { utf16/hex }
608         \__tag_struct_prop_gput:nne
609         { \int_use:N \c@g__tag_struct_abs_int }
610         { ActualText }
611         { <\l__tag_tmpa_str>}
612     }
613 },
614 lang .code:n      = % Lang property
615 {
616     \__tag_struct_prop_gput:nne
617     { \int_use:N \c@g__tag_struct_abs_int }
618     { Lang }
619     { (#1) }
620 },
621 }

```

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:nN
\__tag_struct_Ref_label:nN
\__tag_struct_Ref_dest:nN
\__tag_struct_Ref_num:nN

```

These commands are helper commands that are stored as a list in the Ref key of a structure. They are executed when the structure elements are written in `__tag_struct_write_obj`. They are used in `__tag_struct_format_Ref`. They allow to add a Ref by object reference, label, destname and structure number

```

622 \cs_new_protected:Npn \__tag_struct_Ref_obj:nN #1 #2 %#1 a object reference
623 {
624     \tl_put_right:Ne#2
625     {
626         \c_space_tl#1
627     }
628 }
629
630 \cs_new_protected:Npn \__tag_struct_Ref_label:nN #1 #2 %#1 a label
631 {
632     \prop_get:NnNTF \g__tag_struct_label_num_prop {#1} \l__tag_tmpb_tl
633     {
634         \tl_put_right:Ne#2
635         {
636             \c_space_tl\tag_struct_object_ref:e{ \l__tag_tmpb_tl }
637         }
638     }
639     {
640         \msg_warning:nnn {tag}{struct-Ref-unknown}{Label~'#1'}
641     }
642 }
643 \cs_new_protected:Npn \__tag_struct_Ref_dest:nN #1 #2 %#1 a dest name
644 {
645     \prop_get:NnNTF \g__tag_struct_dest_num_prop {#1} \l__tag_tmpb_tl

```

```

646     {
647       \tl_put_right:Ne#2
648       {
649         \c_space_tl\tag_struct_object_ref:e{ \l__tag_tmpb_tl }
650       }
651     }
652     {
653       \msg_warning:nnn {tag}{struct-Ref-unknown}{Destination~'#1'}
654     }
655   }
656 \cs_new_protected:Npn \__tag_struct_Ref_num:nN #1 #2 %#1 a structure number
657 {
658   \tl_put_right:Ne#2
659   {
660     \c_space_tl\tag_struct_object_ref:e{ #1 }
661   }
662 }
663

```

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

```

ref (struct key)
E (struct key) 664 \keys_define:nn { __tag / struct }
665   {
666     ref .code:n      = % ref property
667     {
668       \clist_map_inline:on {#1}
669       {
670         \tag_struct_gput:nne
671         { \int_use:N \c@g__tag_struct_abs_int}{ref_label}{ ##1 }
672       }
673     },
674     E .code:n      = % E property
675     {
676       \str_set_convert:Nnon
677       \l__tag_tmpa_str
678       { #1 }
679       { default }
680       { utf16/hex }
681       \__tag_struct_prop_gput:nne
682       { \int_use:N \c@g__tag_struct_abs_int }
683       { E }
684       { <\l__tag_tmpa_str> }
685     },
686   }

```

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

```

687 \int_new:N\g__tag_struct_AFobj_int

```

(End of definition for \g__tag_struct_AFobj_int.)

```
688 \cs_generate_variant:Nn \pdffile_embed_stream:nnN {neN}
689 \cs_new_protected:Npn \__tag_struct_add_inline_AF:nn #1 #2
690 % #1 content, #2 extension
691 {
692   \tl_if_empty:nF{#1}
693   {
694     \group_begin:
695     \int_gincr:N \g__tag_struct_AFobj_int
696     \pdffile_embed_stream:neN
697     {#1}
698     {tag-Afile\int_use:N\g__tag_struct_AFobj_int.#2}
699     \l__tag_tmpa_tl
700     \__tag_struct_add_AF:ee
701     { \int_use:N \c@g__tag_struct_abs_int }
702     { \l__tag_tmpa_tl }
703     \__tag_struct_prop_gput:nne
704     { \int_use:N \c@g__tag_struct_abs_int }
705     { AF }
706     {
707       [
708         \tl_use:c
709         { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
710       ]
711     }
712     \group_end:
713   }
714 }
715
716 \cs_generate_variant:Nn \__tag_struct_add_inline_AF:nn {on}
717 \cs_new_protected:Npn \__tag_struct_add_AF:nn #1 #2
718 % #1 struct num #2 object reference
719 {
720   \tl_if_exist:cTF
721   {
722     g__tag_struct_#1_AF_tl
723   }
724   {
725     \tl_gput_right:ce
726     { g__tag_struct_#1_AF_tl }
727     { \c_space_tl #2 }
728   }
729   {
730     \tl_new:c
731     { g__tag_struct_#1_AF_tl }
732     \tl_gset:ce
733     { g__tag_struct_#1_AF_tl }
734     { #2 }
735   }
736 }
737 \cs_generate_variant:Nn \__tag_struct_add_AF:nn {en,ee}
738 \keys_define:nn { __tag / struct }
739 {
```

```

740 AF .code:n          = % AF property
741 {
742   \pdf_object_if_exist:eTF {#1}
743   {
744     \__tag_struct_add_AF:ee
745     { \int_use:N \c@g__tag_struct_abs_int }{\pdf_object_ref:e {#1}}
746     \__tag_struct_prop_gput:nne
747     { \int_use:N \c@g__tag_struct_abs_int }
748     { AF }
749     {
750       [
751         \tl_use:c
752         { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
753       ]
754     }
755   }
756   {
757     % message?
758   }
759 },
760 AFref .code:n       = % AF property
761 {
762   \tl_if_empty:eF {#1}
763   {
764     \__tag_struct_add_AF:ee { \int_use:N \c@g__tag_struct_abs_int }{#1}
765     \__tag_struct_prop_gput:nne
766     { \int_use:N \c@g__tag_struct_abs_int }
767     { AF }
768     {
769       [
770         \tl_use:c
771         { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
772       ]
773     }
774   }
775 },
776 ,AFinline .code:n =
777 {
778   \__tag_struct_add_inline_AF:nn {#1}{txt}
779 }
780 ,AFinline-o .code:n =
781 {
782   \__tag_struct_add_inline_AF:on {#1}{txt}
783 }
784 ,texsource .code:n =
785 {
786   \group_begin:
787   \pdfdict_put:nnn { l_pdffile/Filespec } {Desc}{(TeX~source)}
788   \pdfdict_put:nnn { l_pdffile/Filespec }{AFRelationship} { /Source }
789   \__tag_struct_add_inline_AF:on {#1}{tex}
790   \group_end:
791 }
792 ,mathml .code:n =
793 {

```

```

794     \group_begin:
795     \pdfdict_put:nnn { l_pdffile/Filespec } {Desc}{(mathml-representation)}
796     \pdfdict_put:nnn { l_pdffile/Filespec }{AFRelationship} { /Supplement }
797     \__tag_struct_add_inline_AF:on {#1}{xml}
798     \group_end:
799   }
800 }

```

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!

```

801 \keys_define:nn { __tag / setup }
802 {
803   root-AF .code:n =
804   {
805     \pdf_object_if_exist:nTF {#1}
806     {
807       \__tag_struct_add_AF:ee { 1 }{\pdf_object_ref:n {#1}}
808       \__tag_struct_prop_gput:nne
809       { 1 }
810       { AF }
811       {
812         [
813           \tl_use:c
814           { g__tag_struct_1_AF_tl }
815         ]
816       }
817     }
818   }
819 }
820 },
821 }
822 }

```

6 User commands

We allow to set a language by default

`\l__tag_struct_lang_tl`

```

823 \tl_new:N \l__tag_struct_lang_tl
824 \</package>

```

(End of definition for `\l__tag_struct_lang_tl`.)

`\tag_struct_begin:n`
`\tag_struct_end:`

```

825 <base>\cs_new_protected:Npn \tag_struct_begin:n #1 {\int_gincr:N \c@g__tag_struct_abs_int}
826 <base>\cs_new_protected:Npn \tag_struct_end: {}
827 <base>\cs_new_protected:Npn \tag_struct_end:n {}
828 <*package | debug>
829 <package>\cs_set_protected:Npn \tag_struct_begin:n #1 % #1 key-val
830 <debug>\cs_set_protected:Npn \tag_struct_begin:n #1 % #1 key-val
831 {
832 <package>\__tag_check_if_active_struct:T
833 <debug>\__tag_check_if_active_struct:TF

```

```

834     {
835         \group_begin:
836         \int_gincr:N \c@g__tag_struct_abs_int
837         \__tag_prop_new:c { g__tag_struct_\int_eval:n { \c@g__tag_struct_abs_int }_prop }
838 <debug>         \prop_new:c { g__tag_struct_debug_\int_eval:n { \c@g__tag_struct_abs_int }_prop }
839         \__tag_new_output_prop_handler:n { \int_eval:n { \c@g__tag_struct_abs_int } }
840         \__tag_seq_new:c { g__tag_struct_kids_\int_eval:n { \c@g__tag_struct_abs_int }_seq }
841 <debug>         \seq_new:c { g__tag_struct_debug_kids_\int_eval:n { \c@g__tag_struct_abs_int }_seq }
842         \pdf_object_new_indexed:nn { __tag/struct }
843         { \c@g__tag_struct_abs_int }
844         \__tag_struct_prop_gput:nnn
845         { \int_use:N \c@g__tag_struct_abs_int }
846         { Type }
847         { /StructElem }
848         \tl_if_empty:NF \l__tag_struct_lang_tl
849         {
850             \__tag_struct_prop_gput:nne
851             { \int_use:N \c@g__tag_struct_abs_int }
852             { Lang }
853             { (\l__tag_struct_lang_tl) }
854         }
855         \__tag_struct_prop_gput:nnn
856         { \int_use:N \c@g__tag_struct_abs_int }
857         { Type }
858         { /StructElem }
859
860         \tl_set:Nn \l__tag_struct_stack_parent_tmpa_tl {-1}
861         \keys_set:nn { __tag / struct } { #1 }
862
863         \__tag_struct_set_tag_info:eVV
864         { \int_use:N \c@g__tag_struct_abs_int }
865         \g__tag_struct_tag_tl
866         \g__tag_struct_tag_NS_tl
867         \__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.

```

867         \int_compare:nNnT { \l__tag_struct_stack_parent_tmpa_tl } = { -1 }
868         {
869             \seq_get:NNF
870             \g__tag_struct_stack_seq
871             \l__tag_struct_stack_parent_tmpa_tl
872             {
873                 \msg_error:nn { tag } { struct-faulty-nesting }
874             }
875         }
876         \seq_gpush:NV \g__tag_struct_stack_seq          \c@g__tag_struct_abs_int
877         \__tag_role_get:VVNN
878         \g__tag_struct_tag_tl
879         \g__tag_struct_tag_NS_tl
880         \l__tag_struct_roletag_tl
881         \l__tag_struct_roletag_NS_tl

```

to target role and role NS

```

882         \__tag_struct_prop_gput:nne

```



```

883         { \int_use:N \c@g__tag_struct_abs_int }
884     { rolemap }
885     {
886         {\l__tag_struct_roletag_tl}{\l__tag_struct_roletag_NS_tl}
887     }

```

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

```

888     \str_case:VnTF \l__tag_struct_roletag_tl
889     {
890         {Part} {}
891         {Div} {}
892         {NonStruct} {}
893     }
894     {
895         \prop_get:cnNT
896         { g__tag_struct_ \l__tag_struct_stack_parent_tmpa_tl _prop }
897         { parentrole }
898         \l__tag_get_tmpc_tl
899         {
900             \__tag_struct_prop_gput:nno
901             { \int_use:N \c@g__tag_struct_abs_int }
902             { parentrole }
903             {
904                 \l__tag_get_tmpc_tl
905             }
906         }
907     }
908     {
909         \__tag_struct_prop_gput:mne
910         { \int_use:N \c@g__tag_struct_abs_int }
911         { parentrole }
912         {
913             {\l__tag_struct_roletag_tl}{\l__tag_struct_roletag_NS_tl}
914         }
915     }
916     \seq_gpush:Ne \g__tag_struct_tag_stack_seq
917     {{\g__tag_struct_tag_tl}{\l__tag_struct_roletag_tl}}
918     \tl_gset:NV \g__tag_struct_stack_current_tl \c@g__tag_struct_abs_int
919     %\seq_show:N \g__tag_struct_stack_seq
920     \bool_if:NF
921     \l__tag_struct_elem_stash_bool
922     {

```

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

```

923         \__tag_struct_get_parentrole:eNN
924         {\l__tag_struct_stack_parent_tmpa_tl}
925         \l__tag_get_parent_tmpa_tl
926         \l__tag_get_parent_tmpb_tl
927         \__tag_check_parent_child:VVVVN
928         \l__tag_get_parent_tmpa_tl

```

```

929     \l__tag_get_parent_tmpb_tl
930     \g__tag_struct_tag_tl
931     \g__tag_struct_tag_NS_tl
932     \l__tag_parent_child_check_tl
933 \int_compare:nNnT {\l__tag_parent_child_check_tl}<0
934 {
935     \prop_get:cnN
936     { g__tag_struct_ \l__tag_struct_stack_parent_tmpa_tl _prop}
937     {S}
938     \l__tag_tmpa_tl
939     \quark_if_no_value:NT\l__tag_tmpa_tl{\tl_set:Nn \l__tag_tmpa_tl{UNKNOWN}}
940     \msg_warning:nneee
941     { tag }
942     {role-parent-child}
943     { \l__tag_get_parent_tmpa_tl/\l__tag_get_parent_tmpb_tl }
944     { \g__tag_struct_tag_tl/\g__tag_struct_tag_NS_tl }
945     { not~allowed~
946       (struct~\l__tag_struct_stack_parent_tmpa_tl,~\l__tag_tmpa_tl
947         \c_space_tl-->~struct~\int_eval:n {\c@g__tag_struct_abs_int})
948     }
949     \cs_set_eq:NN \l__tag_role_remap_tag_tl \g__tag_struct_tag_tl
950     \cs_set_eq:NN \l__tag_role_remap_NS_tl \g__tag_struct_tag_NS_tl
951     \__tag_role_remap:
952     \cs_gset_eq:NN \g__tag_struct_tag_tl \l__tag_role_remap_tag_tl
953     \cs_gset_eq:NN \g__tag_struct_tag_NS_tl \l__tag_role_remap_NS_tl
954     \__tag_struct_set_tag_info:eVV
955     { \int_use:N \c@g__tag_struct_abs_int }
956     \g__tag_struct_tag_tl
957     \g__tag_struct_tag_NS_tl
958 }

```

Set the Parent.

```

959 \__tag_struct_prop_gput:nne
960 { \int_use:N \c@g__tag_struct_abs_int }
961 { P }
962 {
963     \pdf_object_ref_indexed:nm { __tag/struct} { \l__tag_struct_stack_parent_tmpa_tl }
964 }
965 %record this structure as kid:
966 %\tl_show:N \g__tag_struct_stack_current_tl
967 %\tl_show:N \l__tag_struct_stack_parent_tmpa_tl
968 \use:c { __tag_struct_kid_struct_gput_ \l__tag_struct_addkid_tl :ee }
969 { \l__tag_struct_stack_parent_tmpa_tl }
970 { \g__tag_struct_stack_current_tl }
971 %\prop_show:c { g__tag_struct_ \g__tag_struct_stack_current_tl _prop }
972 %\seq_show:c { g__tag_struct_kids_ \l__tag_struct_stack_parent_tmpa_tl _seq }
973 }

```

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.

```

974 <debug>     \prop_gset_eq:cc
975 <debug>     { g__tag_struct_debug_ \int_eval:n {\c@g__tag_struct_abs_int}_prop }
976 <debug>     { g__tag_struct_ \int_eval:n {\c@g__tag_struct_abs_int}_prop }

```

```

977 <debug>          \prop_gput:cne
978 <debug>          { g__tag_struct_debug\_int_eval:n {\c@g__tag_struct_abs_int}_prop }
979 <debug>          { P }
980 <debug>          {
981 <debug>              \bool_if:NTF \l__tag_struct_elem_stash_bool
982 <debug>              {no-parent:~stashed}
983 <debug>              {
984 <debug>                  parent~structure:~\l__tag_struct_stack_parent_tmpa_tl\c_space_tl =~
985 <debug>                  \prop_item:cn{ g__tag_struct\_l__tag_struct_stack_parent_tmpa_tl _p
986 <debug>              }
987 <debug>          }
988 <debug>          \prop_gput:cne
989 <debug>          { g__tag_struct_debug\_int_eval:n {\c@g__tag_struct_abs_int}_prop }
990 <debug>          { NS }
991 <debug>          { \g__tag_struct_tag_NS_tl }

992          %\prop_show:c { g__tag_struct\_g__tag_struct_stack_current_tl _prop }
993          %\seq_show:c {g__tag_struct_kids\_l__tag_struct_stack_parent_tmpa_tl _seq}
994 <debug> \__tag_debug_struct_begin_insert:n { #1 }
995          \group_end:
996      }
997 <debug>{ \__tag_debug_struct_begin_ignore:n { #1 }}
998      }
999 <package>\cs_set_protected:Nn \tag_struct_end:
1000 <debug>\cs_set_protected:Nn \tag_struct_end:
1001   { %take the current structure num from the stack:
1002     %the objects are written later, lua mode hasn't all needed info yet
1003     %\seq_show:N \g__tag_struct_stack_seq
1004 <package>\__tag_check_if_active_struct:T
1005 <debug>\__tag_check_if_active_struct:TF
1006     {
1007         \seq_gpop:NN \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
1008         \seq_gpop:NNTF \g__tag_struct_stack_seq \l__tag_tmpa_tl
1009         {
1010             \__tag_check_info_closing_struct:o { \g__tag_struct_stack_current_tl }
1011         }
1012         { \__tag_check_no_open_struct: }
1013         % get the previous one, shouldn't be empty as the root should be there
1014         \seq_get:NNTF \g__tag_struct_stack_seq \l__tag_tmpa_tl
1015         {
1016             \tl_gset:NV \g__tag_struct_stack_current_tl \l__tag_tmpa_tl
1017         }
1018         {
1019             \__tag_check_no_open_struct:
1020         }
1021         \seq_get:NNT \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
1022         {
1023             \tl_gset:Ne \g__tag_struct_tag_tl
1024             { \exp_last_unbraced:NV\use_i:nn \l__tag_tmpa_tl }
1025             \prop_get:NVNT\g__tag_role_tags_NS_prop \g__tag_struct_tag_tl\l__tag_tmpa_tl
1026             {
1027                 \tl_gset:Ne \g__tag_struct_tag_NS_tl { \l__tag_tmpa_tl }
1028             }
1029         }
1030 <debug>\__tag_debug_struct_end_insert:

```

```

1031     }
1032 <debug>{\_tag_debug_struct_end_ignore:}
1033   }
1034
1035 \cs_set_protected:Npn \tag_struct_end:n #1
1036 {
1037 <debug>   \_tag_check_if_active_struct:T{\_tag_debug_struct_end_check:n{#1}}
1038   \tag_struct_end:
1039 }
1040 </package | debug>

```

(End of definition for \tag_struct_begin:n and \tag_struct_end:. These functions are documented on page 103.)

\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.

```

1041 <base>\cs_new_protected:Npn \tag_struct_use:n #1 {}
1042 <*package | debug>
1043 \cs_set_protected:Npn \tag_struct_use:n #1 %#1 is the label
1044 {
1045   \_tag_check_if_active_struct:T
1046   {
1047     \prop_if_exist:cTF
1048     { g__tag_struct\_property_ref:enn{tagpdfstruct-#1}{tagstruct}{unknown}_prop } %
1049     {
1050       \_tag_check_struct_used:n {#1}
1051       %add the label structure as kid to the current structure (can be the root)
1052       \_tag_struct_kid_struct_gput_right:ee
1053       { \g__tag_struct_stack_current_tl }
1054       { \property_ref:enn{tagpdfstruct-#1}{tagstruct}{1} }
1055       %add the current structure to the labeled one as parents
1056       \_tag_prop_gput:cne
1057       { g__tag_struct\_property_ref:enn{tagpdfstruct-#1}{tagstruct}{1}_prop }
1058       { P }
1059       {
1060         \pdf_object_ref_indexed:nn { \_tag/struct } { \g__tag_struct_stack_current_tl }
1061       }

```

debug code

```

1062 <debug>   \prop_gput:cne
1063 <debug>   { g__tag_struct_debug\_property_ref:enn{tagpdfstruct-#1}{tagstruct}{1}_pr
1064 <debug>   { P }
1065 <debug>   {
1066 <debug>     parent~structure:~\g__tag_struct_stack_current_tl\c_space_tl=~
1067 <debug>     \g__tag_struct_tag_tl
1068 <debug>   }

```

check if the tag is allowed as child. Here we have to retrieve the tag info for the child, while the data for the parent is in the global tl-vars:

```

1069   \_tag_struct_get_parentrole:eNN
1070   {\property_ref:enn{tagpdfstruct-#1}{tagstruct}{1}}
1071   \l__tag_tmpa_tl
1072   \l__tag_tmpb_tl
1073   \_tag_check_parent_child:VVVVN
1074   \g__tag_struct_tag_tl

```

```

1075     \g__tag_struct_tag_NS_tl
1076     \l__tag_tmpa_tl
1077     \l__tag_tmpb_tl
1078     \l__tag_parent_child_check_tl
1079 \int_compare:nNnT {\l__tag_parent_child_check_tl}<0
1080 {
1081     \cs_set_eq:NN \l__tag_role_remap_tag_tl \g__tag_struct_tag_tl
1082     \cs_set_eq:NN \l__tag_role_remap_NS_tl \g__tag_struct_tag_NS_tl
1083     \__tag_role_remap:
1084     \cs_gset_eq:NN \g__tag_struct_tag_tl \l__tag_role_remap_tag_tl
1085     \cs_gset_eq:NN \g__tag_struct_tag_NS_tl \l__tag_role_remap_NS_tl
1086     \__tag_struct_set_tag_info:eVV
1087     { \int_use:N \c@g__tag_struct_abs_int }
1088     \g__tag_struct_tag_tl
1089     \g__tag_struct_tag_NS_tl
1090 }
1091 }
1092 {
1093     \msg_warning:nnn{ tag }{struct-label-unknown}{#1}
1094 }
1095 }
1096 }
1097 \endpackage|debug

```

(End of definition for `\tag_struct_use:n`. This function is documented on page 103.)

`\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.

```

1098 \base\cs_new_protected:Npn \tag_struct_use_num:n #1 {}
1099 \*package|debug
1100 \cs_set_protected:Npn \tag_struct_use_num:n #1 %1 is structure number
1101 {
1102     \__tag_check_if_active_struct:T
1103     {
1104         \prop_if_exist:cTF
1105         { g__tag_struct_#1_prop } %
1106         {
1107             \prop_get:cnNT
1108             {g__tag_struct_#1_prop}
1109             {P}
1110             \l__tag_tmpa_tl
1111             {
1112                 \msg_warning:nnn { tag } {struct-used-twice} {#1}
1113             }
1114             %add the \#1 structure as kid to the current structure (can be the root)
1115             \__tag_struct_kid_struct_gput_right:ee
1116             { \g__tag_struct_stack_current_tl }
1117             { #1 }
1118             %add the current structure to \#1 as parent
1119             \__tag_struct_prop_gput:nne
1120             { #1 }
1121             { P }
1122             {

```

```

1123         \pdf_object_ref_indexed:nn { __tag/struct }{ \g__tag_struct_stack_current_tl
1124     }
1125     <debug>         \prop_gput:cne
1126     <debug>         { g__tag_struct_debug_#1_prop }
1127     <debug>         { P }
1128     <debug>         {
1129     <debug>             parent~structure:~\g__tag_struct_stack_current_tl\c_space_tl=~
1130     <debug>             \g__tag_struct_tag_tl
1131     <debug>         }

```

check if the tag is allowed as child. Here we have to retrieve the tag info for the child, while the data for the parent is in the global tl-vars:

```

1132         \__tag_struct_get_parentrole:eNN
1133         {#1}
1134         \l__tag_tmpa_tl
1135         \l__tag_tmpb_tl
1136     \__tag_check_parent_child:VVVVN
1137     \g__tag_struct_tag_tl
1138     \g__tag_struct_tag_NS_tl
1139     \l__tag_tmpa_tl
1140     \l__tag_tmpb_tl
1141     \l__tag_parent_child_check_tl
1142     \int_compare:nNnT {\l__tag_parent_child_check_tl}<0
1143     {
1144         \cs_set_eq:NN \l__tag_role_remap_tag_tl \g__tag_struct_tag_tl
1145         \cs_set_eq:NN \l__tag_role_remap_NS_tl \g__tag_struct_tag_NS_tl
1146         \__tag_role_remap:
1147         \cs_gset_eq:NN \g__tag_struct_tag_tl \l__tag_role_remap_tag_tl
1148         \cs_gset_eq:NN \g__tag_struct_tag_NS_tl \l__tag_role_remap_NS_tl
1149         \__tag_struct_set_tag_info:eVV
1150         { \int_use:N \c@g__tag_struct_abs_int }
1151         \g__tag_struct_tag_tl
1152         \g__tag_struct_tag_NS_tl
1153     }
1154     }
1155     {
1156     \msg_warning:nnn{ tag }{struct-label-unknown}{#1}
1157     }
1158     }
1159 }
1160 </package | debug>

```

(End of definition for \tag_struct_use_num:n. This function is documented on page 103.)

\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.

```

1161 <*package>
1162 \cs_new:Npn \tag_struct_object_ref:n #1
1163 {
1164     \pdf_object_ref_indexed:nn {__tag/struct}{ #1 }
1165 }
1166 \cs_generate_variant:Nn \tag_struct_object_ref:n {e}
1167 </package>

```

(End of definition for `\tag_struct_object_ref:n`. This function is documented on page 103.)

`\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 all 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. At last the keyword `ref_num` expects a structure number.

```
1168 <base>\cs_new_protected:Npn \tag_struct_gput:nnn #1 #2 #3{}
1169 <*package>
1170 \cs_set_protected:Npn \tag_struct_gput:nnn #1 #2 #3
1171 {
1172   \cs_if_exist_use:cF {__tag_struct_gput_data_#2:nn}
1173   { %warning??
1174     \use_none:nn
1175   }
1176   {#1}{#3}
1177 }
1178 \cs_generate_variant:Nn \tag_struct_gput:nnn {ene,nne}
1179 </package>
```

(End of definition for `\tag_struct_gput:nnn`. This function is documented on page 104.)

`__tag_struct_gput_data_ref_aux:nnn`

```
1180 <*package>
1181 \cs_new_protected:Npn \__tag_struct_gput_data_ref_aux:nnn #1 #2 #3
1182   % #1 receiving struct num, #2 key word #3 value
1183   {
1184     \prop_get:cnNTF
1185     { g__tag_struct_#1_prop }
1186     { Ref }
1187     \l__tag_get_tmpc_tl
1188     {
1189       \tl_put_right:No \l__tag_get_tmpc_tl
1190       {\cs:w __tag_struct_Ref_#2:nN \cs_end: {#3},}
1191     }
1192     {
1193       \tl_set:No \l__tag_get_tmpc_tl
1194       {\cs:w __tag_struct_Ref_#2:nN \cs_end: {#3},}
1195     }
1196     \__tag_struct_prop_gput:nno
1197     { #1 }
1198     { Ref }
1199     { \l__tag_get_tmpc_tl }
1200   }
1201 \cs_new_protected:Npn \__tag_struct_gput_data_ref:nn #1 #2
1202   {
1203     \__tag_struct_gput_data_ref_aux:nnn {#1}{obj}{#2}
1204   }
```

```

1205 \cs_new_protected:Npn \__tag_struct_gput_data_ref_label:nn #1 #2
1206 {
1207   \__tag_struct_gput_data_ref_aux:nnn {#1}{label}{#2}
1208 }
1209 \cs_new_protected:Npn \__tag_struct_gput_data_ref_dest:nn #1 #2
1210 {
1211   \__tag_struct_gput_data_ref_aux:nnn {#1}{dest}{#2}
1212 }
1213 \cs_new_protected:Npn \__tag_struct_gput_data_ref_num:nn #1 #2
1214 {
1215   \__tag_struct_gput_data_ref_aux:nnn {#1}{num}{#2}
1216 }
1217
1218 \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_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:ee` increases the counter given back by `\tag_struct_parent_int:.`

`\tag_struct_insert_annot:ee` It must be used together with `\tag_struct_parent_int:` to insert an annotation. `\tag_struct_parent_int:` `\tag_struct_parent_int:.` TODO: decide how it should be guarded if tagging is deactivated.

`\tag_struct_parent_int:`

```

1219 \cs_new_protected:Npn \tag_struct_insert_annot:nn #1 #2 % #1 should be an object reference
1220                                     % #2 struct parent num
1221 {
1222   \__tag_check_if_active_struct:T
1223   {
1224     \tag_struct_insert_annot:nn {#1}{#2}
1225   }
1226 }
1227
1228 \cs_generate_variant:Nn \tag_struct_insert_annot:nn {xx,ee}
1229 \cs_new:Npn \tag_struct_parent_int: {\int_use:c { c@g__tag_parenttree_obj_int }}
1230
1231 \</package>
1232

```

(End of definition for `\tag_struct_insert_annot:nn` and `\tag_struct_parent_int:.` These functions are documented on page 103.)

7 Attributes and attribute classes

```

1233 \<header>
1234 \ProvidesExplPackage {tagpdf-attr-code} {2025-01-12} {0.991}
1235 {part of tagpdf - code related to attributes and attribute classes}
1236 \</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.
`\g__tag_attr_objref_prop` `\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`
`\l__tag_attr_value_tl`


```

1237 (*package)
1238 \prop_new:N \g__tag_attr_entries_prop
1239 \prop_new_linked:N \g__tag_attr_class_used_prop
1240 \tl_new:N \l__tag_attr_value_tl
1241 \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.

```

1242 \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},
}

1243 \cs_new_protected:Npn \__tag_attr_new_entry:nn #1 #2 %#1:name, #2: content
1244 {
1245   \prop_gput:Nen \g__tag_attr_entries_prop
1246   {\pdf_name_from_unicode_e:n{#1}}{#2}
1247 }
1248
1249 \cs_generate_variant:Nn \__tag_attr_new_entry:nn {ee}
1250 \keys_define:nn { __tag / setup }
1251 {
1252   role/new-attribute .code:n =
1253   {
1254     \__tag_attr_new_entry:nn #1
1255   }

```

deprecated name

```

1256   ,newattribute .code:n =
1257   {
1258     \__tag_attr_new_entry:nn #1
1259   },
1260 }

```

(End of definition for __tag_attr_new_entry:nn, role/new-attribute (setup-key), and newattribute (deprecated). These functions are documented on page 106.)

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

```

1261 \keys_define:nn { __tag / struct }
1262 {
1263   attribute-class .code:n =

```

```

1264     {
1265     \clist_set:Ne \l__tag_tmpa_clist { #1 }
1266     \seq_set_from_clist:NN \l__tag_tmpb_seq \l__tag_tmpa_clist
we convert the names into pdf names with slash
1267     \seq_set_map_e:NNn \l__tag_tmpa_seq \l__tag_tmpb_seq
1268     {
1269     \pdf_name_from_unicode_e:n {##1}
1270     }
1271     \seq_map_inline:Nn \l__tag_tmpa_seq
1272     {
1273     \prop_if_in:NnF \g__tag_attr_entries_prop {##1}
1274     {
1275     \msg_error:nnn { tag } { attr-unknown } { ##1 }
1276     }
1277     \prop_gput:Nnn\g__tag_attr_class_used_prop { ##1} {}
1278     }
1279     \tl_set:Ne \l__tag_tmpa_tl
1280     {
1281     \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}
1282     \seq_use:Nn \l__tag_tmpa_seq { \c_space_tl }
1283     \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}
1284     }
1285     \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 0 }
1286     {
1287     \__tag_struct_prop_gput:nne
1288     { \int_use:N \c@g__tag_struct_abs_int }
1289     { C }
1290     { \l__tag_tmpa_tl }
1291     %\prop_show:c { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
1292     }
1293     }
1294     }

```

attribute (*struct key*)

```

1295 \keys_define:nn { __tag / struct }
1296 {
1297     attribute .code:n = % A property (attribute, value currently a dictionary)
1298     {
1299     \clist_set:Ne          \l__tag_tmpa_clist { #1 }
1300     \clist_if_empty:NF \l__tag_tmpa_clist
1301     {
1302     \seq_set_from_clist:NN \l__tag_tmpb_seq \l__tag_tmpa_clist
we convert the names into pdf names with slash
1303     \seq_set_map_e:NNn \l__tag_tmpa_seq \l__tag_tmpb_seq
1304     {
1305     \pdf_name_from_unicode_e:n {##1}
1306     }
1307     \tl_set:Ne \l__tag_attr_value_tl
1308     {
1309     \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}
1310     }
1311     \seq_map_inline:Nn \l__tag_tmpa_seq
1312     {

```

```

1313         \prop_if_in:NnF \g__tag_attr_entries_prop {##1}
1314         {
1315           \msg_error:nnn { tag } { attr-unknown } { ##1 }
1316         }
1317         \prop_if_in:NnF \g__tag_attr_objref_prop {##1}
1318         {%\prop_show:N \g__tag_attr_entries_prop
1319           \pdf_object_unnamed_write:ne
1320           { dict }
1321           {
1322             \prop_item:Nn\g__tag_attr_entries_prop {##1}
1323           }
1324           \prop_gput:Nne \g__tag_attr_objref_prop {##1} {\pdf_object_ref_last:}
1325         }
1326         \tl_put_right:Ne \l__tag_attr_value_tl
1327         {
1328           \c_space_tl
1329           \prop_item:Nn \g__tag_attr_objref_prop {##1}
1330         }
1331     % \tl_show:N \l__tag_attr_value_tl
1332     }
1333     \tl_put_right:Ne \l__tag_attr_value_tl
1334     { %[
1335       \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}%
1336     }
1337     % \tl_show:N \l__tag_attr_value_tl
1338     \__tag_struct_prop_gput:nne
1339     { \int_use:N \c@g__tag_struct_abs_int }
1340     { A }
1341     { \l__tag_attr_value_tl }
1342   }
1343 },
1344 }
1345 \end{package}

```

Part IX

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

```
1 <@@=tag>
2 <*luatex>
3 \ProvidesExplFile {tagpdf-luatex.def} {2025-01-12} {0.991}
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
  \__tag_seq_gput_right:Nn
  \__tag_seq_gput_left:Nn
  \__tag_seq_item:cn
  \__tag_prop_item:cn
  \__tag_seq_show:N
  \__tag_prop_show:N
9 \cs_set_protected:Npn \__tag_prop_new:N #1
10 {
11   \prop_new:N #1
12   \lua_now:e { ltx.__tag.tables.\cs_to_str:N#1 = {} }
13 }
14
15 \cs_set_protected:Npn \__tag_prop_new_linked:N #1
16 {
17   \prop_new_linked:N #1
18   \lua_now:e { ltx.__tag.tables.\cs_to_str:N#1 = {} }
19 }
20
21
22 \cs_set_protected:Npn \__tag_seq_new:N #1
23 {
24   \seq_new:N #1
25   \lua_now:e { ltx.__tag.tables.\cs_to_str:N#1 = {} }
26 }
27
28
29 \cs_set_protected:Npn \__tag_prop_gput:Nnn #1 #2 #3
```

```

30 {
31   \prop_gput:Nnn #1 { #2 } { #3 }
32   \lua_now:e { ltx.__tag.tables.\cs_to_str:N#1 ["#2"] = "\lua_escape:n{#3}" }
33 }
34
35
36 \cs_set_protected:Npn \__tag_seq_gput_right:Nn #1 #2
37 {
38   \seq_gput_right:Nn #1 { #2 }
39   \lua_now:e { table.insert(ltx.__tag.tables.\cs_to_str:N#1, "#2") }
40 }

```

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

```

41 \cs_set_protected:Npn \__tag_seq_gput_left:Nn #1 #2
42 {
43   \seq_gput_left:Nn #1 { #2 }
44   \lua_now:e { table.insert(ltx.__tag.tables.\cs_to_str:N#1, "#2") }
45 }
46
47 %Hm not quite sure about the naming
48 \cs_set:Npn \__tag_seq_item:cn #1 #2
49 {
50   \lua_now:e { tex.print(ltx.__tag.tables.#1[#2]) }
51 }
52
53 \cs_set:Npn \__tag_prop_item:cn #1 #2
54 {
55   \lua_now:e { tex.print(ltx.__tag.tables.#1["#2"]) }
56 }
57
58 %for debugging commands that show both the seq/prop and the lua tables
59 \cs_set_protected:Npn \__tag_seq_show:N #1
60 {
61   \seq_show:N #1
62   \lua_now:e { ltx.__tag.trace.log ("lua~sequence~array~\cs_to_str:N#1",1) }
63   \lua_now:e { ltx.__tag.trace.show_seq (ltx.__tag.tables.\cs_to_str:N#1) }
64 }
65
66 \cs_set_protected:Npn \__tag_prop_show:N #1
67 {
68   \prop_show:N #1
69   \lua_now:e {ltx.__tag.trace.log ("lua~property~table~\cs_to_str:N#1",1) }
70   \lua_now:e {ltx.__tag.trace.show_prop (ltx.__tag.tables.\cs_to_str:N#1) }
71 }

```

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

```

72 </luatex>

```

The module declaration

```

73 <*lua>
74 -- tagpdf.lua
75 -- Ulrike Fischer
76
77 local ProvidesLuaModule = {

```

```

78     name          = "tagpdf",
79     version       = "0.991",          --TAGVERSION
80     date          = "2025-01-12",    --TAGDATE
81     description   = "tagpdf lua code",
82     license       = "The LATEX Project Public License 1.3c"
83 }
84
85 if luatexbase and luatexbase.provides_module then
86   luatexbase.provides_module (ProvidesLuaModule)
87 end
88
89 --[[
90 The code has quite probably a number of problems
91 - more variables should be local instead of global
92 - the naming is not always consistent due to the development of the code
93 - the traversing of the shipout box must be tested with more complicated setups
94 - it should probably handle more node types
95 -
96 --]]
97

```

Some comments about the lua structure.

```

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

```

```

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.showspaces : 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 `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 mccntattributeid  = luatexbase.new_attribute ("g__tag_mc_cnt_attr")
143 local iwspaceOffattributeid = luatexbase.new_attribute ("g__tag_interwordspaceOff_attr")
144 local iwspaceattributeid  = luatexbase.new_attribute ("g__tag_interwordspace_attr")
145 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

```

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

```

148 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.

```

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

```

```

168 local fontid          = font.id
169
170 local HLIST           = node.id("hlist")
171 local VLIST           = node.id("vlist")
172 local RULE            = node.id("rule")
173 local DISC            = node.id("disc")
174 local GLUE            = node.id("glue")
175 local GLYPH           = node.id("glyph")
176 local KERN            = node.id("kern")
177 local PENALTY         = node.id("penalty")
178 local LOCAL_PAR       = node.id("local_par")
179 local MATH             = node.id("math")
180
181 local explicit_disc = 1
182 local regular_disc = 3

```

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

```

183 ltx          = ltx          or { }
184 ltx.tag      = ltx.tag      or { } -- user commands
185 ltx.__tag    = ltx.__tag    or { }
186 ltx.__tag.mc = ltx.__tag.mc or { } -- mc data
187 ltx.__tag.struct = ltx.__tag.struct or { } -- struct data
188 ltx.__tag.tables = ltx.__tag.tables or { } -- tables created with new prop and new seq.
189                                     -- wasn't a so great idea ...
190                                     -- g__tag_role_tags_seq used by tag<-> is in this table
191                                     -- used for pure lua tables too now!
192 ltx.__tag.page   = ltx.__tag.page   or { } -- page data, currently only i->{0->mcnum,1->mcnum}
193 ltx.__tag.trace  = ltx.__tag.trace  or { } -- show commands
194 ltx.__tag.func   = ltx.__tag.func   or { } -- functions
195 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.

```

196 local __tag_get_struct_num =
197   function()
198     local a = token.get_macro("g__tag_struct_stack_current_tl")
199     return a
200   end
201
202 local __tag_get_struct_counter =
203   function()
204     local a = tex.getcount("c@g__tag_struct_abs_int")
205     return a
206   end
207
208 local __tag_get_struct_num_next =
209   function()
210     local a = tex.getcount("c@g__tag_struct_abs_int") + 1
211     return a
212   end

```



```

213
214 ltx.tag.get_struct_num = __tag_get_struct_num
215 ltx.tag.get_struct_counter = __tag_get_struct_counter
216 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`.

```

217 local __tag_log =
218   function (message,loglevel)
219     if (loglevel or 3) <= tex.count["l__tag_loglevel_int"] then
220       texio.write_nl("tagpdf: ".. message)
221     end
222   end
223
224 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.

```

225 function ltx.__tag.trace.show_seq (seq)
226   if (type(seq) == "table") then
227     for i,v in ipairs(seq) do
228       __tag_log ("[" .. i .. "]" => " .. tostring(v),1)
229     end
230   else
231     __tag_log ("sequence " .. tostring(seq) .. " not found",1)
232   end
233 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`

```

234 local __tag_pairs_prop =
235   function (prop)
236     local a = {}
237     for n in pairs(prop) do tableinsert(a, n) end
238     table.sort(a)
239     local i = 0 -- iterator variable
240     local iter = function () -- iterator function
241       i = i + 1
242       if a[i] == nil then return nil
243       else return a[i], prop[a[i]]
244     end
245     end
246     return iter

```

```

247 end
248
249
250 function ltx.__tag.trace.show_prop (prop)
251 if (type(prop) == "table") then
252   for i,v in __tag_pairs_prop (prop) do
253     __tag_log ("[" .. i .. "] => " .. tostring(v),1)
254   end
255 else
256   __tag_log ("prop " .. tostring(prop) .. " not found or not a table",1)
257 end
258 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

```

259 function ltx.__tag.trace.show_mc_data (num,loglevel)
260 if ltx.__tag and ltx.__tag.mc and ltx.__tag.mc[num] then
261   for k,v in pairs(ltx.__tag.mc[num]) do
262     __tag_log ("mc"..num..": " ..tostring(k)..=>" ..tostring(v),loglevel)
263   end
264   if ltx.__tag.mc[num]["kids"] then
265     __tag_log ("mc" .. num .. " has " .. #ltx.__tag.mc[num]["kids"] .. " kids",loglevel)
266     for k,v in ipairs(ltx.__tag.mc[num]["kids"]) do
267       __tag_log ("mc" .. num .. " kid "..k.." =>" .. v.kid.." on page " ..v.page,loglevel)
268     end
269   end
270 else
271   __tag_log ("mc"..num.." not found",loglevel)
272 end
273 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.

```

274 function ltx.__tag.trace.show_all_mc_data (min,max,loglevel)
275   for i = min, max do
276     ltx.__tag.trace.show_mc_data (i,loglevel)
277   end
278   texio.write_nl("")
279 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.

```

280 function ltx.__tag.trace.show_struct_data (num)
281 if ltx.__tag and ltx.__tag.struct and ltx.__tag.struct[num] then
282   for k,v in ipairs(ltx.__tag.struct[num]) do
283     __tag_log ("struct "..num..": " ..tostring(k)..=>" ..tostring(v),1)
284   end
285 else
286   __tag_log ("struct "..num.." not found ",1)
287 end
288 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).

```
289 local __tag_get_mc_cnt_type_tag = function (n)
290   local mcnt      = nodegetattribute(n,mcntattributeid) or -1
291   local mctype    = nodegetattribute(n,mctypeattributeid) or -1
292   local tag       = ltx.__tag.func.get_tag_from(mctype)
293   return mcnt,mctype,tag
294 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.

```
295 local function __tag_get_mathsubtype (mathnode)
296   if mathnode.subtype == 0 then
297     subtype = "beginmath"
298   else
299     subtype = "endmath"
300   end
301   return subtype
302 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.

```
303 ltx.__tag.tables.role_tag_attribute = {}
304 ltx.__tag.tables.role_attribute_tag = {}
```

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

`ltx.__tag.func.alloctag`

```
305 local __tag_alloctag =
306   function (tag)
307     if not ltx.__tag.tables.role_tag_attribute[tag] then
308       table.insert(ltx.__tag.tables.role_attribute_tag,tag)
309       ltx.__tag.tables.role_tag_attribute[tag]=#ltx.__tag.tables.role_attribute_tag
310       __tag_log ("Add "..tag.." "..ltx.__tag.tables.role_tag_attribute[tag],3)
311     end
312   end
313 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.

```

314 local __tag_get_num_from =
315 function (tag)
316   if ltx.__tag.tables.role_tag_attribute[tag] then
317     a= ltx.__tag.tables.role_tag_attribute[tag]
318   else
319     a= -1
320   end
321   return a
322 end
323
324 ltx.__tag.func.get_num_from = __tag_get_num_from
325
326 function ltx.__tag.func.output_num_from (tag)
327   local num = __tag_get_num_from (tag)
328   tex.sprint(catlatex,num)
329   if num == -1 then
330     __tag_log ("Unknown tag "..tag.." used")
331   end
332 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.

```

333 local __tag_get_tag_from =
334 function (num)
335   if ltx.__tag.tables.role_attribute_tag[num] then
336     a = ltx.__tag.tables.role_attribute_tag[num]
337   else
338     a= "UNKNOWN"
339   end
340   return a
341 end
342
343 ltx.__tag.func.get_tag_from = __tag_get_tag_from
344
345 function ltx.__tag.func.output_tag_from (num)
346   tex.sprint(catlatex,__tag_get_tag_from (num))
347 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.

```

348 function ltx.__tag.func.store_mc_data (num,key,data)
349   ltx.__tag.mc[num] = ltx.__tag.mc[num] or { }
350   ltx.__tag.mc[num][key] = data
351   __tag_log ("INFO TEX-STORE-MC-DATA: "..num.." => "..tostring(key).. " => "..tostring(data),3)
352 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.

```
353 function ltx.__tag.func.store_mc_label (label,num)
354   ltx.__tag.mc["labels"] = ltx.__tag.mc["labels"] or { }
355   ltx.__tag.mc.labels[label] = num
356 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.

```
357 function ltx.__tag.func.store_mc_kid (mcnum,kid,page)
358   ltx.__tag.trace.log("INFO TAG-STORE-MC-KID: "..mcnum.." => " .. kid.." on page " .. page,3)
359   ltx.__tag.mc[mcnum]["kids"] = ltx.__tag.mc[mcnum]["kids"] or { }
360   local kidtable = {kid=kid,page=page}
361   tableinsert(ltx.__tag.mc[mcnum]["kids"], kidtable )
362 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.

```
363 function ltx.__tag.func.mc_num_of_kids (mcnum)
364   local num = 0
365   if ltx.__tag.mc[mcnum] and ltx.__tag.mc[mcnum]["kids"] then
366     num = #ltx.__tag.mc[mcnum]["kids"]
367   end
368   ltx.__tag.trace.log ("INFO MC-KID-NUMBERS: " .. mcnum .. "has " .. num .. "KIDS",4)
369   return num
370 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 dvi_{pdf}mx backend
__tag_insert_emc_node

```
371 local __tag_backend_create_emc_node
372 if tex.outputmode == 0 then
373   if token.get_macro("c_sys_backend_str") == "dvipdfmx" then
374     function __tag_backend_create_emc_node ()
375       local emcnode = nodenew("whatsit","special")
376       emcnode.data = "pdf:code EMC"
377       return emcnode
378     end
379   else -- assume a dvips variant
380     function __tag_backend_create_emc_node ()
381       local emcnode = nodenew("whatsit","special")
382       emcnode.data = "ps:SDict begin mark /EMC pdfmark end"
383       return emcnode
384     end
385   end
386 else -- pdf mode
```

```

387 function __tag_backend_create_emc_node ()
388     local emcnode = nodenew("whatsit", "pdf_literal")
389     emcnode.data = "EMC"
390     emcnode.mode=1
391     return emcnode
392 end
393 end
394
395 local function __tag_insert_emc_node (head,current)
396     local emcnode= __tag_backend_create_emc_node()
397     head = node.insert_before(head,current,emcnode)
398     return head
399 end

```

(End of definition for __tag_backend_create_emc_node and __tag_insert_emc_node.)

__tag_backend_create_bmc_node
__tag_insert_bmc_node

This inserts a simple bmc node

```

400 local __tag_backend_create_bmc_node
401 if tex.outputmode == 0 then
402     if token.get_macro("c_sys_backend_str") == "dvipdfmx" then
403         function __tag_backend_create_bmc_node (tag)
404             local bmcnode = nodenew("whatsit", "special")
405             bmcnode.data = "pdf:code /"..tag.." BMC"
406             return bmcnode
407         end
408     else -- assume a dvips variant
409         function __tag_backend_create_bmc_node (tag)
410             local bmcnode = nodenew("whatsit", "special")
411             bmcnode.data = "ps:SDict begin mark/"..tag.." /BMC pdfmark end"
412             return bmcnode
413         end
414     end
415 else -- pdf mode
416     function __tag_backend_create_bmc_node (tag)
417         local bmcnode = nodenew("whatsit", "pdf_literal")
418         bmcnode.data = "/"..tag.." BMC"
419         bmcnode.mode=1
420         return bmcnode
421     end
422 end
423
424 local function __tag_insert_bmc_node (head,current,tag)
425     local bmcnode = __tag_backend_create_bmc_node (tag)
426     head = node.insert_before(head,current,bmcnode)
427     return head
428 end

```

(End of definition for __tag_backend_create_bmc_node and __tag_insert_bmc_node.)

__tag_backend_create_bdc_node
__tag_insert_bdc_node

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

```

429 local __tag_backend_create_bdc_node
430
431 if tex.outputmode == 0 then

```

```

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

```

459 local function __tag_pdf_object_ref (name,index)
460   local object
461   if ltx.pdf.object_id then
462     object = ltx.pdf.object_id (name,index) ..' 0 R'
463   else
464     local tokename = 'c__pdf_object_'..name..'/'..index..'_int'
465     object = token.create(tokename).mode ..' 0 R'
466   end
467   return object
468 end
469 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.

```

470 local function __tag_show_spacemark (head,current,color,height)
471   local markcolor = color or "1 0 0"
472   local markheight = height or 10

```

```

473 local pdfstring
474 if tex.outputmode == 0 then
475   -- ignore dvi mode for now
476 else
477   pdfstring = node.new("whatsit","pdf_literal")
478   pdfstring.data =
479     string.format("q ".markcolor.." RG ".markcolor.." rg 0.4 w 0 %g m 0 %g l S Q",-
3,markheight)
480   head = node.insert_after(head,current,pdfstring)
481   return head
482 end
483 end

```

(End of definition for `__tag_show_spacemark`.)

```

__tag_fakespace This is used to define a lua version of \pdffakespace
ltx.__tag.func.fakespace
484 local function __tag_fakespace()
485   tex.setattribute(iwspaceattributeid,1)
486   tex.setattribute(iwfontattributeid,font.current())
487 end
488 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.

```

489 --[[ a function to mark up places where real space chars should be inserted
490       it only sets an attribute.
491 --]]
492
493 local function __tag_mark_spaces (head)
494   local inside_math = false
495   for n in nodetraverse(head) do
496     local id = n.id
497     if id == GLYPH then
498       local glyph = n
499       default_currfontid = glyph.font
500       if glyph.next and (glyph.next.id == GLUE)
501         and not inside_math and (glyph.next.width >0)
502       then
503         nodesetattribute(glyph.next,iwspaceattributeid,1)
504         nodesetattribute(glyph.next,iwfontattributeid,glyph.font)
505       -- for debugging
506       if ltx.__tag.trace.showspaces then
507         __tag_show_spacemark (head,glyph)
508       end
509     elseif glyph.next and (glyph.next.id==KERN) and not inside_math then
510       local kern = glyph.next
511       if kern.next and (kern.next.id== GLUE) and (kern.next.width >0)
512     then
513       nodesetattribute(kern.next,iwspaceattributeid,1)
514       nodesetattribute(kern.next,iwfontattributeid,glyph.font)
515     end

```



```

516     end
517     -- look also back
518     if glyph.prev and (glyph.prev.id == GLUE)
519         and not inside_math
520         and (glyph.prev.width >0)
521         and not nodehasattribute(glyph.prev,iwspaceattributeid)
522     then
523         nodesetattribute(glyph.prev,iwspaceattributeid,1)
524         nodesetattribute(glyph.prev,iwfontattributeid,glyph.font)
525     -- for debugging
526     if ltx.__tag.trace.showspace then
527         __tag_show_spacemark (head,glyph)
528     end
529     end
530 elseif id == PENALTY then
531     local glyph = n
532     -- ltx.__tag.trace.log ("PENALTY ".. n.subtype.."VALUE"..n.penalty,3)
533     if glyph.next and (glyph.next.id == GLUE)
534         and not inside_math and (glyph.next.width >0) and n.subtype==0
535     then
536         nodesetattribute(glyph.next,iwspaceattributeid,1)
537         -- changed 2024-01-18, issue #72
538         nodesetattribute(glyph.next,iwfontattributeid,default_currfontid)
539     -- for debugging
540     if ltx.__tag.trace.showspace then
541         __tag_show_spacemark (head,glyph)
542     end
543     end
544 elseif id == MATH then
545     inside_math = (n.subtype == 0)
546     end
547 end
548 return head
549 end

```

(End of definition for __tag_mark_spaces.)

```

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

```

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

```

550 local function __tag_activate_mark_space ()
551     if not luatexbase.in_callback ("pre_linebreak_filter","markspaces") then
552         luatexbase.add_to_callback("pre_linebreak_filter",__tag_mark_spaces,"markspaces")
553         luatexbase.add_to_callback("hpack_filter",__tag_mark_spaces,"markspaces")
554     end
555 end
556
557 ltx.__tag.func.markspaceon=__tag_activate_mark_space
558
559 local function __tag_deactivate_mark_space ()
560     if luatexbase.in_callback ("pre_linebreak_filter","markspaces") then
561         luatexbase.remove_from_callback("pre_linebreak_filter","markspaces")
562         luatexbase.remove_from_callback("hpack_filter","markspaces")
563     end
564 end

```

```

565
566 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.

```

567 local default_space_char = nodenew(GLYPH)
568 local default_fontid     = fontid("TU/lmr/m/n/10")
569 local default_currfontid = fontid("TU/lmr/m/n/10")
570 default_space_char.char  = 32
571 default_space_char.font  = default_fontid

```

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

```

572 local function __tag_font_has_space (fontid)
573   t= fonts.hashes.identifiers[fontid]
574   if luaotfload.aux.slot_of_name(fontid,"space")
575     or t.characters and t.characters[32] and t.characters[32]["unicode"]==32
576   then
577     return true
578   else
579     return false
580   end
581 end

```

```

__tag_space_chars_shipout
ltx.__tag.func.space_chars_shipout

```

These is the main function to insert real space chars. It inserts a glyph before every glue which has been marked previously. The attributes are copied from the glue, so if the tagging is done later, it will be tagged like it.

```

582 local function __tag_space_chars_shipout (box)
583   local head = box.head
584   if head then
585     for n in node.traverse(head) do
586       local spaceattr = -1
587       if not nodehasattribute(n,iwspaceoffattributeid) then
588         spaceattr = nodegetattribute(n,iwspaceattributeid) or -1
589       end
590       if n.id == HLIST then -- enter the hlist
591         __tag_space_chars_shipout (n)
592       elseif n.id == VLIST then -- enter the vlist
593         __tag_space_chars_shipout (n)
594       elseif n.id == GLUE then
595         if ltx.__tag.trace.showspaces and spaceattr==1 then
596           __tag_show_spacemark (head,n,"0 1 0")
597         end
598         if spaceattr==1 then
599           local space
600           local space_char = node.copy(default_space_char)
601           local curfont     = nodegetattribute(n,iwfontattributeid)
602           ltx.__tag.trace.log ("INFO SPACE-FUNCTION-FONT: ".. tostring(curfont),3)
603           if curfont and
604             -- luaotfload.aux.slot_of_name(curfont,"space")
605             __tag_font_has_space (curfont)
606           then
607             space_char.font=curfont
608           end
609           head, space = node.insert_before(head, n, space_char) --

```

```

610         n.width      = n.width - space.width
611         space.attr = n.attr
612     end
613 end
614 end
615 box.head = head
616 end
617 end
618
619 function ltx.__tag.func.space_chars_shipout (box)
620     __tag_space_chars_shipout (box)
621 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.

```

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

```
654 function ltx.__tag.func.store_struct_mcabs (structnum,mcnum)
655   ltx.__tag.struct[structnum]=ltx.__tag.struct[structnum] or { }
656   ltx.__tag.struct[structnum]["mc"]=ltx.__tag.struct[structnum]["mc"] or { }
657   -- a structure can contain more than on mc chunk, the content should be ordered
658   tableinsert(ltx.__tag.struct[structnum]["mc"],mcnum)
659   ltx.__tag.trace.log("INFO TEX-MC-INTO-STRUCT: "..
660     mcnum.." inserted in struct "..structnum,3)
661   -- but every mc can only be in one structure
662   ltx.__tag.mc[mcnum]= ltx.__tag.mc[mcnum] or { }
663   ltx.__tag.mc[mcnum]["parent"] = structnum
664 end
665
```

(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.

```
666 -- pay attention: lua counts arrays from 1, tex pages from one
667 -- mcid and arrays in pdf count from 0.
668 function ltx.__tag.func.store_mc_in_page (mcnum,mcpagecnt,page)
669   ltx.__tag.page[page] = ltx.__tag.page[page] or {}
670   ltx.__tag.page[page][mcpagecnt] = mcnum
671   ltx.__tag.trace.log("INFO TAG-MC-INTO-PAGE: page " .. page ..
672     ": inserting MCID " .. mcpagecnt .. " => " .. mcnum,3)
673 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)

```
674 local function __tag_update_mc_attributes (head,mcnum,type)
675   for n in node.traverse(head) do
676     node.set_attribute(n,mcntattributeid,mcnum)
677     node.set_attribute(n,mctypeattributeid,type)
678     if n.id == HLIST or n.id == VLIST then
679       __tag_update_mc_attributes (n.list,mcnum,type)
680     end
681   end
682   return head
683 end
684 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.

```
685 --[[
686   Now follows the core function
687   It wades through the shipout box and checks the attributes
688   ARGUMENTS
```

```

689     box: is a box,
690     mcpagecnt: num, the current page cnt of mc (should start at -1 in shipout box), needed for
691     mccntprev: num, the attribute cnt of the previous node/whatever - if different we have a
692     mcopen: num, records if some bdc/emc is open
693     These arguments are only needed for log messages, if not present are replaced by fix strings
694     name: string to describe the box
695     mctypeprev: num, the type attribute of the previous node/whatever
696
697     there are lots of logging messages currently. Should be cleaned up in due course.
698     One should also find ways to make the function shorter.
699 --]]
700
701 function ltx.__tag.func.mark_page_elements (box,mcpagecnt,mccntprev,mcopen,name,mctypeprev)
702     local name = name or ("SOMEBOX")
703     local mctypeprev = mctypeprev or -1
704     local abspage = status.total_pages + 1 -- the real counter is increased
705                                           -- inside the box so one off
706                                           -- if the callback is not used. (???)
707     ltx.__tag.trace.log ("INFO TAG-ABSPAGE: " .. abspage,3)
708     ltx.__tag.trace.log ("INFO TAG-ARGS: pagecnt".. mcpagecnt..
709                         " prev "..mccntprev ..
710                         " type prev "..mctypeprev,4)
711     ltx.__tag.trace.log ("INFO TAG-TRAVERSING-BOX: ".. tostring(name)..
712                         " TYPE ".. node.type(node.getid(box)),3)
713     local head = box.head -- ShipoutBox is a vlist?
714     if head then
715         mcnthead, mctypehead,taghead = __tag_get_mc_cnt_type_tag (head)
716         ltx.__tag.trace.log ("INFO TAG-HEAD: " ..
717                             node.type(node.getid(head))..
718                             " MC"..tostring(mcnthead)..
719                             " => TAG " .. tostring(mctypehead)..
720                             " => ".. tostring(taghead),3)
721     else
722         ltx.__tag.trace.log ("INFO TAG-NO-HEAD: head is "..
723                             tostring(head),3)
724     end
725     for n in node.traverse(head) do
726         local mccnt, mctype, tag = __tag_get_mc_cnt_type_tag (n)
727         local spaceattr = node.getattribute(n,iwspaceattributeid) or -1
728         ltx.__tag.trace.log ("INFO TAG-NODE: "..
729                             node.type(node.getid(n))..
730                             " MC".. tostring(mccnt)..
731                             " => TAG ".. tostring(mctype)..
732                             " => " .. tostring(tag),3)
733         if n.id == HLIST
734         then -- enter the hlist
735             mcopen,mcpagecnt,mccntprev,mctypeprev=
736                 ltx.__tag.func.mark_page_elements (n,mcpagecnt,mccntprev,mcopen,"INTERNAL HLIST",mctypeprev)
737         elseif n.id == VLIST then -- enter the vlist
738             mcopen,mcpagecnt,mccntprev,mctypeprev=
739                 ltx.__tag.func.mark_page_elements (n,mcpagecnt,mccntprev,mcopen,"INTERNAL VLIST",mctypeprev)
740         elseif n.id == GLUE and not n.leader then -- at glue real space chars are inserted, but t
741                                                     -- been done if the previous shipout wandering, so here it
742         elseif n.id == LOCAL_PAR then -- local_par is ignored

```

```

743 elseif n.id == PENALTY then -- penalty is ignored
744 elseif n.id == KERN then -- kern is ignored
745 ltx.__tag.trace.log("INFO TAG-KERN-SUBTYPE: "..
746 node.type(node.getid(n)).." ".n.subtype,4)
747 else
748 -- math is currently only logged.
749 -- we could mark the whole as math
750 -- for inner processing the mlist_to_hlist callback is probably needed.
751 if n.id == MATH then
752 ltx.__tag.trace.log("INFO TAG-MATH-SUBTYPE: "..
753 node.type(node.getid(n)).." ".__tag_get_mathsubtype(n),4)
754 end
755 -- endmath
756 ltx.__tag.trace.log("INFO TAG-MC-COMPARE: current "..
757 mcnt.." prev "..mcpntprev,4)
758 if mcnt~=mcpntprev then -- a new mc chunk
759 ltx.__tag.trace.log("INFO TAG-NEW-MC-NODE: "..
760 node.type(node.getid(n))..
761 " MC"..tostring(mcnt)..
762 "<=> PREVIOUS "..tostring(mcpntprev),4)
763 if mcpnt~=0 then -- there is a chunk open, close it (hope there is only one ...
764 box.list=__tag_insert_emc_node (box.list,n)
765 mcpnt = mcpnt - 1
766 ltx.__tag.trace.log("INFO TAG-INSERT-EMC: " ..
767 mcpnt .. " MCOPTEN = " .. mcpnt,3)
768 if mcpnt ~=0 then
769 ltx.__tag.trace.log("WARN TAG-OPEN-MC: " .. mcpnt,1)
770 end
771 end
772 if ltx.__tag.mc[mcnt] then
773 if ltx.__tag.mc[mcnt]["artifact"] then
774 ltx.__tag.trace.log("INFO TAG-INSERT-ARTIFACT: "..
775 tostring(ltx.__tag.mc[mcnt]["artifact"]),3)
776 if ltx.__tag.mc[mcnt]["artifact"] == "" then
777 box.list = __tag_insert_bmc_node (box.list,n,"Artifact")
778 else
779 box.list = __tag_insert_bdc_node (box.list,n,"Artifact", "/Type /"..ltx.__tag.mc[mcnt]
780 end
781 else
782 ltx.__tag.trace.log("INFO TAG-INSERT-TAG: "..
783 tostring(tag),3)
784 mcpnt = mcpnt +1
785 ltx.__tag.trace.log("INFO TAG-INSERT-BDC: "..mcpnt,3)
786 local dict= "/MCID "..mcpnt
787 if ltx.__tag.mc[mcnt]["raw"] then
788 ltx.__tag.trace.log("INFO TAG-USE-RAW: "..
789 tostring(ltx.__tag.mc[mcnt]["raw"]),3)
790 dict= dict .. " " .. ltx.__tag.mc[mcnt]["raw"]
791 end
792 if ltx.__tag.mc[mcnt]["alt"] then
793 ltx.__tag.trace.log("INFO TAG-USE-ALT: "..
794 tostring(ltx.__tag.mc[mcnt]["alt"]),3)
795 dict= dict .. " " .. ltx.__tag.mc[mcnt]["alt"]
796 end

```

```

797     if ltx.__tag.mc[mccnt]["actualtext"] then
798         ltx.__tag.trace.log("INFO TAG-USE-ACTUALTEXT: "..
799             tostring(ltx.__tag.mc[mccnt]["actualtext"]),3)
800         dict= dict .. " " .. ltx.__tag.mc[mccnt]["actualtext"]
801     end
802     box.list = __tag_insert_bdc_node (box.list,n,tag, dict)
803     ltx.__tag.func.store_mc_kid (mccnt,mcpagecnt,abspage)
804     ltx.__tag.func.store_mc_in_page(mccnt,mcpagecnt,abspage)
805     ltx.__tag.trace.show_mc_data (mccnt,3)
806     end
807     mcopen = mcopen + 1
808     else
809     if tagunmarkedbool.mode == truebool.mode then
810         ltx.__tag.trace.log("INFO TAG-NOT-TAGGED: this has not been tagged, using artifact",2)
811         box.list = __tag_insert_bmc_node (box.list,n,"Artifact")
812         mcopen = mcopen + 1
813     else
814         ltx.__tag.trace.log("WARN TAG-NOT-TAGGED: this has not been tagged",1)
815     end
816     end
817     mccntprev = mccnt
818     end
819     end -- end if
820 end -- end for
821 if head then
822     mccnthead, mctypehead,taghead = __tag_get_mc_cnt_type_tag (head)
823     ltx.__tag.trace.log ("INFO TAG-ENDHEAD: " ..
824         node.type(node.getid(head))..
825         " MC"..tostring(mccnthead)..
826         " => TAG "..tostring(mctypehead)..
827         " => "..tostring(taghead),4)
828     else
829         ltx.__tag.trace.log ("INFO TAG-ENDHEAD: ".. tostring(head),4)
830     end
831     ltx.__tag.trace.log ("INFO TAG-QUITTING-BOX "..
832         tostring(name)..
833         " TYPE ".. node.type(node.getid(box)),4)
834     return mcopen,mcpagecnt,mccntprev,mctypeprev
835 end
836

```

(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.

```

837 function ltx.__tag.func.mark_shipout (box)
838     mcopen = ltx.__tag.func.mark_page_elements (box,-1,-100,0,"Shipout",-1)
839     if mcopen~=0 then -- there is a chunk open, close it (hope there is only one ...
840         local emcnode = __tag_backend_create_emc_node ()
841         local list = box.list
842         if list then
843             list = node.insert_after (list,node.tail(list),emcnode)
844             mcopen = mcopen - 1

```

```

845     ltx.__tag.trace.log ("INFO SHIPOUT-INSERT-LAST-EMC: MCOPEN " .. mcopen,3)
846 else
847     ltx.__tag.trace.log ("WARN SHIPOUT-UPS: this shouldn't happen",0)
848 end
849 if mcopen ~=0 then
850     ltx.__tag.trace.log ("WARN SHIPOUT-MC-OPEN: " .. mcopen,1)
851 end
852 end
853 end

```

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

7 Parenttree

```

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

```

These functions create the parent tree. The second, main function is used in the tagpdf-tree code. TODO check if the tree code can move into the backend code.

```

854 function ltx.__tag.func.fill_parent_tree_line (page)
855     -- we need to get page-> i=kid -> mcnum -> structnum
856     -- pay attention: the kid numbers and the page number in the parent tree start with 0!
857     local numsentry = ""
858     local pdfpage = page-1
859     if ltx.__tag.page[page] and ltx.__tag.page[page][0] then
860         mcchunks=#ltx.__tag.page[page]
861         ltx.__tag.trace.log("INFO PARENTTREE-NUM: page "..
862             page.." has "..mcchunks.." +1 Elements ",4)
863         for i=0,mcchunks do
864             -- what does this log??
865             ltx.__tag.trace.log("INFO PARENTTREE-CHUNKS: "..
866                 ltx.__tag.page[page][i],4)
867         end
868         if mcchunks == 0 then
869             -- only one chunk so no need for an array
870             local mcnum = ltx.__tag.page[page][0]
871             local structnum = ltx.__tag.mc[mcnum]["parent"]
872             local propname = "g__tag_struct_"..structnum.."_prop"
873             --local objref = ltx.__tag.tables[propname]["objref"] or "XXXX"
874             local objref = __tag_pdf_object_ref('__tag/struct',structnum)
875             ltx.__tag.trace.log("INFO PARENTTREE-STRUCT-OBJREF: =====>"..
876                 tostring(objref),5)
877             numsentry = pdfpage .. " [".. objref .. "]"
878             ltx.__tag.trace.log("INFO PARENTTREE-NUMENTRY: page " ..
879                 page.. " num entry = ".. numsentry,3)
880         else
881             numsentry = pdfpage .. " ["
882             for i=0,mcchunks do
883                 local mcnum = ltx.__tag.page[page][i]
884                 local structnum = ltx.__tag.mc[mcnum]["parent"] or 0
885                 local propname = "g__tag_struct_"..structnum.."_prop"
886                 --local objref = ltx.__tag.tables[propname]["objref"] or "XXXX"
887                 local objref = __tag_pdf_object_ref('__tag/struct',structnum)
888                 numsentry = numsentry .. " " .. objref
889             end
890             numsentry = numsentry .. "]" "

```



```

891     ltx.__tag.trace.log("INFO PARENTTREE-NUMENTRY: page " ..
892     page.. " num entry = ".. numsentry,3)
893     end
894     else
895     ltx.__tag.trace.log ("INFO PARENTTREE-NO-DATA: page "..page,3)
896     numsentry = pdfpage.." []"
897     end
898     return numsentry
899 end
900
901 function ltx.__tag.func.output_parenttree (abspage)
902 for i=1,abspage do
903     line = ltx.__tag.func.fill_parent_tree_line (i) .. "^^J"
904     tex.sprint(catlatex,line)
905 end
906 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.

```

907 do
908     local properties = node.get_properties_table()
909     local is_soft_hyphen_prop = 'tagpdf.rewrite-softhyphen.is_soft_hyphen'
910     local hyphen_char = 0x2D
911     local soft_hyphen_char = 0xAD

```

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

```

912     local softhyphen_fonts = setmetatable({}, {__index = function(t, fid)
913     local fdir = identifiers[fid]
914     local format = fdir and fdir.format
915     local result = (format == 'opentype' or format == 'truetype')
916     local characters = fdir and fdir.characters
917     result = result and (characters and characters[soft_hyphen_char]) ~= nil
918     t[fid] = result
919     return result
920     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.

```

921     local function process_softhyphen_pre(head, _context, _dir)
922     if softhyphenbool.mode ~= truebool.mode then return true end
923     for disc, sub in node.traverse_id(DISC, head) do
924     if sub == explicit_disc or sub == regular_disc then
925     for n, _ch, _f in node.traverse_char(disc.pre) do
926     local props = properties[n]
927     if not props then
928     props = {}
929     properties[n] = props
930     end
931     props[is_soft_hyphen_prop] = true
932     end
933     end
934     end
935     return true

```

936 end

937

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

938 local function process_softhyphen_post(head, _context, _dir)

939 if softhyphenbool.mode ~= truebool.mode then return true end

940 for disc, sub in node.traverse_id(DISC, head) do

941 for n, ch, fid in node.traverse_glyph(disc.pre) do

942 local props = properties[n]

943 if softhyphen_fonts[fid] and ch == hyphen_char and props and props[is_soft_hyphen_pro

944 n.char = soft_hyphen_char

945 props.glyph_info = nil

946 end

947 end

948 end

949 return true

950 end

951

952 luatexbase.add_to_callback('pre_shaping_filter', process_softhyphen_pre, 'tagpdf.rewrite-
softhyphen')

953 luatexbase.add_to_callback('post_shaping_filter', process_softhyphen_post, 'tagpdf.rewrite-
softhyphen')

954 end

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

955 \langle /lua \rangle

Part X

The `tagpdf-roles` module

Tags, roles and namespace code

Part of the `tagpdf` package

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

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

The key-value list knows the following keys:

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

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

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

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

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

`\tag_check_child: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} {2025-01-12} {0.991}
4 {part of tagpdf - code related to roles and structure names}
5 </header>
```

1 Code related to roles and structure names

⁶ `*package`

1.1 Variables

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

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

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

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

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

This are the main variables used by the code:

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

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

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

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

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

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

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

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

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

`\l__tag_role_debug_prop` This property is used to pass some info around for info messages or debugging.

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

```
7 \prop_new:N \g__tag_role_tags_NS_prop
```

(End of definition for `\g__tag_role_tags_NS_prop`.)

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

pdf <http://iso.org/pdf/ssn> (default)

user `\c__tag_role_userNS_id_str` (random id, for user tags)

latex <https://www.latex-project.org/ns/dflt>

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

More namespaces are possible and their objects references and their rolemaps must be collected so that an array can be written to the StructTreeRoot at the end (see tagpdf-tree). We use a prop to store the object reference as it will be needed rather often.

```
9 \prop_new:N \g__tag_role_NS_prop
```

(End of definition for `\g__tag_role_NS_prop`.)

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

```
10 \prop_new:N \g__tag_role_index_prop
```

(End of definition for `\g__tag_role_index_prop`.)

`\l__tag_role_debug_prop` This variable is used to pass more infos to debug messages.

```
11 \prop_new:N \l__tag_role_debug_prop
```

(End of definition for `\l__tag_role_debug_prop`.)

We need also a bunch of temporary variables.

```
\l__tag_role_tag_tmpa_tl
```

```
\l__tag_role_tag_namespace_tmpa_tl
```

```
12 \tl_new:N \l__tag_role_tag_tmpa_tl
```

```
\l__tag_role_tag_namespace_tmpb_tl %
```

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

```
\l__tag_role_role_tmpa_tl
```

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

```
\l__tag_role_role_namespace_tmpa_tl
```

```
15 \tl_new:N \l__tag_role_role_tmpa_tl
```

```
\l__tag_role_role_namespace_tmpa_tl
```

```
16 \tl_new:N \l__tag_role_role_namespace_tmpa_tl
```

```
\l__tag_role_tmpa_seq
```

```
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 \prop_new:N \g__tag_role_rolemap_prop
```

(End of definition for `g__tag_role/RoleMap_dict` and `\g__tag_role_rolemap_prop`.)

```
\__tag_role_NS_new:nnn \__tag_role_NS_new:nnn {<shorthand>} {<URI-ID>} {<Schema>}
```

```
\__tag_role_NS_new:nnn
```

```
20 \pdf_version_compare:NnTF < {2.0}
21 {
22   \cs_new_protected:Npn \__tag_role_NS_new:nnn #1 #2 #3
23   {
24     \prop_new:c { g__tag_role_NS_#1_prop }
25     \prop_new:c { g__tag_role_NS_#1_class_prop }
26     \prop_gput:Nne \g__tag_role_NS_prop {#1}{#2}
27   }
28 }
29 {
30   \cs_new_protected:Npn \__tag_role_NS_new:nnn #1 #2 #3
31   {
32     \prop_new:c { g__tag_role_NS_#1_prop }
33     \prop_new:c { g__tag_role_NS_#1_class_prop }
34     \pdf_object_new:n {tag/NS/#1}
35     \pdfdict_new:n {g__tag_role/namespace_#1_dict}
36     \pdf_object_new:n {__tag_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         \prop_gput:cnn {g__tag_role_NS_#2_prop} {#1}{{}}
90         \prop_gput:Nnn \g__tag_role_tags_class_prop {#1}{#3}
91         \prop_gput:cnn {g__tag_role_NS_#2_class_prop} {#1}{--UNUSED--}
92     }
93 }
94 {
95     \cs_new_protected:Npn \__tag_role_alloctag:nnn #1 #2 #3
96     {
97         \prop_gput:Nnn \g__tag_role_tags_NS_prop {#1}{#2}
98         \prop_gput:cnn {g__tag_role_NS_#2_prop} {#1}{{}}
99         \prop_gput:Nnn \g__tag_role_tags_class_prop {#1}{#3}
100        \prop_gput:cnn {g__tag_role_NS_#2_class_prop} {#1}{--UNUSED--}
101    }
102 }
103 }
104 {
105     \sys_if_engine luatex:TF
106     {
107         \cs_new_protected:Npn \__tag_role_alloctag:nnn #1 #2 #3 %#1 tagname, ns, type
108         {
109             \lua_now:e { ltx.__tag.func.alloctag ('#1') }
110             \prop_gput:Nnn \g__tag_role_tags_NS_prop {#1}{#2}
111             \prop_gput:cnn {g__tag_role_NS_#2_prop} {#1}{{}}
112             \prop_gput:Nnn \g__tag_role_tags_class_prop {#1}{--UNUSED--}
113             \prop_gput:cnn {g__tag_role_NS_#2_class_prop} {#1}{#3}
114         }
115     }
116     {
117         \cs_new_protected:Npn \__tag_role_alloctag:nnn #1 #2 #3
118         {
119             \prop_gput:Nnn \g__tag_role_tags_NS_prop {#1}{#2}
120             \prop_gput:cnn {g__tag_role_NS_#2_prop} {#1}{{}}
121             \prop_gput:Nnn \g__tag_role_tags_class_prop {#1}{--UNUSED--}
122             \prop_gput:cnn {g__tag_role_NS_#2_class_prop} {#1}{#3}
123         }
124     }
125 }
126 \cs_generate_variant:Nn \__tag_role_alloctag:nnn {nnV}

```

(End of definition for __tag_role_alloctag:nnn.)

1.3.1 pdf 1.7 and earlier

`__tag_role_add_tag:nn` The pdf 1.7 version has only two arguments: new and rolemap name. The role must be an existing tag and should not be empty. We allow to change the role of an existing tag: as the rolemap is written at the end not confusion can happen.

```

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

```

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

```

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

```

(End of definition for __tag_role_add_tag:nn.)

For the parent-child test we must be able to get the role. We use the same number of arguments as for the 2.0 command. If there is no role, we assume a standard tag.

__tag_role_get:nnNN

```

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

```

(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
169 \\cs_new_protected:Nn \\_tag_role_add_tag:nnnn %tag/namespace/role/namespace
170 {
171   \\_tag_check_add_tag_role:nnn {#1/#2}{#3}{#4}
172   \\int_compare:nNnT {\\l__tag_loglevel_int} > { 0 }
173   {
174     \\msg_info:nnn { tag }{new-tag}{#1}
175   }
176   \\prop_if_exist:cTF
177   { g__tag_role_NS_#4_class_prop }
178   {
179     \\prop_get:cnN { g__tag_role_NS_#4_class_prop } {#3}\\l__tag_tmpa_tl
180     \\quark_if_no_value:NT \\l__tag_tmpa_tl
181     {
182       \\tl_set:Nn\\l__tag_tmpa_tl{--UNKNOWN--}
183     }
184   }
185   { \\tl_set:Nn\\l__tag_tmpa_tl{--UNKNOWN--} }
186   \\_tag_role_alloctag:nnV {#1}{#2}\\l__tag_tmpa_tl
```

Do not remap standard tags. TODO add warning?

```
187   \\tl_if_in:nnF {-pdf-pdf2-mathml-}{-#2-}
188   {
189     \\pdfdict_gput:nne {g__tag_role/RoleMapNS_#2_dict}{#1}
190     {
191       [
192         \\pdf_name_from_unicode_e:n{#3}
193         \\c_space_tl
194         \\pdf_object_ref:n {tag/NS/#4}
195       ]
196     }
197   }
```

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

```
198   \\tl_if_empty:nF { #2 }
199   {
200     \\prop_get:cnN { g__tag_role_NS_#4_prop } {#3}\\l__tag_tmpa_tl
201     \\quark_if_no_value:NTF \\l__tag_tmpa_tl
202     {
203       \\prop_gput:cne { g__tag_role_NS_#2_prop } {#1}
204       {f\\tl_to_str:n{#3}}{\\tl_to_str:n{#4}}
205     }
206     {
207       \\prop_gput:cno { g__tag_role_NS_#2_prop } {#1}{\\l__tag_tmpa_tl}
208     }
209   }
```

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

```
210   \\bool_if:NT \\l__tag_role_update_bool
211   {
212     \\tl_if_empty:nF { #3 }
```

```

213     {
214         \tl_if_eq:nnF{#1}{#3}
215         {
216             \prop_get:NnN \g__tag_role_rolemap_prop {#3}\l__tag_tmpa_tl
217             \quark_if_no_value:NTF \l__tag_tmpa_tl
218             {
219                 \prop_gput:Nne \g__tag_role_rolemap_prop {#1}{\tl_to_str:n{#3}}
220             }
221             {
222                 \prop_gput:NnV \g__tag_role_rolemap_prop {#1}\l__tag_tmpa_tl
223             }
224         }
225     }
226 }
227 }
228 \cs_generate_variant:Nn \__tag_role_add_tag:nnnn {VVVV}

```

(End of definition for __tag_role_add_tag:nnnn.)

For the parent-child test we must be able to get the role. We use the same number of arguments as for the <2.0 command (and assume that we don't need a name space)

__tag_role_get:nnNN

```

229 \pdf_version_compare:NnF < {2.0}
230 {
231     \cs_new:Npn \__tag_role_get:nnNN #1#2#3#4
232     %#1 tag, #2 NS,
233     %#3 tlvar which hold the role tag
234     %#4 tlvar which hold the name of the target NS
235     {
236         \prop_if_exist:cTF {g__tag_role_NS_#2_prop}
237         {
238             \prop_get:cnNTF {g__tag_role_NS_#2_prop} {#1}\l__tag_get_tmpc_tl
239             {
240                 \tl_set:Ne #3 {\exp_last_unbraced:NV\use_i:nn \l__tag_get_tmpc_tl}
241                 \tl_set:Ne #4 {\exp_last_unbraced:NV\use_ii:nn \l__tag_get_tmpc_tl}
242             }
243             {
244                 \msg_warning:nnn { tag } {role-unknown-tag} { #1 }
245                 \tl_set:Nn #3 {#1}
246                 \tl_set:Nn #4 {#2}
247             }
248         }
249         {
250             \msg_warning:nnn { tag } {role-unknown-NS} { #2 }
251             \tl_set:Nn #3 {#1}
252             \tl_set:Nn #4 {#2}
253         }
254     }
255     \cs_generate_variant:Nn \__tag_role_get:nnNN {VVNN}
256 }

```

(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.

```

257 \bool_new:N\l__tag_role_update_bool
258 \bool_set_true:N \l__tag_role_update_bool
259 \pdf_version_compare:NnTF < {2.0}
260 {
261   \cs_new_protected:Npn \__tag_role_read_namespace_line:nw #1#2,#3,#4,#5,#6\q_stop %
262   % #1 NS, #2 tag, #3 rolemap, #4 NS rolemap #5 type
263   {
264     \tl_if_empty:nF { #2 }
265     {
266       \bool_if:NTF \l__tag_role_update_bool
267       {
268         \tl_if_empty:nTF {#5}
269         {
270           \prop_get:NnN \g__tag_role_tags_class_prop {#3}\l__tag_tmpa_tl
271           \quark_if_no_value:NT \l__tag_tmpa_tl
272           {
273             \tl_set:Nn\l__tag_tmpa_tl{--UNKNOWN--}
274           }
275         }
276         {
277           \tl_set:Nn \l__tag_tmpa_tl {#5}
278         }
279         \__tag_role_alloctag:nnV {#2}{#1}\l__tag_tmpa_tl
280         \tl_if_eq:nnF {#2}{#3}
281         {
282           \__tag_role_add_tag:nn {#2}{#3}
283         }
284         \prop_gput:cnn {g__tag_role_NS_#1_prop} {#2}{#{#3}{}}
285       }
286       {
287         \prop_gput:cnn {g__tag_role_NS_#1_prop} {#2}{#{#3}{}}
288         \prop_gput:cnn {g__tag_role_NS_#1_class_prop} {#2}{--UNUSED--}
289       }
290     }
291   }
292 }
293 {
294   \cs_new_protected:Npn \__tag_role_read_namespace_line:nw #1#2,#3,#4,#5,#6\q_stop %
295   % #1 NS, #2 tag, #3 rolemap, #4 NS rolemap #5 type
296   {
297     \tl_if_empty:nF {#2}
298     {
299       \tl_if_empty:nTF {#5}
300       {
301         \prop_get:cnN { g__tag_role_NS_#4_class_prop } {#3}\l__tag_tmpa_tl
302         \quark_if_no_value:NT \l__tag_tmpa_tl

```

```

303         {
304         \tl_set:Nn \l__tag_tmpa_tl {--UNKNOWN--}
305         }
306     }
307     {
308     \tl_set:Nn \l__tag_tmpa_tl {#5}
309     }
310     \__tag_role_alloctag:nnV {#2}{#1}\l__tag_tmpa_tl
311     \bool_lazy_and:nnT
312     { ! \tl_if_empty_p:n {#3} } {! \str_if_eq_p:nn {#1}{pdf2}}
313     {
314     \__tag_role_add_tag:nxxx {#2}{#1}{#3}{#4}
315     }
316     \prop_gput:cnn {g__tag_role_NS_#1_prop} {#2}{#3}{#4}
317 }
318 }
319 }

```

(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

```

320 \cs_new_protected:Npn \__tag_role_read_namespace:nn #1 #2 %name of namespace #2 name of file
321 {
322     \prop_if_exist:cF {g__tag_role_NS_#1_prop}
323     { \msg_warning:nnn {tag}{namespace-unknown}{#1} }
324     \file_if_exist:nTF { tagpdf-ns-#2.def }
325     {
326     \ior_open:Nn \g_tmpa_ior {tagpdf-ns-#2.def}
327     \msg_info:nnn {tag}{read-namespace}{#2}
328     \ior_map_inline:Nn \g_tmpa_ior
329     {
330     \__tag_role_read_namespace_line:nw {#1} ##1,,, \q_stop
331     }
332     \ior_close:N\g_tmpa_ior
333     }
334     {
335     \msg_info:nnn{tag}{namespace-missing}{#2}
336     }
337 }
338

```

(End of definition for __tag_role_read_namespace:nn.)

__tag_role_read_namespace:n

This command reads the default namespace file.

```

339 \cs_new_protected:Npn \__tag_role_read_namespace:n #1 %name of namespace
340 {
341     \__tag_role_read_namespace:nn {#1}{#1}
342 }

```

(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.

```
343 \__tag_role_read_namespace:n {pdf}
344 \__tag_role_read_namespace:n {pdf2}
345 \__tag_role_read_namespace:n {mathml}
```

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

```
346 \bool_set_false:N\l__tag_role_update_bool
347 \__tag_role_read_namespace:n {latex-book}
348 \bool_set_true:N\l__tag_role_update_bool
349 \__tag_role_read_namespace:n {latex}
350 \__tag_role_read_namespace:nn {latex} {latex-lab}
351 \__tag_role_read_namespace:n {pdf}
352 \__tag_role_read_namespace:n {pdf2}
```

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

```
353 \pdf_version_compare:NnTF < {2.0}
354 {
355   \hook_gput_code:nnn {begindocument}{tagpdf}
356   {
357     \bool_lazy_and:nnT
358     {
359       \cs_if_exist_p:N \chapter
360     }
361     {
362       \cs_if_exist_p:N \c@chapter
363     }
364     {
365       \prop_map_inline:cn{g__tag_role_NS_latex-book_prop}
366       {
367         \__tag_role_add_tag:ne {#1}{\use_i:nn #2\c_empty_tl\c_empty_tl}
368       }
369     }
370   }
371 }
372 {
373   \hook_gput_code:nnn {begindocument}{tagpdf}
374   {
375     \bool_lazy_and:nnT
376     {
377       \cs_if_exist_p:N \chapter
378     }
379     {
380       \cs_if_exist_p:N \c@chapter
381     }
382     {
383       \prop_map_inline:cn{g__tag_role_NS_latex-book_prop}
384       {
385         \prop_gput:Nnn \g__tag_role_tags_NS_prop { #1 }{ latex-book }
386         \prop_gput:Nne
387         \g__tag_role_rolemap_prop {#1}{\use_i:nn #2\c_empty_tl\c_empty_tl}
388       }
389     }
390   }
391 }
```

```

390     }
391 }

```

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.

```

392 \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` and `\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!

```

393 \cs_new_protected:Npn \_tag_store_parent_child_rule:nnn #1 #2 #3 % num parent, num child, #3
394 {
395   \intarray_gset:Nnn \g_tag_role_parent_child_intarray
396     { #1#2 }{0\prop_item:Nn\c__tag_role_rules_prop{#3}}
397 }

```

(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

```

398 \int_zero:N \l__tag_tmpa_int

```

Open the file depending on the PDF version

```

399 \pdf_version_compare:NnTF < {2.0}
400 {
401   \ior_open:Nn \g_tmpa_ior {tagpdf-parent-child.csv}
402 }
403 {
404   \ior_open:Nn \g_tmpa_ior {tagpdf-parent-child-2.csv}
405 }

```

Now the main loop over the file

```

406 \ior_map_inline:Nn \g_tmpa_ior
407 {

```

ignore lines containing only comments

```

408   \tl_if_empty:nF{#1}
409   {

```

count the lines ...

```

410     \int_incr:N\l__tag_tmpa_int

```

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

```

411     \seq_set_from_clist:Nn\l__tag_tmpa_seq { #1 }
412     \int_compare:nNnTF {\l__tag_tmpa_int}=1

```

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

```

413     {
414         \seq_map_indexed_inline:Nn \l__tag_tmpa_seq
415         {
416             \prop_gput:Nne\g__tag_role_index_prop
417             {##2}
418             {\int_compare:nNnT{##1}<{10}{0}##1}
419         }
420     }

```

now the data lines.

```

421     {
422         \seq_set_from_clist:Nn\l__tag_tmpa_seq { #1 }

```

get the name of the child tag from the first column

```

423         \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

```

424         \prop_get:NVN \g__tag_role_index_prop \l__tag_tmpa_tl \l__tag_tmpb_tl

```

remove column 2+3

```

425         \seq_pop_left:NN\l__tag_tmpa_seq\l__tag_tmpa_tl
426         \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.

```

427         \seq_map_indexed_inline:Nn \l__tag_tmpa_seq
428         {
429             \exp_args:Nne
430             \__tag_store_parent_child_rule:nnn {##1}{\l__tag_tmpb_tl}{ ##2 }
431         }
432     }
433 }
434 }

```

close the read handle.

```

435 \ior_close:N\g_tmpa_ior

```

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

```

436 \prop_get:NnN\g__tag_role_index_prop{StructTreeRoot}\l__tag_tmpa_tl
437 \prop_gput:Nne\g__tag_role_index_prop{Root}{\l__tag_tmpa_tl}
438 \prop_get:NnN\g__tag_role_index_prop{Hn}\l__tag_tmpa_tl
439 \pdf_version_compare:NnTF < {2.0}
440 {
441     \int_step_inline:nn{6}
442     {
443         \prop_gput:Nne\g__tag_role_index_prop{H#1}{\l__tag_tmpa_tl}
444     }
445 }
446 {
447     \int_step_inline:nn{10}
448     {
449         \prop_gput:Nne\g__tag_role_index_prop{H#1}{\l__tag_tmpa_tl}
450     }

```


all mathml tags are currently handled identically

```

451     \prop_get:NnN\g__tag_role_index_prop {mathml}\l__tag_tmpa_tl
452     \prop_get:NnN\g__tag_role_index_prop {math}\l__tag_tmpb_tl
453     \prop_map_inline:Nn \g__tag_role_NS_mathml_prop
454     {
455         \prop_gput:NnV\g__tag_role_index_prop{#1}\l__tag_tmpa_tl
456     }
457     \prop_gput:NnV\g__tag_role_index_prop{math}\l__tag_tmpb_tl
458 }

```

1.6.2 Retrieving the parent-child rule

`__tag_role_get_parent_child_rule:nnnN`

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

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

```

459 \tl_new:N \l__tag_parent_child_check_tl
460 \cs_new_protected:Npn \__tag_role_get_parent_child_rule:nnnN #1 #2 #3 #4
461 % #1 parent (string) #2 child (string) #3 text for messages (eg. about Div or Rolemapping)
462 % #4 tl for state
463 {
464     \prop_get:NnN \g__tag_role_index_prop{#1}\l__tag_tmpa_tl
465     \prop_get:NnN \g__tag_role_index_prop{#2}\l__tag_tmpb_tl
466     \bool_lazy_and:nnTF
467     { ! \quark_if_no_value_p:N \l__tag_tmpa_tl }
468     { ! \quark_if_no_value_p:N \l__tag_tmpb_tl }
469     {

```

Get the rule from the intarray

```

470         \tl_set:N#4
471         {
472             \intarray_item:Nn
473             \g__tag_role_parent_child_intarray
474             {\l__tag_tmpa_tl\l__tag_tmpb_tl}
475         }

```

If the state is `‡` something is wrong ...

```

476         \int_compare:nNnT
477         {#4} = {\prop_item:Nn\c__tag_role_rules_prop{‡}}
478         {
479             %warn ?

```

we must take the current child from the stack if is already there, depending on location the check is called, this could also remove the parent, but that is ok too.

```

480     }

```

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

```

481     \group_begin:
482     \int_compare:nNnT {\l__tag_tmpa_int*\l__tag_loglevel_int} > { 0 }
483     {
484         \prop_get:NVNF\c__tag_role_rules_num_prop #4 \l__tag_tmpa_tl
485         {
486             \tl_set:Nn \l__tag_tmpa_tl {unknown}

```

```

487     }
488     \tl_set:Nn \l__tag_tmpb_tl {#1}
489     \msg_note:nneee
490     { tag }
491     { role-parent-child }
492     { #1 }
493     { #2 }
494     {
495         #4~(=\l__tag_tmpa_tl')
496         \iow_newline:
497         #3
498     }
499     }
500     \group_end:
501 }
502 {
503     \tl_set:Nn#4 {0}
504     \msg_warning:nneee
505     { tag }
506     {role-parent-child}
507     { #1 }
508     { #2 }
509     { unknown! }
510 }
511 }
512 \cs_generate_variant:Nn\__tag_role_get_parent_child_rule:nnnN {VVVN,VVN}

```

(End of definition for __tag_role_get_parent_child_rule:nnnN.)

`__tag_check_parent_child:nnnnN`

This command translates rolemaps its arguments and then calls `__tag_role_get_parent_child_rule:nnnN`. It does not try to resolve inheritance of Div etc but instead warns that the rule can not be detected in this case. In pdf 2.0 the name spaces of the tags are relevant, so we have arguments for them, but in pdf <2.0 they are ignored and can be left empty.

```

513 \pdf_version_compare:NnTF < {2.0}
514 {
515     \cs_new_protected:Npn \__tag_check_parent_child:nnnnN #1 #2 #3 #4 #5
516     {%#1 parent tag,#2 NS, #3 child tag, #4 NS, #5 tl var
517     {

```

for debugging messages we store the arguments.

```

518     \prop_put:Nnn \l__tag_role_debug_prop {parent} {#1}
519     \prop_put:Nnn \l__tag_role_debug_prop {child} {#3}

```

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

```

520     \prop_get:NnNTF \g__tag_role_index_prop {#1}\l__tag_tmpa_tl
521     {
522         \tl_set:Nn \l__tag_tmpa_tl {#1}
523     }
524     {
525         \prop_get:NnNF \g__tag_role_rolemap_prop {#1}\l__tag_tmpa_tl
526         {
527             \tl_set:Nn \l__tag_tmpa_tl {\q_no_value}
528         }
529     }

```

now the child

```

530     \prop_get:NnNTF \g__tag_role_index_prop {#3}\l__tag_tmpb_tl
531     {
532       \tl_set:Nn \l__tag_tmpb_tl {#3}
533     }
534     {
535       \prop_get:NnNF \g__tag_role_rolemap_prop {#3}\l__tag_tmpb_tl
536       {
537         \tl_set:Nn \l__tag_tmpb_tl {\q_no_value}
538       }
539     }

```

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

```

540     \bool_lazy_and:nnTF
541     { ! \quark_if_no_value_p:N \l__tag_tmpa_tl }
542     { ! \quark_if_no_value_p:N \l__tag_tmpb_tl }
543     {
544       \__tag_role_get_parent_child_rule:VVnN
545       \l__tag_tmpa_tl \l__tag_tmpb_tl
546       {Rolemapped~from:~'#1'~-->~'#3'}
547       #5
548     }
549     {
550       \tl_set:Nn #5 {0}
551       \msg_warning:nneee
552       { tag }
553       {role-parent-child}
554       { #1 }
555       { #3 }
556       { unknown! }
557     }
558   }
559   \cs_new_protected:Npn \__tag_check_parent_child:nnN #1#2#3
560   {
561     \__tag_check_parent_child:nnnnN {#1}{#2}{#3}
562   }
563 }

```

and now the pdf 2.0 version The version with three arguments retrieves the default names space and then calls the full command. Not sure if this will ever be needed but we leave it for now.

```

564 {
565   \cs_new_protected:Npn \__tag_check_parent_child:nnN #1 #2 #3
566   {
567     \prop_get:NnN\g__tag_role_tags_NS_prop {#1}\l__tag_role_tag_namespace_tmpa_tl
568     \prop_get:NnN\g__tag_role_tags_NS_prop {#2}\l__tag_role_tag_namespace_tmpb_tl
569     \str_if_eq:nnT{#2}{MC}{\tl_clear:N \l__tag_role_tag_namespace_tmpb_tl}
570     \bool_lazy_and:nnTF
571     { ! \quark_if_no_value_p:N \l__tag_role_tag_namespace_tmpa_tl }
572     { ! \quark_if_no_value_p:N \l__tag_role_tag_namespace_tmpb_tl }
573     {
574       \__tag_check_parent_child:nVnVN
575       {#1}\l__tag_role_tag_namespace_tmpa_tl
576       {#2}\l__tag_role_tag_namespace_tmpb_tl

```

```

577         #3
578     }
579     {
580     \tl_set:Nn #3 {0}
581     \msg_warning:nneee
582     { tag }
583     {role-parent-child}
584     { #1 }
585     { #2 }
586     { unknown! }
587     }
588 }

```

and now the real command.

```

589 \cs_new_protected:Npn \__tag_check_parent_child:nnnnN #1 #2 #3 #4 #5 %tag,NS,tag,NS, tl va
590 {
591     \prop_put:Nnn \l__tag_role_debug_prop {parent} {#1/#2}
592     \prop_put:Nnn \l__tag_role_debug_prop {child} {#3/#4}

```

If the namespace is empty, we assume a standard tag, otherwise we retrieve the rolemap-
ping from the namespace

```

593     \tl_if_empty:nTF {#2}
594     {
595     \tl_set:Nn \l__tag_tmpa_tl {#1}
596     }
597     {
598     \prop_if_exist:cTF { g__tag_role_NS_#2_prop }
599     {
600     \prop_get:cnNTF
601     { g__tag_role_NS_#2_prop }
602     {#1}
603     \l__tag_tmpa_tl
604     {
605     \tl_set:Ne \l__tag_tmpa_tl {\tl_head:N\l__tag_tmpa_tl}
606     \tl_if_empty:NT\l__tag_tmpa_tl
607     {
608     \tl_set:Nn \l__tag_tmpa_tl {#1}
609     }
610     }
611     {
612     \tl_set:Nn \l__tag_tmpa_tl {\q_no_value}
613     }
614     }
615     {
616     \msg_warning:nnn { tag } {role-unknown-NS} { #2}
617     \tl_set:Nn \l__tag_tmpa_tl {\q_no_value}
618     }
619     }

```

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

```

620     \tl_if_empty:nTF {#4}
621     {
622     \tl_set:Nn \l__tag_tmpb_tl {#3}
623     }

```

```

624     {
625       \prop_if_exist:cTF { g__tag_role_NS_#4_prop }
626       {
627         \prop_get:cnNTF
628         { g__tag_role_NS_#4_prop }
629         {#3}
630         \l__tag_tmpb_tl
631         {
632           \tl_set:Ne \l__tag_tmpb_tl { \tl_head:N\l__tag_tmpb_tl }
633           \tl_if_empty:NT\l__tag_tmpb_tl
634           {
635             \tl_set:Nn \l__tag_tmpb_tl {#3}
636           }
637         }
638         {
639           \tl_set:Nn \l__tag_tmpb_tl {\q_no_value}
640         }
641       }
642       {
643         \msg_warning:nnn { tag } {role-unknown-NS} { #4}
644         \tl_set:Nn \l__tag_tmpb_tl {\q_no_value}
645       }
646     }

```

and now get the relation

```

647     \bool_lazy_and:nnTF
648     { ! \quark_if_no_value_p:N \l__tag_tmpa_tl }
649     { ! \quark_if_no_value_p:N \l__tag_tmpb_tl }
650     {
651       \__tag_role_get_parent_child_rule:VVnN
652       \l__tag_tmpa_tl \l__tag_tmpb_tl
653       {Rolemapped~from~'#1/#2'--->~'#3\str_if_empty:nF{#4}{/#4}' }
654       #5
655     }
656     {
657       \tl_set:Nn #5 {0}
658       \msg_warning:nneee
659       { tag }
660       {role-parent-child}
661       { #1 }
662       { #3 }
663       { unknown! }
664     }
665   }
666 }
667 \cs_generate_variant:Nn\__tag_check_parent_child:nnN {VVN}
668 \cs_generate_variant:Nn\__tag_check_parent_child:nnnnN {VVVVN,nVnVN,VVnnN}
669 \end{package}

```

(End of definition for __tag_check_parent_child:nnnnN.)

\tag_check_child:nnTF

```

670 <base>\prg_new_protected_conditional:Npnn \tag_check_child:nn #1 #2 {T,F,TF}{\prg_return_true}
671 *package
672 \prg_set_protected_conditional:Npnn \tag_check_child:nn #1 #2 {T,F,TF}

```

```

673 {
674   \seq_get:NN\g__tag_struct_stack_seq\l__tag_tpa_tl
675   \__tag_struct_get_parentrole:eNN
676     {\l__tag_tpa_tl}
677     \l__tag_get_parent_tpa_tl
678     \l__tag_get_parent_tpb_tl
679   \__tag_check_parent_child:VVnnN
680     \l__tag_get_parent_tpa_tl
681     \l__tag_get_parent_tpb_tl
682     {#1}{#2}
683     \l__tag_parent_child_check_tl
684   \int_compare:nNnTF { \l__tag_parent_child_check_tl } < {0}
685     {\prg_return_false:}
686     {\prg_return_true:}
687 }

```

(End of definition for `\tag_check_child:nTF`. This function is documented on page 163.)

1.7 Remapping of tags

In some context it can be necessary to remap or replace the tags. That means instead of `tag=H1` or `tag=section` one wants the effect of `tag=Span`. Or instead of `tag=P` one wants `tag=Code`.

The following command provide some general interface for this. The core idea is that before a tag is set it is fed through a function that can change it. We want to be able to chain such functions, so all of them manipulate the same variables.

```

\l__tag_role_remap_tag_tl
\l__tag_role_remap_NS_tl
688 \tl_new:N \l__tag_role_remap_tag_tl
689 \tl_new:N \l__tag_role_remap_NS_tl

```

(End of definition for `\l__tag_role_remap_tag_tl` and `\l__tag_role_remap_NS_tl`.)

`__tag_role_remap:` This function is used in the structure and the mc code before using a tag. By default it does nothing with the `tl` vars. Perhaps this should be a hook?

```

690 \cs_new_protected:Npn \__tag_role_remap: { }

```

(End of definition for `__tag_role_remap:.`)

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

```

691 \cs_set_eq:NN \__tag_role_remap_id: \__tag_role_remap:

```

(End of definition for `__tag_role_remap_id:.`)

1.8 Key-val user interface

The user interface uses the key `add-new-tag`, which takes either a keyval list as argument, or a tag/role.

```

tag (rolemap-key)
tag-namespace (rolemap-key)
role (rolemap-key)
role-namespace (rolemap-key)
role/new-tag (setup-key)
add-new-tag (deprecated)

692 \keys_define:nn { __tag / tag-role }
693 {
694   ,tag .tl_set:N = \l__tag_role_tag_tmpa_tl
695   ,tag-namespace .tl_set:N = \l__tag_role_tag_namespace_tmpa_tl
696   ,role .tl_set:N = \l__tag_role_role_tmpa_tl
697   ,role-namespace .tl_set:N = \l__tag_role_role_namespace_tmpa_tl
698 }
699
700 \keys_define:nn { __tag / setup }
701 {
702   role/mathml-tags .bool_gset:N = \g__tag_role_add_mathml_bool
703   ,role/new-tag .code:n =
704   {
705     \keys_set_known:nnnN
706     {__tag/tag-role}
707     {
708       tag-namespace=user,
709       role-namespace=, %so that we can test for it.
710       #1
711       }{__tag/tag-role}\l__tag_tmpa_tl
712     \tl_if_empty:NF \l__tag_tmpa_tl
713     {
714       \exp_args:NNno \seq_set_split:Nnn \l__tag_tmpa_seq { / } {\l__tag_tmpa_tl/}
715       \tl_set:Ne \l__tag_role_tag_tmpa_tl { \seq_item:Nn \l__tag_tmpa_seq {1} }
716       \tl_set:Ne \l__tag_role_role_tmpa_tl { \seq_item:Nn \l__tag_tmpa_seq {2} }
717     }
718     \tl_if_empty:NT \l__tag_role_role_namespace_tmpa_tl
719     {
720       \prop_get:NVNTF
721       \g__tag_role_tags_NS_prop
722       \l__tag_role_role_tmpa_tl
723       \l__tag_role_role_namespace_tmpa_tl
724       {
725         \prop_if_in:NVF\g__tag_role_NS_prop \l__tag_role_role_namespace_tmpa_tl
726         {
727           \tl_set:Nn \l__tag_role_role_namespace_tmpa_tl {user}
728         }
729       }
730       {
731         \tl_set:Nn \l__tag_role_role_namespace_tmpa_tl {user}
732       }
733     }
734     \pdf_version_compare:NnTF < {2.0}
735     {
736       %TODO add check for emptyness?
737       \__tag_role_add_tag:VV
738       \l__tag_role_tag_tmpa_tl
739       \l__tag_role_role_tmpa_tl
740     }
741     {
742       \__tag_role_add_tag:VVVV
743       \l__tag_role_tag_tmpa_tl
744       \l__tag_role_tag_namespace_tmpa_tl

```

```

745     \l__tag_role_role_tmpa_tl
746     \l__tag_role_role_namespace_tmpa_tl
747   }
748 }
749 ,role/map-tags .choice:
750 ,role/map-tags/false .code:n = { \socket_assign_plug:nn { tag/struct/tag } {latex-
tags} }
751 ,role/map-tags/pdf .code:n = { \socket_assign_plug:nn { tag/struct/tag } {pdf-
tags} }

752 ,role/user-NS .code:n =
753 {
754   \pdf_version_compare:NnF < {2.0}
755   {
756     \pdf_string_from_unicode:nnN{utf8/string}{https://www.latex-project.org/ns/local/#1}
757     \tl_if_empty:NF \l__tag_tmpa_str
758     {
759       \pdfdict_gput:nne
760       {g__tag_role/Namespace_user_dict}
761       {NS}
762       {\l__tag_tmpa_str}
763     }
764   }
765 }

```

deprecated names

```

766   , mathml-tags .bool_gset:N = \g__tag_role_add_mathml_bool
767   , add-new-tag .meta:n = {role/new-tag={#1}}
768 }
769 </package>

```

(End of definition for tag (rolemap-key) and others. These functions are documented on page 163.)

Part XI

The tagpdf-space module

Code related to real space chars

Part of the tagpdf package

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} {2025-01-12} {0.991}
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]{\p
34       activate/spaces/false .code:n = { \RemoveFromHook{shipout/firstpage}[tagpdf/spac
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{ltx.__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: 81 \cs_new_protected:Npn \tag_spacechar_off: {}
\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	
<code>\#</code>	1114, 1118
<code>\\</code>	10, 23, 27, 28, 44, 49, 50, 51, 56, 58, 60, 67, 70, 72, 78, 80, 91, 261, 262, 263, 413, 476, 484
<code>_</code>	454, 465
A	
<code>activate_ (setup-key)</code>	37, <u>287</u>
<code>activate-all</code> (deprecated) (key)	<u>1</u>
<code>activate-mc</code> (deprecated) (key)	<u>1</u>
<code>activate-struct</code> (deprecated) (key) ...	<u>1</u>
<code>activate-tree</code> (deprecated) (key)	<u>1</u>
<code>activate/all</code> (key)	<u>1</u> , <u>260</u>
<code>activate/mc</code> (key)	<u>1</u> , <u>260</u>
<code>activate/socket_ (setup-key)</code>	<u>287</u>
<code>activate/softhyphen</code> (key)	<u>1</u> , <u>294</u>
<code>activate/space_ (setup-key)</code>	<u>185</u>
<code>activate/spaces</code> (key)	<u>1</u>
<code>activate/spaces_ (setup-key)</code>	<u>6</u>
<code>activate/struct</code> (key)	<u>1</u> , <u>260</u>
<code>activate/struct-dest</code> (key)	<u>1</u> , <u>260</u>
<code>activate/tagunmarked</code> (key)	<u>1</u> , <u>291</u>
<code>activate/tree</code> (key)	<u>1</u> , <u>260</u>
<code>actualtext</code> (key)	<u>1</u> , <u>526</u>
<code>actualtext_ (mc-key)</code>	73, <u>257</u> , <u>458</u>
<code>add-new-tag_ (deprecated)</code>	<u>692</u>
<code>add-new-tag_ (setup-key)</code>	<u>163</u>
<code>\AddToHook</code>	13, 16, 33, 58, 71, 78, 107, 305, 389, 411, 545, 547, 548, 552, 556, 563, 592, 644
<code>AF</code> (key)	<u>1</u> , <u>687</u>
<code>AFinline</code> (key)	<u>1</u> , <u>687</u>
<code>AFinline-o</code> (key)	<u>1</u> , <u>687</u>
<code>AFref</code> (key)	<u>1</u> , <u>687</u>
<code>alt</code> (key)	<u>1</u> , <u>526</u>
<code>alt_ (mc-key)</code>	73, <u>257</u> , <u>458</u>
<code>artifact_ (mc-key)</code>	73, <u>257</u> , <u>458</u>
artifact-bool internal commands:	
<code>__artifact-bool</code>	<u>121</u>
artifact-type internal commands:	
<code>__artifact-type</code>	<u>121</u>
<code>attr-unknown</code>	<u>21</u> , <u>84</u>
<code>attribute</code> (key)	<u>1</u> , <u>1295</u>
<code>attribute-class</code> (key)	<u>1</u> , <u>1261</u>
B	
benchmark commands:	
<code>\benchmark_tic:</code>	493, 495
<code>\benchmark_toc:</code>	496
bool commands:	
<code>\bool_gset_eq:NN</code> ..	664, 679, 691, 709
<code>\bool_gset_false:N</code>	50, 51, 61, 240, 446, 665, 692
<code>\bool_gset_true:N</code>	47, 49, 56, 131, 179, 374
<code>\bool_if:NTF</code>	9, 13, 18, 31, 40, 40, 67, 75, 80, 85, 91, 96, 111, 146, 194, 202, 210, 228, 239, 241, 258, 266, 297, 314, 323, 323, 343, 378, 381, 395, 440, 451, 462, 476, 478, 495, 503, 528, 535, 596, 659, 674, 686, 704, 920, 981
<code>\bool_if:nTF</code>	6, 363
<code>\bool_if_exist_p:N</code>	44
<code>\bool_lazy_all:nTF</code>	119
<code>\bool_lazy_and:nnTF</code>	43, 153, 163, 307, 311, 357, 375, 399, 466, 539, 540, 570, 647
<code>\bool_lazy_and_p:nn</code>	8
<code>\bool_new:N</code> ...	7, 16, 20, 21, 41, 42, 63, 73, 126, 127, 128, 129, 130, 132, 134, 136, 137, 138, 257, 325, 326, 655
<code>\bool_set_false:N</code>	17, 180, 205, 206, 207, 229, 230, 231, 241, 346, 423, 632, 658, 685
<code>\bool_set_true:N</code>	16, 133, 135, 215, 216, 217, 240, 241, 242, 258, 348, 422, 631
<code>\box</code>	408
box commands:	
<code>\box_dp:N</code>	180, 184
<code>\box_ht:N</code>	170
<code>\box_new:N</code>	121, 122
<code>\box_set_dp:Nn</code>	178, 180
<code>\box_set_eq:NN</code>	193
<code>\box_set_ht:Nn</code>	177, 179
<code>\box_use_drop:N</code>	182, 186
<code>\boxmaxdepth</code>	85, 181
C	
c@g internal commands:	
<code>\c@g__tag_MCID_abs_int</code>	11, 15, 28, 37, 50, 57, 60, 68, 74, 100, 138, 174, 182, 244, 247, 274, 279, 308, 349, 356, 421
<code>\c@g__tag_parenttree_obj_int</code>	<u>154</u> , 474

exp commands:		int commands:	
\exp_args:Ne	122, 479	\int_abs:n	145
\exp_args:NNe	86, 89, 94, 195, 215	\int_case:mnTF	99, 114, 309
\exp_args:Nne	79, 342, 346, 429, 464, 497	\int_compare:nNnTF	22, 57, 70, 97, 116, 132, 140, 150, 172, 173, 188, 213, 218, 237, 264, 267, 292, 298, 385, 392, 399, 406, 406, 412, 418, 426, 434, 441, 449, 456, 476, 482, 569, 578, 684, 867, 933, 1079, 1142
\exp_args:NNne	94	\int_compare:nTF	
\exp_args:NNno	714	179, 332, 1281, 1283, 1285, 1309, 1335	
\exp_args:NV	198, 204, 352, 381, 392, 397	\int_compare_p:nNn	544
\exp_last_unbraced:NV		\int_decr:N	212, 237
... 186, 187, 240, 241, 462, 466, 1024		\int_eval:n	138, 190, 296, 313, 388, 477, 485, 541, 546, 549, 709, 752, 771, 837, 838, 839, 840, 841, 947, 975, 976, 978, 989, 1291
\exp_not:n	227, 246, 313	\int_gincr:N	182, 244, 274, 345, 349, 349, 353, 356, 357, 363, 367, 371, 375, 474, 695, 825, 836
		\int_gset:Nn	7, 81, 157, 292
		\int_gzero:N	300
		\int_if_zero:nTF	
		... 212, 213, 237, 238, 473, 481	
		\int_incr:N	92, 204, 228, 410
		\int_new:N	77, 120, 125, 200, 268, 328, 329, 330, 331, 687
		\int_rand:n	61, 62, 64, 66, 68, 70, 71
		\int_set:Nn	279, 282, 285, 286, 287
		\int_step_inline:nn	441, 447
		\int_step_inline:nnn	25, 90, 261
		\int_step_inline:nnnn	
		... 148, 173, 176, 193, 317, 323	
		\int_to_arabic:n	145, 147
		\int_to_Hex:n	61, 62, 64, 66, 68, 70, 71
		\int_use:N	11, 15, 18, 28, 37, 39, 50, 57, 58, 60, 68, 74, 75, 99, 100, 100, 101, 115, 138, 140, 165, 172, 174, 203, 220, 227, 234, 246, 247, 253, 271, 279, 308, 387, 421, 454, 465, 499, 531, 567, 574, 575, 579, 583, 584, 593, 609, 617, 671, 682, 698, 701, 704, 745, 747, 764, 766, 845, 851, 856, 863, 866, 883, 901, 910, 955, 960, 1087, 1150, 1229, 1288, 1339
		\int_zero:N	89, 104, 398
		intarray commands:	
		\intarray_gset:Nnn	295, 395
		\intarray_item:Nn	297, 300, 472
		\intarray_new:Nn	287, 392
		interwordspace _□ (deprecated)	185, <u>6</u>
		ior commands:	
		\ior_close:N	332, 435
		\ior_map_inline:Nn	328, 406
F			
file commands:			
\file_if_exist:nTF	324		
\file_input:n	310		
firstkey (key)	<u>1</u>		
firstkid (key)	<u>526</u>		
flag commands:			
\flag_clear:n	241		
\flag_height:n	178, 253		
\flag_new:n	176		
\flag_raise:n	254		
fnote internal commands:			
__fnote_gput_ref:nn	73		
\fontencoding	6		
\fontfamily	6		
\fontseries	6		
\fontshape	6		
\fontsize	6		
\footins	601		
G			
g internal commands:			
\g_tag_struct_ref_by_dest:	80		
group commands:			
\group_begin:	66, 75, 177, 372, 481, 694, 786, 794, 835		
\group_end:	73, 78, 232, 424, 500, 712, 790, 798, 995		
H			
\halign	<u>41</u>		
\hangindent	406		
\hbox	397		
hbox commands:			
\hbox_set:Nn	171, 172		
hook commands:			
\hook_gput_code:nnn	7, 11, 33, 57, 65, 79, 155, 241, 290, 291, 355, 373, 379, 383, 740, 753, 763, 776		
\hook_new:n	340		
\hook_use:n	345		
I			
\ignorespaces	<u>37</u>		

<code>\msg_new:nnnn</code>	104	pdf commands:
<code>\msg_note:nn</code>	28, 201	<code>\pdf_activate_indexed_structure_-</code>
<code>\msg_note:nnn</code>		destination:
.....	203, 220, 401, 408, 443, 451	<code>\pdf_bdc:nn</code>
<code>\msg_note:nnnn</code>		<code>\pdf_bdc_shipout:nn</code>
.....	226, 245, 387, 394, 428, 436	<code>\pdf_bmc:n</code>
<code>\msg_note:nnnnn</code>	489	<code>\l_pdf_current_structure_-</code>
<code>\msg_redirect_name:nnn</code>	567	destination_tl
<code>\msg_show_item_unbraced:n</code>	278	<code>\pdf_emc:</code>
<code>\msg_show_item_unbraced:nn</code>	269	<code>\pdf_name_from_unicode_e:n</code>
<code>\msg_term:nnnnn</code>	263, 272
<code>\msg_warning:nn</code>	24, 215, 266
<code>\msg_warning:nnm</code>		100, 110, 115,
.....	12, 14, 42, 44, 53, 183, 206,	158, 167, 192, 271, 1246, 1269, 1305
244, 250, 251, 259, 282, 305, 323,		<code>\pdf_object_if_exist:n</code>
616, 640, 643, 653, 1093, 1112, 1156		<code>\pdf_object_if_exist:nTF</code> ..
<code>\msg_warning:nnnn</code>	414, 464, 548	<code>\pdf_object_new:n</code>
<code>\msg_warning:nnnnn</code>
.....	219, 412, 504, 551, 581, 658, 940	105, 34, 36, 153, 255, 302, 313
N		<code>\pdf_object_new_indexed:nn</code> .
<code>namespace_(rolemap-key)</code>	163	<code>\pdf_object_ref:n</code> ..
<code>new-tag</code>	21, 94	105, 56, 130,
<code>newattribute_(deprecated)</code>	106, 1243	132, 134, 140, 194, 310, 327, 745, 807
<code>\newcommand</code>	628, 629	<code>\pdf_object_ref_indexed:nn</code>
<code>\newcounter</code>	6, 8, 154
<code>\NewDocumentCommand</code>	6,	56, 73, 95, 168, 204, 235,
23, 29, 34, 40, 46, 51, 56, 126, 318, 633		251, 410, 470, 963, 1060, 1123, 1164
<code>\newmarks</code>	13	<code>\pdf_object_ref_last:</code>
<code>no-struct-dest (deprecated) (key)</code>	1
<code>\noindent</code>	406	105, 103, 117, 123, 274, 1324
<code>\nointerlineskip</code>	185	<code>\pdf_object_unnamed_write:nn</code> ..
P	
<code>\PackageError</code>	13	99, 110, 119, 266, 1319
<code>\PackageWarning</code>	28, 589	<code>\pdf_object_write:nnn</code>
<code>page/exclude-header-footer_(setup-</code>	
key)	40, 712	250, 274, 303, 322, 329, 334
<code>page/tabsorder (key)</code>	1, 296	<code>\pdf_object_write_indexed:nnnn</code> .
<code>para-flattened_(deprecated)</code>	419
<code>para-hook-count-wrong</code>	21, 104	138, 423
<code>para/flattened_(tool-key)</code>	419	<code>\pdf_pageobject_ref:n</code>
<code>para/maintag_(setup-key)</code>	419	<code>\pdf_string_from_unicode:nnN</code> 42, 756
<code>para/maintag_(tool-key)</code>	419	<code>\pdf_uncompress:</code>
<code>para/tag_(setup-key)</code>	419	288, 290
<code>para/tag_(tool-key)</code>	419	<code>\pdf_version_compare:NnTF</code>
<code>para/tagging_(setup-key)</code>	39, 419
<code>para/tagging_(tool-key)</code>	419	20, 81, 127, 150, 156, 229,
<code>\PARALABEL</code>	521	259, 316, 353, 399, 439, 513, 734, 754
<code>paratag_(deprecated)</code>	419	pdfannot commands:
<code>paratagging_(deprecated)</code>	39, 419	<code>\pdfannot_dict_put:nnn</code>
<code>paratagging-show_(deprecated)</code> ..	39, 419
<code>parent (key)</code>	1, 526	141, 747, 770, 788, 793
		<code>\pdfannot_link_ref_last:</code> ..
		757, 780
		pdfdict commands:
		<code>\pdfdict_gput:nnn</code>
	
		38, 45, 53, 189, 269, 326, 759
		<code>\pdfdict_if_empty:nTF</code>
		320
		<code>\pdfdict_new:n</code>
		18, 35, 37
		<code>\pdfdict_put:nnn</code> ..
		787, 788, 795, 796
		<code>\pdfdict_use:n</code>
		276, 324, 331
		<code>\pdffakespace</code>
		39, 316
		pdffile commands:
		<code>\pdffile_embed_stream:nnN</code> .
		688, 696
		<code>\pdffile_embed_stream:nnn</code>
		142
		<code>\pdfglyphtounicode</code>
		30
		<code>\pdfinterwordspaceoff</code>
		187, 117
		<code>\pdfinterwordspaceon</code>
		187, 33, 121

pdfmanagement commands:	
\pdfmanagement_add:nnn	51, 69, 70, 298, 300, 302, 385
\pdfmanagement_if_active_p:	9, 10
\pdfmanagement_remove:nn	304
pdfmanagement internal commands:	
\l_pdfmanagement_delayed_	
shipout_bool	44, 45
prg commands:	
\prg_do_nothing:	82, 102, 117, 288, 380, 381, 382, 383, 731, 732, 733, 734
\prg_generate_conditional_	
variant:Nnn	139
\prg_new_conditional:Nnn	68, 226
\prg_new_conditional:Npnn	112, 136, 151, 161, 355, 361, 372
\prg_new_eq_conditional:NNn	82, 233
\prg_new_protected_conditional:Npnn	670
\prg_replicate:nn	144
\prg_return_false:	78, 113, 131, 142, 145, 158, 168, 230, 358, 370, 376, 685
\prg_return_true:	79, 128, 141, 155, 165, 229, 359, 369, 375, 670, 686
\prg_set_conditional:Npnn	117
\prg_set_protected_conditional:Npnn	672
process commands:	
process_softhyphen_pre _{UUUU} process_	
softhyphen_post	907
\ProcessOptions	53
prop commands:	
\prop_clear:N	175
\prop_count:N	196
\prop_get:NnN	137, 145, 179, 200, 215, 216, 270, 301, 408, 424, 436, 438, 451, 452, 464, 465, 567, 568, 935
\prop_get:NnNTF	43, 96, 160, 168, 181, 198, 201, 216, 235, 238, 284, 407, 484, 520, 525, 530, 535, 600, 627, 632, 645, 720, 895, 1025, 1107, 1184
\prop_gput:Nnn	26, 30, 31, 33, 34, 56, 88, 89, 90, 91, 94, 97, 98, 98, 99, 100, 101, 103, 110, 111, 112, 113, 119, 120, 121, 122, 143, 144, 148, 151, 165, 183, 203, 207, 219, 222, 262, 284, 284, 287, 288, 316, 333, 385, 386, 409, 411, 416, 437, 443, 449, 455, 457, 530, 977, 988, 1062, 1125, 1245, 1277, 1324
\prop_gremove:Nn	136, 138
\prop_gset_eq:NN	137, 974
\prop_if_exist:NTF	176, 202, 236, 322, 405, 598, 625, 1047, 1104
\prop_if_exist_p:N	541
\prop_if_in:NnTF	72, 130, 173, 181, 280, 725, 1273, 1313, 1317
\prop_item:Nn	41, 76, 145, 185, 187, 224, 292, 396, 477, 482, 507, 516, 985, 1322, 1329
\prop_map_function:NN	267
\prop_map_inline:Nn	82, 260, 265, 286, 318, 365, 378, 383, 453
\prop_map_tokens:Nn	336
\prop_new:N	7, 8, 9, 10, 11, 11, 19, 24, 25, 32, 33, 116, 135, 180, 838, 1238, 1241
\prop_new_linked:N	17, 62, 67, 69, 181, 1239
\prop_put:Nnn	144, 182, 518, 519, 591, 592
\prop_show:N	68, 95, 189, 971, 992, 1291, 1318
property commands:	
\property_gset:nnnn	270
\property_new:nnnn	163, 166, 170, 173, 177
\property_record:nn	152
\property_ref:nn	104, 157
\property_ref:nnn	41, 156, 161, 180, 184, 201, 202, 202, 277, 334, 345, 461, 1048, 1054, 1057, 1063, 1070
\providecommand	62, 63, 64, 65, 66, 69, 70, 323, 594, 595
\ProvidesExplFile	3
\ProvidesExplPackage	3, 3, 3, 3, 3, 3, 3, 7, 7, 26, 37, 1234
Q	
\quad	235, 236
quark commands:	
\q_no_value	527, 537, 612, 617, 639, 644
\quark_if_no_value:NTF	138, 146, 180, 201, 217, 271, 302, 939
\quark_if_no_value_p:N	467, 468, 541, 542, 571, 572, 648, 649
\q_stop	261, 294, 330
R	
raw _␣ (mc-key)	73, 257, 458
ref (key)	1, 526, 664
\RemoveFromHook	34, 550, 551
\renewcommand	631, 632
\RenewDocumentCommand	8
\RequirePackage	20, 54, 316, 319, 325, 328, 590
\rlap	465
role _␣ (rolemap-key)	163, 692

role-missing	21, 86	\setbox	396
role-namespace _□ (rolemap-key)	163, 692	Setup keys:	
role-parent-child	21, 90	activate-all (deprecated)	1
role-remapping	21, 92	activate-mc (deprecated)	1
role-tag	21, 94	activate-struct (deprecated)	1
role-unknown	21, 86	activate-tree (deprecated)	1
role-unknown-NS	21, 86	activate/all	1, 260
role-unknown-tag	21, 86	activate/mc	1, 260
role/new-attribute _□ (setup-key)	106, 1243	activate/softhyphen	1, 294
role/new-tag _□ (setup-key)	692	activate/spaces	1
root-AF (key)	1, 801	activate/struct	1, 260
		activate/struct-dest	1, 260
		activate/tagunmarked	1, 291
		activate/tree	1, 260
		debug/log	1, 278
		debug/show	277
		debug/uncompress	278
		log (deprecated)	278
		no-struct-dest (deprecated)	1
		page/tabsorder	1, 296
		root-AF	1, 801
		tabsorder (deprecated)	1, 296
		tagunmarked (deprecated)	1, 291
		uncompress (deprecated)	278
		shipout commands:	
		\g_shipout_readonly_int	99, 172, 234, 388
		show-kids	21, 64
		show-spaces _□ (deprecated)	185, 6
		show-struct	21, 64
		\ShowTagging	18, 38, 125
		skip commands:	
		\skip_horizontal:n	77
		\c_zero_skip	77
		socket commands:	
		\socket_assign_plug:nn	525, 543, 544, 560, 750, 751
		\socket_new:nn	471, 472, 503
		\socket_new_plug:nnn	474, 493, 504, 513, 526
		\socket_use:n	76, 545, 547, 554, 558
		\socket_use:nn	81, 557
		\socket_use:nnn	86
		\socket_use:nw	97
		\socket_use_expandable:n	92
		\socket_use_expandable:nw	66, 112
		stash (key)	1, 526
		stash _□ (mc-key)	74, 121
		str commands:	
		\str_case:nnTF	60, 888
		\str_const:Nn	59
		\str_if_empty:nTF	653
		\str_if_eq:nnTF	127, 374, 460, 569
		\str_if_eq_p:nn	312, 365, 367

S

\selectfont	6
seq commands:	
\seq_clear:N	299, 322
\seq_const_from_clist:Nn	21, 34
\seq_count:N	22, 25, 57, 311, 418, 1281, 1283, 1285, 1309, 1335
\seq_get:NN	674
\seq_get:NNTF	443, 458, 869, 1014, 1021
\seq_gpop:NN	1007
\seq_gpop:NNTF	105, 1008
\seq_gpop_left:NN	287
\seq_gpush:Nn	13, 15, 88, 95, 876, 916
\seq_gput_left:Nn	43, 185, 253, 291
\seq_gput_right:Nn	38, 146, 152, 184, 216, 237, 276, 342
\seq_gset_eq:NN	159, 221, 306
\seq_if_empty:NNTF	200, 412
\seq_item:Nn	58, 116, 118, 125, 129, 136, 140, 186, 328, 335, 348, 365, 367, 374, 508, 509, 517, 518, 715, 716
\seq_log:N	175, 199, 251, 271, 429, 444
\seq_map_function:NN	276
\seq_map_indexed_inline:Nn	414, 427
\seq_map_inline:Nn	282, 300, 1271, 1311
\seq_new:N	12, 14, 14, 15, 16, 17, 18, 19, 21, 22, 24, 117, 118, 136, 182, 841, 1242
\seq_pop_left:NN	423, 425, 426
\seq_put_right:Nn	301
\seq_remove_all:Nn	304
\seq_set_eq:NN	207, 208
\seq_set_from_clist:NN	1266, 1302
\seq_set_from_clist:Nn	87, 90, 196, 216, 411, 422
\seq_set_map_e:NNn	1267, 1303
\seq_set_split:Nnn	50, 146, 506, 515, 714
\seq_show:N	61, 188, 218, 219, 252, 302, 303, 305, 352, 919, 972, 993, 1003
\seq_use:Nn	50, 110, 111, 205, 235, 236, 363, 1282

\tag_struct_end:	103, 26, 53, 534, 538,	703, 759, 782, 825, 826, 999, 1000, 1038	__tag_backend_create_bdc_node ..	429
\tag_struct_end:n	103, 827, 1035		__tag_backend_create_bmc_node ..	400
\tag_struct_gput:nnn	104,		__tag_backend_create_emc_node ..	371
\tag_struct_gput_ref:nnn	104		_tag_check_add_tag_role:nn ...	129, 209, 209
\tag_struct_insert_annot:nn	103, 136, 757, 780, 1219, 1219, 1228		_tag_check_add_tag_role:nnn ..	171, 228
\tag_struct_object_ref:n	103, 636, 649, 660, 1161, 1162, 1166		_tag_check_benchmark_tic: .	348, 352, 356, 360, 364, 368, 372, 489, 495
\tag_struct_parent_int:	103, 136, 750, 757, 773, 780, 1219, 1229		_tag_check_benchmark_toc: .	350, 354, 358, 362, 366, 370, 374, 490, 496
\tag_struct_use:n	103, 104, 58, 1041, 1041, 1043		_tag_check_if_active_mc:	151
\tag_struct_use_num:n	103, 1098, 1098, 1100		_tag_check_if_active_mc:TF ...	84, 103, 150, 175, 191, 237, 362, 368, 435, 441
\tag_suspend:n	7, 67, 199, 224, 234, 257, 405, 668, 696		_tag_check_if_active_struct: .	161
\tag_tool:n	37, 13, 13, 14, 16, 20		_tag_check_if_active_struct:TF	39, 150, 832, 833, 1004, 1005, 1037, 1045, 1102, 1222
tag internal commands:			_tag_check_if_mc_in_galley: ..	355
__tag_activate_mark_space	550		_tag_check_if_mc_in_galley:TF	211, 232
\g_tag_active_mc_bool	40, 122, 126, 153, 263, 270		_tag_check_if_mc_tmb_missing:	361
\l_tag_active_mc_bool	125, 132, 153, 206, 216, 230, 241		_tag_check_if_mc_tmb_missing:TF	112, 220, 237, 361
\l_tag_active_socket_bool	75, 80, 85, 91, 96, 111, 132, 207, 217, 231, 242, 295		_tag_check_if_mc_tmb_missing_-p:	361
\g_tag_active_space_bool	13, 56, 61, 126		_tag_check_if_mc_tme_missing:	372
\g_tag_active_struct_bool	121, 126, 163, 265, 272, 309, 440		_tag_check_if_mc_tme_missing:TF	155, 224, 241, 372
\l_tag_active_struct_bool	124, 132, 163, 205, 215, 229, 240		_tag_check_if_mc_tme_missing_-p:	372
\g_tag_active_struct_dest_bool	126, 269, 276, 308		_tag_check_info_closing_struct:n	186, 186, 194, 1010
\g_tag_active_tree_bool	9, 67, 123, 126, 264, 271, 343, 381		_tag_check_init_mc_used:	285, 285, 288, 294
_tag_add_missing_mcs:Nn	86, 167, 167, 219		_tag_check_mc_if_nested:	178, 247, 247, 373
_tag_add_missing_mcs_to_-stream:Nn	65, 65, 66, 189, 189, 225		_tag_check_mc_if_open:	239, 247, 255, 445
\g_tag_attr_class_used_prop	284, 286, 1237, 1277		_tag_check_mc_in_galley:TF ...	355
\g_tag_attr_class_used_seq	282, 1242		_tag_check_mc_in_galley_p: ...	355
\g_tag_attr_entries_prop	293, 1237, 1245, 1273, 1313, 1318, 1322		_tag_check_mc_pushed_popped:nn	89, 96, 109, 112, 117, 262, 262
_tag_attr_new_entry:nn	682, 1243, 1243, 1249, 1254, 1258		_tag_check_mc_tag:N	191, 274, 274, 385
\g_tag_attr_objref_prop	1237, 1317, 1324, 1329		_tag_check_mc_used:n	145, 290, 290, 329
\l_tag_attr_value_tl	1307, 1326, 1331, 1333, 1337, 1341		\g_tag_check_mc_used_intarray .	285, 295, 297, 300
			_tag_check_no_open_struct: ...	195, 195, 1012, 1019
			_tag_check_para_begin_show:nn	448, 489, 521

__tag_check_para_end_show:nn ..	459, 533	__tag_get_data_struct_id: 485 , 485
__tag_check_parent_child:nnN ..	559, 565, 667	__tag_get_data_struct_num: 490 , 491
__tag_check_parent_child:nnnnN .	513	__tag_get_data_struct_tag: 477 , 477
__tag_check_parent_child:nnnnN	208, 401, 515,	__tag_get_mathsubtype
561, 574, 589, 668, 679, 927, 1073, 1136	 14 , 15 , 19 , 20 ,
__tag_check_show_MCID_by_page:	309, 309	102, 137, 167, 178, 187, 229, 251,
__tag_check_struct_used:n	199, 199, 1050	259, 265, 273, 278, 291, 312, 326, 336
__tag_check_structure_has_tag:n	171, 171, 866	__tag_get_mc_cnt_type_tag
__tag_check_structure_tag:N	179, 179, 510, 523 289
__tag_check_timeout_v:n 103, 103,	110, 111, 114, 149, 157, 164, 202,	__tag_get_num_from
211, 283, 499, 515, 531, 604, 609, 616	 314
__tag_debug_mc_begin_ignore:n .	390, 428	\l__tag_get_parent_tmpa_tl
__tag_debug_mc_begin_insert:n .	370, 383 113 , 206 , 209 , 222 ,
__tag_debug_mc_end_ignore: 404, 453		399, 402, 415, 677, 680, 925, 928, 943
__tag_debug_mc_end_insert: 397, 443		\l__tag_get_parent_tmpa_tl _{uuuu} \l__
__tag_debug_struct_begin_		tag_get_parent_tmpb_tl _{uuuu} \l__
ignore:n	432, 997	tag_tmpa_str
__tag_debug_struct_begin_	 109
insert:n	424, 994	\l__tag_get_parent_tmpb_tl
__tag_debug_struct_end_check:n	454, 1037 114 , 207 , 210 , 222 ,
__tag_debug_struct_end_ignore:	447, 1032	400, 403, 415, 678, 681, 926, 929, 943
__tag_debug_struct_end_insert:	439, 1030	__tag_get_tag_from
\g__tag_delayed_shipout_bool	42, 47, 51, 239 333
__tag_exclude_headfoot_begin: .	656, 717, 718	\l__tag_get_tmpe_tl
__tag_exclude_headfoot_end:	670, 719, 720 109 ,
__tag_exclude_struct_headfoot_		168, 173, 184, 186, 187, 238, 240,
begin:n	683, 724, 725	241, 898, 904, 1187, 1189, 1193, 1199
__tag_exclude_struct_headfoot_		__tag_gincr_para_begin_int: ...
end:	699, 726, 727 343 , 347 , 365 , 381 , 394 , 487 , 514
__tag_fakespace	484	__tag_gincr_para_end_int:
__tag_fakespace:	71, 73, 320 343 , 355 , 373 , 383 , 530
__tag_finish_structure:	13, 16, 340 , 341	__tag_gincr_para_main_begin_
__tag_get_data_mc_counter:	9, 9	int: .. 343 , 343 , 361 , 380 , 480 , 505
__tag_get_data_mc_tag:	256, 256, 354 , 354	__tag_gincr_para_main_end_int:
__tag_get_data_struct_counter:	496, 497 343 , 351 , 369 , 382 , 537
		__tag_hook_kernel_after_foot: .
	 642 , 651 , 720 , 727 , 734
		__tag_hook_kernel_after_head: .
	 640 , 649 , 719 , 726 , 733
		__tag_hook_kernel_before_foot:
	 641 , 650 , 718 , 725 , 732
		__tag_hook_kernel_before_head:
	 639 , 648 , 717 , 724 , 731
		\g__tag_in_mc_bool
	 16 , 18 , 179 , 228 , 240 ,
		374, 446, 664, 665, 679, 691, 692, 709
		__tag_insert_bdc_node
	 429
		__tag_insert_bmc_node
	 400
		__tag_insert_emc_node
	 371
		__tag_lastpagelabel:
	 89 , 90 , 108
		__tag_log
	 217
		\l__tag_loglevel_int 125 , 132 , 172 ,
		173, 188, 218, 237, 265, 268, 279,
		282, 285, 286, 287, 292, 385, 392,
		399, 406, 426, 434, 441, 449, 456, 482
		__tag_mark_spaces
	 489
		__tag_mc_artifact_begin_marks:n
	 23 , 45 , 81 , 382

\l__tag_mc_artifact_bool
. [20](#), [124](#), [180](#), [194](#), [241](#), [378](#)
\l__tag_mc_artifact_type_tl
. [19](#), [128](#), [132](#), [136](#),
[140](#), [144](#), [148](#), [152](#), [156](#), [349](#), [380](#), [382](#)
__tag_mc_bdc:nn [234](#), [237](#), [269](#), [311](#), [344](#)
__tag_mc_bdc_mcid:n . . . [123](#), [239](#), [316](#)
__tag_mc_bdc_mcid:nn
. [239](#), [242](#), [272](#), [318](#), [323](#)
__tag_mc_bdc_shipout:nn . . [238](#), [250](#)
__tag_mc_begin_marks:nn
. [23](#), [23](#), [44](#), [80](#), [389](#)
__tag_mc_bmc:n [234](#), [235](#), [340](#)
__tag_mc_bmc_artifact: [338](#), [338](#), [351](#)
__tag_mc_bmc_artifact:n [338](#), [342](#), [352](#)
\l__tag_mc_botmarks_seq
. [86](#), [21](#), [90](#), [111](#),
[161](#), [208](#), [216](#), [219](#), [221](#), [236](#), [357](#), [374](#)
__tag_mc_disable_marks: [78](#), [78](#)
__tag_mc_emc: [158](#), [234](#), [236](#), [448](#)
__tag_mc_end_marks: . . [23](#), [63](#), [82](#), [449](#)
\l__tag_mc_firstmarks_seq
. [86](#), [21](#), [87](#), [110](#), [196](#), [199](#),
[200](#), [207](#), [208](#), [218](#), [235](#), [357](#), [365](#), [367](#)
\g__tag_mc_footnote_marks_seq . . . [14](#)
__tag_mc_get_marks: . [84](#), [84](#), [210](#), [231](#)
__tag_mc_handle_artifact:N
. [119](#), [338](#), [346](#), [380](#)
__tag_mc_handle_mc_label:n
. [26](#), [26](#), [199](#), [393](#)
__tag_mc_handle_mcid:nn
. [239](#), [321](#), [326](#), [386](#)
__tag_mc_handle_stash:n [49](#), [140](#),
[142](#), [143](#), [172](#), [229](#), [327](#), [327](#), [337](#), [421](#)
__tag_mc_if_in: [68](#), [82](#), [226](#), [233](#)
__tag_mc_if_in:TF [68](#), [86](#), [226](#), [249](#), [257](#)
__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](#), [289](#)
__tag_mc_insert_mcid_single_
kids:n [131](#), [136](#), [290](#)
\l__tag_mc_key_label_tl
. [22](#), [196](#), [199](#), [321](#), [389](#), [390](#), [393](#), [494](#)
\l__tag_mc_key_properties_tl
. [22](#), [181](#), [270](#), [285](#), [286](#),
[306](#), [307](#), [388](#), [468](#), [477](#), [478](#), [490](#), [491](#)
\l__tag_mc_key_stash_bool
. [20](#), [31](#), [40](#), [123](#), [202](#), [395](#)
\g__tag_mc_key_tag_tl [19](#), [22](#),
[184](#), [244](#), [256](#), [262](#), [354](#), [376](#), [447](#), [464](#)
\l__tag_mc_key_tag_tl [22](#), [183](#), [191](#),
[193](#), [243](#), [261](#), [375](#), [385](#), [387](#), [389](#), [463](#)
__tag_mc_lua_set_mc_type_attr:n
. [83](#), [83](#), [107](#), [193](#)
__tag_mc_lua_unset_mc_type_
attr: [83](#), [109](#), [242](#)
\g__tag_mc_main_marks_seq [14](#)
\g__tag_mc_marks [13](#),
[25](#), [34](#), [47](#), [54](#), [65](#), [71](#), [88](#), [91](#), [197](#), [217](#)
\g__tag_mc_multicol_marks_seq . . . [14](#)
\g__tag_mc_parenttree_prop
. [17](#), [18](#), [103](#), [166](#), [185](#), [333](#)
\l__tag_mc_ref_abspage_tl
. [11](#), [275](#), [287](#), [295](#), [303](#)
__tag_mc_set_label_used:n [30](#), [30](#), [50](#)
\g__tag_mc_stack_seq
. [18](#), [88](#), [95](#), [105](#), [271](#)
__tag_mc_store:nnn . . [93](#), [93](#), [107](#), [134](#)
\l__tag_mc_tmpa_tl . [12](#), [289](#), [292](#), [296](#)
g__tag_MCID_abs_int [7](#)
\g__tag_MCID_byabspage_prop
. [267](#), [285](#), [294](#), [302](#)
\g__tag_MCID_tmp_bypage_int
. [268](#), [271](#), [292](#), [300](#), [313](#)
\g__tag_mode_lua_bool
. . . [41](#), [49](#), [50](#), [146](#), [241](#), [297](#), [314](#),
[323](#), [323](#), [401](#), [596](#), [659](#), [674](#), [686](#), [704](#)
__tag_new_output_prop_handler:n
. [70](#), [80](#), [104](#), [839](#)
__tag_pairs_prop [234](#)
\l__tag_para_attr_class_tl
. [324](#), [415](#), [519](#)
\g__tag_para_begin_int
. [324](#), [349](#), [367](#), [454](#), [578](#), [583](#)
\l__tag_para_bool [324](#), [421](#), [430](#), [437](#),
[443](#), [476](#), [495](#), [528](#), [631](#), [632](#), [658](#), [685](#)
\g__tag_para_end_int
. [324](#), [357](#), [375](#), [465](#), [578](#), [584](#)
\l__tag_para_flattened_bool
. . . . [324](#), [426](#), [433](#), [446](#), [478](#), [503](#), [535](#)
\l__tag_para_main_attr_class_tl
. [324](#), [509](#)
\g__tag_para_main_begin_int
. [324](#), [345](#), [363](#), [569](#), [574](#)
\g__tag_para_main_end_int
. [324](#), [353](#), [371](#), [569](#), [575](#)
__tag_para_main_store_struct:
. [385](#), [385](#), [485](#), [511](#)
\g__tag_para_main_struct_tl [324](#), [387](#)
\l__tag_para_main_tag_tl
. [324](#), [425](#), [432](#), [445](#), [483](#), [508](#)
\l__tag_para_show_bool
. [324](#), [422](#), [423](#), [438](#), [451](#), [462](#)
\l__tag_para_tag_default_tl [324](#)

\l__tag_para_tag_tl
 324, 393, 424, 431, 439, 444, 488, 518
 \l__tag_parent_child_check_tl ..
 212, 213, 405, 406, 459, 683,
 684, 932, 933, 1078, 1079, 1141, 1142
 __tag_parenttree_add_objr:nn ..
 162, 162, 465
 \l__tag_parenttree_content_tl ..
 169, 188, 200, 220, 228, 249, 252
 \g__tag_parenttree_objr_tl
 161, 164, 249
 __tag_pdf_name_e:n 100, 100
 __tag_pdf_object_ref 459
 __tag_prop_gput:Nnn
 9, 29, 92, 120, 127,
 131, 180, 183, 191, 293, 301, 307, 1056
 __tag_prop_item:Nn ... 9, 53, 180, 187
 __tag_prop_new:N 9,
 9, 11, 103, 180, 180, 194, 267, 837
 __tag_prop_new_linked:N
 15, 17, 180, 181
 __tag_prop_show:N 9, 66, 180, 189, 197
 \c__tag_property_mc_clist
 123, 249, 310
 __tag_property_record:nn
 . 28, 149, 149, 158, 245, 306, 451, 532
 __tag_property_ref_lastpage:nn
 . 82, 159, 159, 159, 173, 176, 313, 327
 \c__tag_property_struct_clist ..
 123, 534
 \l__tag_Ref_tmpa_tl 109
 g__tag_role/RoleMap_dict 18
 \g__tag_role_add_mathml_bool ...
 73, 258, 702, 766
 __tag_role_add_tag:nn
 127, 127, 155, 282, 367, 737
 __tag_role_add_tag:nnnn
 169, 169, 228, 314, 742
 __tag_role_alloctag:nnn 81,
 85, 95, 107, 117, 126, 142, 186, 279, 310
 \l__tag_role_debug_prop
 164, 11, 518, 519, 591, 592
 __tag_role_get:nnNN
 156, 158, 166, 229, 231, 255, 519, 877
 __tag_role_get_parent_child_
 rule:nnnN 178, 459, 460, 512, 544, 651
 \g__tag_role_index_prop . 164, 10,
 416, 424, 436, 437, 438, 443, 449,
 451, 452, 455, 457, 464, 465, 520, 530
 \g__tag_role_NS<ns>_class_prop 164
 \g__tag_role_NS<ns>_prop 164
 \g__tag_role_NS_mathml_prop 260, 453
 __tag_role_NS_new:nnn
 . 166, 20, 22, 30, 74, 75, 76, 77, 78, 80
 \g__tag_role_NS_prop
 164, 9, 26, 56, 168, 318, 336, 725
 \g__tag_role_parent_child_
 intarray 392, 395, 473
 __tag_role_read_namespace:n 339,
 339, 343, 344, 345, 347, 349, 351, 352
 __tag_role_read_namespace:nn ..
 320, 320, 341, 350
 __tag_role_read_namespace_
 line:nw 257, 261, 294, 330
 __tag_role_remap:
 690, 690, 691, 951, 1083, 1146
 __tag_role_remap_id: 691, 691
 \l__tag_role_remap_NS_tl
 688, 950, 953, 1082, 1085, 1145, 1148
 \l__tag_role_remap_tag_tl
 688, 949, 952, 1081, 1084, 1144, 1147
 \l__tag_role_role_namespace_
 tmpa_tl 12,
 697, 718, 723, 725, 727, 731, 746
 \l__tag_role_role_tmpa_tl
 12, 696, 716, 722, 739, 745
 \g__tag_role_rolemap_prop
 164, 18, 145, 148, 151, 160,
 216, 219, 222, 262, 265, 387, 525, 535
 \c__tag_role_rules_num_prop 393, 484
 \c__tag_role_rules_prop 393, 396, 477
 \l__tag_role_tag_namespace_tmpa_
 tl 12, 567, 571, 575, 695, 744
 \l__tag_role_tag_namespace_tmpb_
 tl 14, 568, 569, 572, 576
 \l__tag_role_tag_namespace_tmpb_
 tl_uuuuuu% 12
 \l__tag_role_tag_tmpa_tl
 12, 694, 715, 738, 743
 \g__tag_role_tags_class_prop ...
 164, 8, 90, 99, 112, 121, 137, 270
 \g__tag_role_tags_NS_prop .. 164,
 7, 88, 97, 110, 119, 130, 181, 216,
 280, 385, 507, 516, 567, 568, 721, 1025
 \l__tag_role_tmpa_seq 12
 \l__tag_role_update_bool
 210, 257, 258, 266, 346, 348
 \c__tag_role_userNS_id_str
 165, 59, 80
 \g__tag_root_default_tl 287
 \g__tag_saved_in_mc_bool
 655, 664, 679, 691, 709
 __tag_seq_gput_left:Nn
 9, 41, 185, 193, 248
 __tag_seq_gput_right:Nn 9,
 36, 180, 184, 192, 211, 221, 232, 271
 __tag_seq_item:Nn ... 9, 48, 180, 186

__tag_seq_new:N	__tag_struct_insert_annot:nn	..
....	9, 9, 22, 105, 180, 182, 195, 840	436, 436, 1224
__tag_seq_show:N	9, 59, 180, 188, 196	__tag_struct_kid_mc_gput_-	
__tag_show_spacemark	right:nn	... 195, 207, 208, 227, 330
\l_tag_showspaces_bool	... 7, 16, 17	__tag_struct_kid_OBJR_gput_-	
\g_tag_softyphen_bool	... 138, 294	right:nnn	.. 260, 260, 263, 284, 452
__tag_space_chars_shipout	__tag_struct_kid_struct_gput_-	
__tag_start_para_ints:	left:nn 244, 244, 245, 259
.....	218, 243, 359, 359	__tag_struct_kid_struct_gput_-	
__tag_stop_para_ints:	right:nn
.....	208, 232, 359, 378	228, 228, 229, 243, 1052, 1115
__tag_store_parent_child_-		g__tag_struct_kids_1_seq 102
rule:nnn 393, 393, 430	\g__tag_struct_label_num_prop	..
g__tag_struct_1_prop 102	62, 530, 632
__tag_struct_add_AF:nn	\l__tag_struct_lang_tl
.....	700, 717, 737, 744, 764, 807	637, 823, 848, 853
__tag_struct_add_inline_AF:nn	..	__tag_struct_mcid_dict:n
.....	689, 716, 778, 782, 789, 797	98, 101, 195, 214
\l__tag_struct_addkid_tl	64, 554, 968	\c__tag_struct_null_tl 10, 329
\g__tag_struct_AFobj_int	687, 695, 698	\g__tag_struct_objR_seq 8
\g__tag_struct_cont_mc_prop	...	__tag_struct_output_prop_aux:nn
.....	11, 95, 96, 98, 101, 224	70, 70, 84
\g__tag_struct_dest_num_prop	66, 645	__tag_struct_prop_gput:nnn
\l__tag_struct_elem_stash_bool	..	88, 89, 90, 96, 107, 112, 117, 122,	
.....	63, 536, 921, 981	129, 155, 164, 170, 331, 344, 358,	
__tag_struct_exchange_kid_-		566, 578, 592, 608, 616, 681, 703,	
command:N 285, 285, 294, 325	746, 765, 808, 844, 850, 855, 882,	
__tag_struct_fill_kid_key:n	...	900, 909, 959, 1119, 1196, 1287, 1338	
.....	135, 295, 295, 421	\g__tag_struct_ref_by_dest_prop
__tag_struct_format_parentrole:nnN	69, 82
.....	389, 390	__tag_struct_Ref_dest:nN	.. 622, 643
__tag_struct_format_Ref 123	__tag_struct_Ref_label:nN	622, 630
__tag_struct_format_Ref:nnN	391, 391	__tag_struct_Ref_num:nN	.. 622, 656
__tag_struct_format_rolemap:nnN	__tag_struct_Ref_obj:nN	.. 622, 622
.....	389, 389	\g__tag_struct_roletag_NS_tl 58
__tag_struct_get_dict_content:nN	\l__tag_struct_roletag_NS_tl	...
.....	137, 375, 375, 422	61, 881, 886, 913
__tag_struct_get_id:n	\l__tag_struct_roletag_tl
..	95, 100, 113, 114, 139, 140, 428, 487	58, 880, 886, 888, 913, 917
__tag_struct_get_parentrole:nnN	__tag_struct_set_tag_info:nnn	..
.....	178,	150, 152, 162, 177, 862, 954, 1086, 1149	
178, 194, 204, 397, 675, 923, 1069, 1132		\g__tag_struct_stack_current_tl	..
__tag_struct_gput_data_ref:nn	..	16, 29, 38, 69, 75, 99, 148, 154,	
.....	1201, 1218	162, 168, 205, 216, 226, 312, 331,	
__tag_struct_gput_data_ref_-		335, 398, 409, 418, 482, 487, 493,	
aux:nnn	918, 966, 970, 971, 992, 1010, 1016,	
..	1180, 1181, 1203, 1207, 1211, 1215	1053, 1060, 1066, 1116, 1123, 1129	
__tag_struct_gput_data_ref_-		\l__tag_struct_stack_parent_-	
dest:nn 1209	tmpa_tl 16, 445, 454, 471,
__tag_struct_gput_data_ref_-		546, 860, 867, 871, 896, 924, 936,	
label:nn 1205	946, 963, 967, 969, 972, 984, 985, 993	
__tag_struct_gput_data_ref_-		\g__tag_struct_stack_seq
num:nn 1213	12, 22, 25, 444,
		674, 870, 876, 919, 1003, 1008, 1014	

\c__tag_struct_StructElem_-	329, 411, 418, 422, 423, 424, 425,
entries_seq	426, 427, 436, 437, 438, 443, 449,
\c__tag_struct_StructTreeRoot_-	451, 455, 458, 462, 464, 466, 467,
entries_seq	474, 484, 486, 495, 520, 520, 521,
\g__tag_struct_tag_NS_tl	522, 525, 527, 541, 545, 595, 603,
.....	605, 606, 608, 612, 617, 648, 652,
520, 522, 865, 879, 931, 944, 950,	674, 676, 699, 702, 711, 712, 714,
953, 957, 991, 1027, 1075, 1082,	938, 939, 946, 1007, 1008, 1014,
1085, 1089, 1138, 1145, 1148, 1152	1016, 1021, 1024, 1025, 1027, 1071,
\g__tag_struct_tag_stack_seq ...	1076, 1110, 1134, 1139, 1279, 1290
.....	\l__tag_tmpb_box
14, 50,
251, 252, 429, 444, 458, 916, 1007, 1021	109, 172, 179, 180, 184, 186
\g__tag_struct_tag_tl	\l__tag_tmpb_seq
.....
58, 183, 184, 187, 375, 376,	109, 1266, 1267, 1302, 1303
508, 510, 517, 520, 521, 523, 864,	\l__tag_tmpb_tl
878, 917, 930, 944, 949, 952, 956,
1023, 1025, 1067, 1074, 1081, 1084,	176, 88, 103, 109, 117,
1088, 1130, 1137, 1144, 1147, 1151	119, 407, 424, 430, 452, 457, 465,
__tag_struct_write_obj	468, 474, 488, 520, 522, 530, 532,
.....	535, 537, 542, 545, 622, 630, 632,
123	632, 633, 635, 636, 639, 644, 645,
__tag_struct_write_obj:n	649, 649, 652, 1072, 1077, 1135, 1140
.....	__tag_tree_fill_parenttree: ...
150, 403, 403
\l__tag_tag_stop_int 199, 203, 204,	170, 171, 246
212, 213, 220, 227, 228, 237, 238, 246	__tag_tree_final_checks: 20, 20, 346
\g__tag_tagunmarked_bool 137, 291, 293	\g__tag_tree_id_pad_int .. 77, 81, 145
\l__tag_tmpa_box	__tag_tree_lua_fill_parenttree:
.....
109, 171, 177, 178, 182, 193, 194	226, 226, 243
\l__tag_tmpa_clist	\g__tag_tree_openaction_struct_-
.....	tl
109, 1265, 1266, 1299, 1300, 1302	31, 37, 56
\l__tag_tmpa_int	__tag_tree_parenttree_rerun_-
89, 92, 97,	msg:
100, 104, 109, 113, 398, 410, 412, 482	170, 213, 248
\l__tag_tmpa_prop	__tag_tree_update_openaction: .
.....
109, 175, 183, 196, 198	41, 74
\l__tag_tmpa_seq	__tag_tree_write_classmap:
.....
50, 57, 58, 109, 299, 301,	279, 279, 361
303, 304, 305, 306, 322, 342, 352,	__tag_tree_write_idtree: .. 85, 353
411, 414, 422, 423, 425, 426, 427,	__tag_tree_write_namespaces: ..
506, 508, 509, 515, 517, 518, 714,
715, 716, 1267, 1271, 1281, 1282,	314, 314, 365
1283, 1285, 1303, 1309, 1311, 1335	__tag_tree_write_parenttree: ..
\l__tag_tmpa_str 42, 43, 48, 115, 281,
286, 291, 302, 307, 314, 473, 478,	239, 239, 349
486, 491, 562, 569, 574, 581, 588,	__tag_tree_write_rolemap:
595, 604, 611, 677, 684, 756, 757, 762
\l__tag_tmpa_tl	256, 256, 357
.....	__tag_tree_write_structelements:
41, 42, 46, 48, 49, 50, 55, 86, 87,
93, 93, 96, 98, 101, 105, 105, 107,	146, 146, 369
108, 109, 112, 113, 115, 118, 119,	__tag_tree_write_structtreeroot:
137, 137, 138, 140, 142, 142, 145,
146, 151, 179, 180, 182, 185, 186,	125, 125, 373
198, 199, 200, 201, 202, 204, 204,	__tag_whatsits:
207, 216, 216, 217, 218, 222, 226,
235, 270, 271, 273, 277, 279, 281,	35, 57, 62, 63, 66, 356, 357
287, 288, 288, 291, 300, 301, 302,	tag-namespace_(rolemap-key)
304, 306, 308, 310, 311, 319, 327,	692
	tag/struct/1 internal commands:
	__tag/struct/1
	30
	tag/tree/namespaces internal commands:
	__tag/tree/namespaces
	313

tag/tree/parenttree internal commands:	
__tag/tree/parenttree	153
tag/tree/rolemap internal commands:	
__tag/tree/rolemap	255
tagabspage	8, 163
tagmcabs	8, 163
\tagmcbegin	37, 164, 22, 402, 408
\tagmccend	37, 22, 408
tagmccid	8, 163
\tagmccifinTF	37, 39
\tagmccuse	37, 22
\tagpdfparaOff	39, 628
\tagpdfparaOn	39, 628
\tagpdfsetup	37, 106, 163, 6
\tagpdfsuppressmarks	39, 633
\tagstart	7, 223, 254
\tagstop	7, 222, 253
tagstruct	8, 163
\tagstructbegin	38, 135, 163, 164, 45, 290, 393, 395
\tagstructend	38, 45, 291, 408
tagstructobj	8, 163
\tagstructuse	38, 45
\tagtool	37, 13
tagunmarked (deprecated) (key)	1, 291
test/lang _U (setup-key)	635
\TeX and \LaTeX 2 ϵ commands:	
\@M	168
\@auxout	94
\@bsphack	151
\@ccclv	605
\@esphack	153
\@gobble	31, 55
\@ifpackageloaded	28, 588
\@kernel@after@foot	651
\@kernel@after@head	649
\@kernel@before@ccclv	595, 602
\@kernel@before@foot	650
\@kernel@before@footins	598, 600
\@kernel@before@head	646, 648
\@kernel@tag@hangfrom	391
\@kernel@tagsupport@makecol	594, 607
\@makecol	604, 609
\@maxdepth	181
\@mult@ptagging@hook	612, 614
\@outputbox	610
\@secondoftwo	31, 55
\@tempboxa	396, 406, 408
\c@chapter	362, 380
\c@page	604, 609
\count@	619
\mult@firstbox	617
\mult@rightbox	621
\new@label@record	96
\on@line	500, 515, 531
\page@sofar	616
\process@cols	617
tex commands:	
\tex_botmarks:D	91
\tex_firstmarks:D	88
\tex_kern:D	184
\tex_marks:D	25, 34, 47, 54, 65, 71
\tex_special:D	66
\tex_splitbotmarks:D	217
\tex_splitfirstmarks:D	197
texsource (key)	1, 687
\the	604, 609
\tiny	454, 465
title (key)	1, 526
title-o (key)	1, 526
tl commands:	
\c_empty_tl	367, 387
\c_space_tl	54, 55, 58, 60, 76, 103, 166, 190, 191, 193, 198, 200, 202, 209, 252, 290, 368, 385, 400, 427, 604, 609, 626, 636, 649, 660, 727, 947, 984, 1066, 1129, 1282, 1328
\tl_clear:N	87, 88, 105, 181, 190, 191, 281, 377, 569
\tl_const:Nn	10
\tl_count:n	78, 82, 145
\tl_gput_right:Nn	164, 725
\tl_gset:Nn	18, 32, 37, 99, 244, 262, 288, 300, 333, 387, 447, 464, 508, 509, 517, 518, 521, 522, 732, 918, 1016, 1023, 1027
\tl_gset_eq:NN	184, 376
\tl_head:N	605, 632
\tl_if_empty:NTF	42, 43, 108, 196, 276, 299, 350, 390, 606, 633, 712, 718, 757, 848
\tl_if_empty:nTF	51, 69, 77, 89, 143, 198, 211, 212, 230, 264, 268, 278, 297, 299, 299, 408, 477, 483, 485, 585, 593, 601, 620, 692, 762
\tl_if_empty_p:n	312
\tl_if_eq:NNTF	329, 357
\tl_if_eq:NnTF	107
\tl_if_eq:nnTF	214, 267, 280
\tl_if_exist:NTF	138, 340, 413, 612, 720
\tl_if_head_eq_charcode:nNTF	48
\tl_if_in:nnTF	187
\tl_new:N	11, 12, 12, 13, 14, 15, 16, 17, 19, 20, 22, 23, 24, 25, 31, 32, 58, 59, 60, 61, 64, 109, 110, 111, 112, 113, 114, 161, 169, 287, 332, 334, 336, 338, 341, 342, 459, 688, 689, 730, 823, 1240

