

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

Ulrike Fischer[†]

Released 2025-05-16

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	9
4.2 a LastPage label	10
5 Variables	10
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 v0.99q, last revised 2025-05-16.

[†]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	29
4	Retrieving data	29
5	User conditionals	29
6	Internal checks	30
6.1	checks for active tagging	30
6.2	Checks related to structures	31
6.3	Checks related to roles	32
6.4	Check related to mc-chunks	33
6.5	Checks related to the state of MC on a page or in a split stream	36
6.6	Benchmarks	39
 III The tagpdf-user module		
Code related to L ^A T _E X2e user commands and document com-		
mands		
Part of the tagpdf package		40
1	Setup commands	40
2	Commands related to mc-chunks	40
3	Commands related to structures	41
4	Debugging	41
5	Extension commands	42
5.1	Fake space	42
5.2	Tagging of paragraphs	42
5.3	Header and footer	43
5.4	Link tagging	43
6	Socket support	43

7	User commands and extensions of document commands	44
8	Setup and preamble commands	44
9	Commands for the mc-chunks	45
10	Commands for the structure	45
11	Socket support	46
12	Debugging	47
13	Commands to extend document commands	51
	13.1 Document structure	51
	13.2 Structure destinations	51
	13.3 Fake space	52
	13.4 Paratagging	52
	13.5 output routine stuff	58
	13.6 Language support	59
	13.7 Header and footer	59
	13.8 Links	62
	13.9 Attaching css-files for derivation	63

IV The tagpdf-tree module

Commands trees and main dictionaries

Part of the tagpdf package 66

1	Trees, pdfmanagement and finalization code	66
	1.1 Check structure	66
	1.2 Catalog: MarkInfo and StructTreeRoot and OpenAction	67
	1.3 Writing the IDtree	68
	1.4 Writing structure elements	69
	1.5 ParentTree	70
	1.6 Rolemap dictionary	73
	1.7 Classmap dictionary	74
	1.8 Namespaces	74
	1.9 Finishing the structure	75
	1.10 StructParents entry for Page	76

V The tagpdf-mc-shared module

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

Part of the tagpdf package 77

1	Public Commands	77
2	Public keys	78

3	Marked content code – shared	79
3.1	Variables and counters	79
3.2	Functions	80
3.3	Keys	84
VI	The tagpdf-mc-generic module	
	Code related to Marked Content (mc-chunks), generic mode	
	Part of the tagpdf package	86
1	Marked content code – generic mode	86
1.1	Variables	86
1.2	Functions	87
1.3	Looking at MC marks in boxes	90
1.4	Keys	96
VII	The tagpdf-mc-luacode module	
	Code related to Marked Content (mc-chunks), luamode-specific	
	Part of the tagpdf package	98
1	Marked content code – luamode code	98
1.1	Commands	100
1.2	Key definitions	104
VIII	The tagpdf-struct module	
	Commands to create the structure	
	Part of the tagpdf package	106
1	Public Commands	106
2	Public keys	107
2.1	Keys for the structure commands	107
2.2	Setup keys	109
3	Variables	109
3.1	Variables used by the keys	111
3.2	Variables used by tagging code of basic elements	112
4	Commands	112
4.1	Initialization of the StructTreeRoot	113
4.2	Adding the /ID key	114
4.3	Filling in the tag info	115
4.4	Handlings kids	116
4.5	Output of the object	120
4.6	Commands for the parent-child checks	124
5	Keys	127
6	User commands	135

7	Attributes and attribute classes	144
7.1	Variables	144
7.2	Commands and keys	145
IX	The tagpdf-luatex.def	
	Driver for luatex	
	Part of the tagpdf package	148
1	Loading the lua	148
2	User commands to access data	152
3	Logging functions	153
4	Helper functions	155
4.1	Retrieve data functions	155
4.2	Functions to insert the pdf literals	157
5	Function for the real space chars	159
6	Function for the tagging	163
7	Parenttree	168
8	parent-child rules	170
X	The tagpdf-roles module	
	Tags, roles and namespace code	
	Part of the tagpdf package	173
1	Code related to roles and structure names	173
1.1	Variables	174
1.2	Namespaces	176
1.3	Adding a new tag	177
1.3.1	pdf 1.7 and earlier	178
1.3.2	The pdf 2.0 version	180
1.4	Helper command to read the data from files	182
1.5	Reading the default data	184
1.6	Parent-child rules	185
1.6.1	Reading in the csv-files	186
1.6.2	Retrieving the parent-child rule	188
1.7	Key-val user interface	193
XI	The tagpdf-space module	
	Code related to real space chars	
	Part of the tagpdf package	196
1	Code for interword spaces	196

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>	

<code>activate/spaces</code> (<i>setup key</i>)	<code>activate/spaces</code> activates the additional parsing needed for interword spaces. It replaces the deprecated key <code>interwordspace</code> .
<code>activate/mc</code> (<i>setup key</i>)	A key to to activate the marked content code. It should be used only in special cases,
<code>activate-mc</code> (<i>deprecated</i>) (<i>setup key</i>)	e.g. for debugging.
<code>activate/tree</code> (<i>setup key</i>)	This key activates the code that finalize the various trees. It should be used only in
<code>activate-tree</code> (<i>deprecated</i>) (<i>setup key</i>)	special cases, e.g. for debugging.
<code>activate/struct</code> (<i>setup key</i>)	This key activates the code for structures. It should be used only in special cases, e.g.
<code>activate-struct</code> (<i>deprecated</i>) (<i>setup key</i>)	for debugging.
<code>activate/all</code> (<i>setup key</i>)	This is a meta key for the three previous keys and is normally what should be used to
<code>activate-all</code> (<i>deprecated</i>) (<i>setup key</i>)	activate tagging.
<code>activate/struct-dest</code> (<i>setup key</i>)	The key allows to suppress the creation of structure destinations
<code>activate-struct-dest</code> (<i>deprecated</i>) (<i>setup key</i>)	
<code>debug/log</code> (<i>setup key</i>)	The <code>debug/log</code> key takes currently the values <code>none</code> , <code>v</code> , <code>vv</code> , <code>vvv</code> , <code>all</code> . More details are in <code>tagpdf-checks</code> .
<code>activate/tagunmarked</code> (<i>setup key</i>)	This key allows to set if (in luamode) unmarked text should be marked up as artifact.
<code>activate-tagunmarked</code> (<i>deprecated</i>) (<i>setup key</i>)	The initial value is true.
<code>activate/softhyphen</code> (<i>setup key</i>)	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.

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

1 Initialization and test if pdfmanagement is active.

```

1 <@@=tag>
2 <*package>
3 \ProvidesExplPackage {tagpdf} {2025-05-16} {0.99q}
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
27 <*debug>
28 \ProvidesExplPackage {tagpdf-debug} {2025-05-16} {0.99q}
29   { debug code for tagpdf }
30   \@ifpackageloaded{tagpdf}{\PackageWarning{tagpdf-debug}{tagpdf~not~loaded,~quitting}\ending
31 </debug> We map the internal module name “tag” to “tagpdf” in messages.
32 <*package>
33 \prop_gput:Nnn \g_msg_module_name_prop { tag }{ tagpdf }
34 \prop_gput:Nnn \g_msg_module_name_prop { tag / debug }{tagpdf~DEBUG}
35 </debug>

```

Debug mode has its special mapping:

```

32 <*debug>
33 \prop_gput:Nnn \g_msg_module_type_prop { tag / debug }{ }
34 \prop_gput:Nnn \g_msg_module_name_prop { tag / debug }{tagpdf~DEBUG}
35 </debug>

```


2 base package

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

```
36 <*base>
37 \ProvidesExplPackage {tagpdf-base} {2025-05-16} {0.99q}
38 {part of tagpdf - provide base, no-op versions of the user commands }
39 </base>
```

3 Package options

The boolean is kept for now for compability, can go in 2026.

```
40 <*package>
41 \bool_new:N\g__tag_mode_lua_bool
42 \sys_if_engine luatex:T {\bool_gset_true:N \g__tag_mode_lua_bool}
43 \DeclareOption {luamode} { }
44 \DeclareOption {genericmode}{ }
45 \ProcessOptions
```

4 Packages

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

```
46 \RequirePackage{tagpdf-base}
47 </package>
```

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

```
48 <*base>
49 \cs_new_protected:Npn \__tag_whatsits: {}
50 \AddToHook{begindocument}
51 {
52   \str_case:onF { \c_sys_backend_str }
53   {
54     { luatex } { \cs_set_protected:Npn \__tag_whatsits: {} }
55     { dvisvgm } { \cs_set_protected:Npn \__tag_whatsits: {} }
56   }
57   {
58     \cs_set_protected:Npn \__tag_whatsits: {\tex_special:D {} }
59   }
60 }
61 </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.

```
62 <*package>
63 \AddToHook{package/latex-lab-testphase-new-or-2/after}
64 {
65   \cs_set_protected:Npn \__fnote_gput_ref:nn #1 #2 %#1 the structure number receiving the r
66   {
```

```

67     \tag_struct_gput:nnn {#1}{ref_num}{#2}
68   }
69 }
70 \AddToHook{package/latex-lab-testphase-toc/after}
71 {
72   \cs_set_protected:Npn \g__tag_struct_ref_by_dest:
73   {
74     \prop_map_inline:Nn\g__tag_struct_ref_by_dest_prop
75     {
76       \tag_struct_gput:nnn {##1}{ref_dest}{##2}
77     }
78   }
79 }
80 \</package>

```

4.2 a LastPage label

See also issue #2 in Accessible-xref

__tag_lastpagelabel:

```

81 \<package>
82 \cs_new_protected:Npn \__tag_lastpagelabel:
83 {
84   \legacy_if:nT { @files }
85   {
86     \exp_args:NNne \exp_args:NNe\iow_now:Nn \@auxout
87     {
88       \token_to_str:N \new@label@record
89       { @tag@LastPage }
90       {
91         { abspage } { \int_use:N \g_shipout_readonly_int }
92         { tagmcabs } { \int_use:N \c@g__tag_MCID_abs_int }
93         { tagstruct } { \int_use:N \c@g__tag_struct_abs_int }
94       }
95     }
96   }
97 }
98
99 \AddToHook{enddocument/afterlastpage}
100 { \__tag_lastpagelabel: }

```

(End of definition for __tag_lastpagelabel:.)

5 Variables

A few temporary variables

\l__tag_tmpa_tl	101 \tl_new:N	\l__tag_tmpa_tl
\l__tag_tmpp_tl	102 \tl_new:N	\l__tag_tmpp_tl
\l__tag_tmppc_tl	103 \tl_new:N	\l__tag_tmppc_tl
\l__tag_tmp_unused_tl	104 \tl_new:N	\l__tag_tmp_unused_tl
\l__tag_get_tmppc_tl	105 \tl_new:N	\l__tag_Ref_tmppa_tl
\l__tag_get_parent_tmppa_tl	106 \tl_new:N	\l__tag_get_tmppc_tl
\l__tag_get_parent_tmppb_tl	107 \tl_new:N	\l__tag_get_parent_tmppa_tl
\l__tag_get_parent_tmppc_tl		
\l__tag_get_child_tmppa_tl		
\l__tag_get_child_tmppb_tl		
\l__tag_get_child_tmppc_tl		
\l__tag_tmppa_str		
\l__tag_tmppa_prop		
\l__tag_tmppa_seq		
\l__tag_tmppb_seq		
\l__tag_tmppa_clist		
\l__tag_tmppa_int		
\l__tag_tmppa_box		

```

108 \tl_new:N \l__tag_get_parent_tmpb_tl
109 \tl_new:N \l__tag_get_parent_tmpc_tl
110 \tl_new:N \l__tag_get_child_tmpa_tl
111 \tl_new:N \l__tag_get_child_tmpb_tl
112 \tl_new:N \l__tag_get_child_tmpc_tl
113 \str_new:N \l__tag_tmpa_str
114 \prop_new:N \l__tag_tmpa_prop
115 \seq_new:N \l__tag_tmpa_seq
116 \seq_new:N \l__tag_tmpb_seq
117 \clist_new:N \l__tag_tmpa_clist
118 \int_new:N \l__tag_tmpa_int
119 \box_new:N \l__tag_tmpa_box
120 \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
121 \clist_const:Nn \c__tag_property_mc_clist {tagabspace,tagmcabs,tagmcid}
122 \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.

```

123 \int_new:N \l__tag_loglevel_int

```

(End of definition for `\l__tag_loglevel_int`.)

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

```

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

```

130 \bool_new:N \l__tag_active_mc_bool
131 \bool_set_true:N \l__tag_active_mc_bool

```

```

132 \bool_new:N \l__tag_active_struct_bool
133 \bool_set_true:N \l__tag_active_struct_bool
134 \bool_new:N \l__tag_active_socket_bool

(End of definition for \l__tag_active_mc_bool, \l__tag_active_struct_bool, and \l__tag_active_socket_bool.)

```

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

```

135 \bool_new:N \g__tag_tagunmarked_bool

(End of definition for \g__tag_tagunmarked_bool.)

```

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

```

136 \bool_new:N \g__tag_softthyphen_bool

(End of definition for \g__tag_softthyphen_bool.)

```

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

```

137 \int_new:N \g__tag_unique_cnt_int

(End of definition for \g__tag_unique_cnt_int.)

```

6 Variants of l3 commands

```

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

```

7 Label and Reference commands

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

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

```

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, eo}
(End of definition for \__tag_property_record:nn.)

```

`__tag_property_ref_lastpage:nn`

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

```

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
tagstructobj
tagabspage
tagmcabs
tagmcid

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

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

```

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
    \cs_set_eq:NN \_tag_seq_show:N \seq_show:N
    \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, Nno, cnn, cen, cne, cno, c }
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 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.

\tag_resume:n

\tag_stop:

\tag_start:

\tag_stop:n

\tag_start:n

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 \begin{package} debug
200 \begin{package} \int_new:N \l__tag_tag_stop_int
\l__tag_tag_stop_int
201 \cs_set_protected:Npn \tag_stop:
202 {
203 \begin{package} \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 activate/struct (*setup key*) 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*)

log (deprecated) (*setup key*) 278 debug/log .choice:,

uncompress (deprecated) (*setup key*) 279 debug/log / none .code:n = {\int_set:Nn \l__tag_loglevel_int { 0 }},
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*) The initial value is true.

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 {<keyword>}`

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

`\tag_if_box_tagged_p:N *` `\tag_if_box_tagged:NTF <box> {<true code>} {<false code>}`

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

2 Description of log messages

2.1 \ShowTagging command

Argument	type	note
<code>\ShowTaggingmc-data = num</code>	log+term	lua-only
<code>\ShowTaggingmc-current</code>	log+term	
<code>\ShowTaggingstruck-stack= [log show]</code>	log or term+stop	
<code>\ShowTaggingdebug/structures = num</code>	log+termn	debug mode only

2.2 Messages in checks and commands

command	message	action
\@@_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

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

<u>struct-unknown</u> <u>struct-no-objnum</u> <u>struct-orphan</u> <u>struct-faulty-nesting</u> <u>struct-missing-tag</u> <u>struct-used-twice</u> <u>struct-label-unknown</u> <u>struct-show-closing</u>	Various messages related to structure. Check the definition in the code for their meaning and the arguments they take.
<u>tree-struct-still-open</u>	Message issued at the end of the compilation if there are (beside Root) other open structures on the stack.
<u>tree-statistic</u>	Message issued at the end of the compilation showing the number of objects to write
<u>show-struct</u> <u>show-kids</u>	These two messages are used in debug mode to show the current structures in the log and terminal.
<u>attr-unknown</u>	Message if an attribute is unknown.
<u>role-missing</u> <u>role-unknown</u> <u>role-unknown-tag</u> <u>role-unknown-NS</u> <u>role-tag</u> <u>new-tag</u> <u>role-parent-child-result</u> <u>role-remapping</u>	Messages related to role mapping.
<u>tree-mcid-index-wrong</u>	Used in the tree code, typically indicates the document must be rerun.
<u>sys-no-interwordspace</u>	Message if an engine doesn't support inter word spaces
<u>para-hook-count-wrong</u>	<p>Message if the number of begin paragraph and end paragraph differ. This normally means faulty structure.</p> <pre> 1 <@@=tag> 2 <*header> 3 \ProvidesExplPackage {tagpdf-checks-code} {2025-05-16} {0.99q} 4 {part of tagpdf - code related to checks, conditionals, debugging and messages} 5 </header> </pre>

3 Messages

3.1 Messages related to mc-chunks

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

```
6 <*package>
7 \msg_new:nnn { tag } {mc-nested} { nested-marked-content-found--mcid~#1 }
```

(End of definition for mc-nested. This function is documented on page 20.)

mc-tag-missing If the tag is missing

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

(End of definition for mc-tag-missing. This function is documented on page 20.)

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

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

(End of definition for mc-label-unknown. This function is documented on page 20.)

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

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

(End of definition for mc-used-twice. This function is documented on page 20.)

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

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

(End of definition for mc-not-open. This function is documented on page 20.)

mc-pushed Informational messages about mc-pushing.

mc-popped

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

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

mc-current Informational messages about current mc state.

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

`role-unknown-tag`

`role-unknown-NS`

```

86  \msg_new:nnn { tag } {role-missing}      { tag~#1-has-no~role~assigned  }
87  \msg_new:nnn { tag } {role-unknown}      { role~#1-is-not-known  }
88  \msg_new:nnn { tag } {role-unknown-tag} { tag~#1-is-not-known  }
89  \msg_new:nnn { tag } {role-unknown-NS}   { \tl_if_empty:nTF{#1}{Empty~NS}{NS~#1-is-not-known}

```

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

`role-parent-child-check`

This is an info message that inform which elements are checked, typically used to show the original tags, not the rolemapped one.

```

90  \msg_new:nnn { tag } {role-parent-child-check}
91  { Checking~Parent~Child~'#1'--->~'#2' }

```

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

`role-parent-child-result`

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

```

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

```

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

`role-struct-parent-child-forbidden`

The most important message is that the relation is not allowed between two structures. Argument #1 is the parent structure number, #2 is the child structure number, #3 NS:tag info of the parent (TODO perhaps rolemapped), #4 NS:tag of the child. (TODO)

```

94  \msg_new:nnn { tag } {role-struct-parent-child-forbidden}
95  {
96    Parent~Child~'#3'--->~'#4'.\\
97    Relation~is~not~allowed! ~\msg_line_context:\\
98    struct~#1,~
99    \exp_last_unbraced:Ne\use_i:nn { \prop_item:cn{ g__tag_struct_#1_prop}{tag} }
100    \c_space_tl-->\c_space_tl
101    struct~#2,~
102    \exp_last_unbraced:Ne\use_i:nn { \prop_item:cn{ g__tag_struct_#2_prop}{tag} }
103  }

```

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

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

```

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

```

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

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

```

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

```

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

__tag_check_forbidden_parent_child:nnnn

```

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

```

```

145     \prop_get:cnN {g__tag_struct_#3_prop}{rolemap}\l__tag_get_child_tmpc_tl
146     \msg_warning:nneeee
147     { tag }
148     {role-struct-parent-child-forbidden}
149     { #2 }
150     { #3 }
151     {
152         \exp_last_unbraced:No \use_ii:nn { \l__tag_get_parent_tmpc_tl }
153         :
154         \exp_last_unbraced:No \use_i:nn { \l__tag_get_parent_tmpc_tl }
155     }
156     {
157         \exp_last_unbraced:No \use_ii:nn { \l__tag_get_child_tmpc_tl }
158         :
159         \exp_last_unbraced:No \use_i:nn { \l__tag_get_child_tmpc_tl }
160     }
161 }
162 }
163 \cs_generate_variant:Nn\__tag_check_struct_forbidden_parent_child:nnn{onn}

```

(End of definition for __tag_check_forbidden_parent_child:nnnn.)

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

```

164 \msg_new:nnn { tag } {role-parent-child-unresolved}
165 {
166     Structure~\int_eval:n {\c@g__tag_struct_abs_int}-was~moved~into~structure~#1.\
167     Parent-Child~'#2'~-->~'#3'~can~not~checked.
168 }

```

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

__tag_check_unresolved_parent_child:nnnn

```

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

```

(End of definition for __tag_check_unresolved_parent_child:nnnn.)

tag/check/parent-child **tag/check/parent-child-end** **Socket**s used around the parent-child checks so that we can disable them.

```

183 \socket_new:nn{tag/check/parent-child}{1}

```

```

184 \socket_new:nn{tag/check/parent-child-end}{0}
185 \socket_new_plug:nnn {tag/check/parent-child-end}{check}
186 {
187   \sys_if_engine luatex:T
188   {
189     \lua_now:e
190     {
191       ltx.__tag.func.check_parent_child_rules ( 2 )
192     }
193   }
194 }

```

And a key to disable the check

```

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

```

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

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

```

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

```

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

role-tag Info messages.

new-tag

```

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

```

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

3.5 Miscellaneous

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

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

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

sys-no-interwordspace Currently only pdf_lat_ex and lua_lat_ex have some support for real spaces.

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

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

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

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

(End of definition for __tag_check_typeout_v:n.)

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

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

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

4 Retrieving data

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

```
229 \base\cs_new:Npn \tag_get:n #1 { \use:c {__tag_get_data_#1:} }
```

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

5 User conditionals

\tag_if_active_p: This tests if tagging is active. This allows packages to add conditional code. The test is true if all booleans, the global and the two local one are true.
\tag_if_active:TF

```
230 \base
231 \cs_if_exist:Nf\tag_if_active:T
232 {
233   \prg_new_conditional:Npnn \tag_if_active: { p , T , TF , F }
234   { \prg_return_false: }
235 }
236 \base
237 \package
238 \prg_set_conditional:Npnn \tag_if_active: { p , T , TF , F }
239 {
240   \bool_lazy_all:nTF
241   {
```

```

242         {\g__tag_active_struct_bool}
243         {\g__tag_active_mc_bool}
244         {\g__tag_active_tree_bool}
245         {\l__tag_active_struct_bool}
246         {\l__tag_active_mc_bool}
247     }
248     {
249         \prg_return_true:
250     }
251     {
252         \prg_return_false:
253     }
254 }
255 \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.

```

256 \begin{base}
257 \prg_new_conditional:Npnn \tag_if_box_tagged:N #1 {p,T,F,TF}
258 {
259     \tl_if_exist:cTF {l_tag_box_}\int_use:N #1_tl}
260     {
261         \int_compare:nNnTF {0\tl_use:c{l_tag_box_}\int_use:N #1_tl}}>{0}
262         { \prg_return_true: }
263         { \prg_return_false: }
264     }
265     {
266         \prg_return_false:
267         % warning??
268     }
269 }
270 \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
271 \begin{package}
272 \prg_new_conditional:Npnn \__tag_check_if_active_mc: {T,F,TF}
273 {
274     \bool_lazy_and:nnTF { \g__tag_active_mc_bool } { \l__tag_active_mc_bool }
275     {
276         \prg_return_true:

```

```

277     }
278     {
279         \prg_return_false:
280     }
281 }
282 \prg_new_conditional:Npnn \__tag_check_if_active_struct: {T,F,TF}
283 {
284     \bool_lazy_and:nnTF { \g__tag_active_struct_bool } { \l__tag_active_struct_bool }
285     {
286         \prg_return_true:
287     }
288     {
289         \prg_return_false:
290     }
291 }

```

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

6.2 Checks related to structures

__tag_check_structure_has_tag:n

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

```

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

```

(End of definition for __tag_check_structure_has_tag:n.)

__tag_check_structure_tag:N

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

```

302 \cs_new_protected:Npn \__tag_check_structure_tag:N #1
303 {
304     \prop_get:NoNF \g__tag_role_tags_NS_prop {#1}\l__tag_tmp_unused_tl
305     {
306         \msg_warning:nne { tag } {role-unknown-tag} {#1}
307     }
308 }

```

(End of definition for __tag_check_structure_tag:N.)

__tag_check_info_closing_struct:n

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

```

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

```

```

314     }
315 }
316
317 \cs_generate_variant:Nn \__tag_check_info_closing_struct:n {o,e}

```

(End of definition for __tag_check_info_closing_struct:n.)

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

```

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

```

(End of definition for __tag_check_no_open_struct:.)

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

```

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

```

(End of definition for __tag_check_struct_used:n.)

6.3 Checks related to roles

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

```

332 \cs_new_protected:Npn \__tag_check_add_tag_role:nn #1 #2 %#1 tag, #2 role
333 {
334     \tl_if_empty:nTF {#2}
335     {
336         \msg_error:nnn { tag } {role-missing} {#1}
337     }
338     {
339         \prop_get:NnNTF \g__tag_role_tags_NS_prop {#2} \l__tag_tmpa_tl
340         {
341             \int_compare:nNtT {\l__tag_loglevel_int} > { 0 }
342             {
343                 \msg_info:nnnn { tag } {role-tag} {#1} {#2}
344             }
345         }
346         {
347             \msg_error:nnn { tag } {role-unknown} {#2}
348         }
349     }
350 }

```


Similar with a namespace

```

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

```

(End of definition for __tag_check_add_tag_role:nn.)

6.4 Check related to mc-chunks

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

```

370 \cs_new_protected:Npn \__tag_check_mc_if_nested:
371 {
372   \__tag_mc_if_in:T
373   {
374     \msg_warning:nne { tag } {mc-nested} { \__tag_get_mc_abs_cnt: }
375   }
376 }
377
378 \cs_new_protected:Npn \__tag_check_mc_if_open:
379 {
380   \__tag_mc_if_in:F
381   {
382     \msg_warning:nne { tag } {mc-not-open} { \__tag_get_mc_abs_cnt: }
383   }
384 }

```

(End of definition for __tag_check_mc_if_nested: and __tag_check_mc_if_open:.)

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

```

385 \cs_new_protected:Npn \__tag_check_mc_pushed_popped:nn #1 #2
386 {
387   \int_compare:nNtT
388   { \l__tag_loglevel_int } = { 2 }
389   { \msg_info:nne {tag}{mc-#1}{#2} }
390   \int_compare:nNtT

```

```

391     { \l__tag_loglevel_int } > { 2 }
392     {
393         \msg_info:nne {tag}{mc-#1}{#2}
394         \seq_log:N \g__tag_mc_stack_seq
395     }
396 }

```

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

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

```

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

```

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

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

`__tag_check_init_mc_used:`

```

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

```

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

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

```

416 \cs_new_protected:Npn \__tag_check_mc_used:n #1 % #1 mcid absnt
417 {
418     \int_compare:nNnT {\l__tag_loglevel_int} > { 2 }
419     {
420         \__tag_check_init_mc_used:
421         \intarray_gset:Nnn \g__tag_check_mc_used_intarray
422             {#1}
423             { \intarray_item:Nn \g__tag_check_mc_used_intarray {#1} + 1 }
424         \int_compare:nNnT

```

```

425         {
426             \intarray_item:Nn \g__tag_check_mc_used_intarray {#1}
427         }
428         >
429         { 1 }
430         {
431             \msg_warning:nnn { tag } {mc-used-twice} {#1}
432         }
433     }
434 }

```

(End of definition for __tag_check_mc_used:n.)

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

```

435 \cs_new_protected:Npn \__tag_check_show_MCID_by_page:
436 {
437     \tl_set:Nc \l__tag_tmpa_tl
438     {
439         \__tag_property_ref_lastpage:nn
440         {abspage}
441         {-1}
442     }
443     \int_step_inline:nnnn {1}{1}
444     {
445         \l__tag_tmpa_tl
446     }
447     {
448         \seq_clear:N \l__tag_tmpa_seq
449         \int_step_inline:nnnn
450         {1}
451         {1}
452         {
453             \__tag_property_ref_lastpage:nn
454             {tagmcabs}
455             {-1}
456         }
457         {
458             \int_compare:nT
459             {
460                 \property_ref:enn
461                 {mcid-####1}
462                 {tagabspage}
463                 {-1}
464                 =
465                 ##1
466             }
467             {
468                 \seq_gput_right:Nc \l__tag_tmpa_seq
469                 {
470                     Page##1-####1-
471                     \property_ref:enn
472                     {mcid-####1}
473                     {tagmcid}
474                     {-1}

```

```

475         }
476     }
477 }
478 \seq_show:N \l__tag_tmpa_seq
479 }
480 }

```

(End of definition for `__tag_check_show_MCID_by_page:.`)

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

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

`__tag_check_mc_in_galley_p:` At first we need a test to decide if `\tag_mc_begin:n` (`tmb`) and `\tag_mc_end:` (`tme`) has been used at all on the current galley. As each command issues two slightly different marks we can do it by comparing firstmarks and botmarks. The test assumes that the marks have been already mapped into the sequence with `\@@_mc_get_marks:.` As `\seq_if_eq:NNTF` doesn't exist we use the `tl-test`.

`__tag_check_mc_in_galley:TF`

```

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

```

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

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

`__tag_check_if_mc_tmb_missing:TF`

```

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

```

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

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

`__tag_check_if_mc_tme_missing:TF`

```

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

```

```

501     { \prg_return_true: }
502     { \prg_return_false: }
503 }

```

(End of definition for __tag_check_if_mc_tme_missing:TF.)

```

504 \</package>
505 <*debug>

```

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

```

506 \msg_new:nnn { tag / debug } {mc-begin} { MC~begin~#1~with~options:~\tl_to_str:n{#2}~[\msg_line_context:] }
507 \msg_new:nnn { tag / debug } {mc-end} { MC~end~#1~[\msg_line_context:] }
508
509 \cs_new_protected:Npn \__tag_debug_mc_begin_insert:n #1
510 {
511   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
512   {
513     \msg_note:nnnn { tag / debug } {mc-begin} {inserted} { #1 }
514   }
515 }
516 \cs_new_protected:Npn \__tag_debug_mc_begin_ignore:n #1
517 {
518   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
519   {
520     \msg_note:nnnn { tag / debug } {mc-begin } {ignored} { #1 }
521   }
522 }
523 \cs_new_protected:Npn \__tag_debug_mc_end_insert:
524 {
525   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
526   {
527     \msg_note:nnn { tag / debug } {mc-end} {inserted}
528   }
529 }
530 \cs_new_protected:Npn \__tag_debug_mc_end_ignore:
531 {
532   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
533   {
534     \msg_note:nnn { tag / debug } {mc-end } {ignored}
535   }
536 }

```

And now something for the structures

```

537 \msg_new:nnn { tag / debug } {struct-begin}
538 {
539   Struct~\tag_get:n{struct_num}~begin~#1~with~options:~\tl_to_str:n{#2}~\[\msg_line_context:]
540 }
541 \msg_new:nnn { tag / debug } {struct-end}
542 {
543   Struct~end~#1~[\msg_line_context:]
544 }
545 \msg_new:nnn { tag / debug } {struct-end-wrong}
546 {
547   Struct~end~'#1'~doesn't~fit~start~'#2'~[\msg_line_context:]

```

```

548     }
549
550 \cs_new_protected:Npn \__tag_debug_struct_begin_insert:n #1
551 {
552     \int_compare:nNnT { \l__tag_loglevel_int } > {0}
553     {
554         \msg_note:nnnn { tag / debug } {struct-begin} {inserted} { #1 }
555         \seq_log:N \g__tag_struct_tag_stack_seq
556     }
557 }
558 \cs_new_protected:Npn \__tag_debug_struct_begin_ignore:n #1
559 {
560     \int_compare:nNnT { \l__tag_loglevel_int } > {0}
561     {
562         \msg_note:nnnn { tag / debug } {struct-begin } {ignored} { #1 }
563     }
564 }
565 \cs_new_protected:Npn \__tag_debug_struct_end_insert:
566 {
567     \int_compare:nNnT { \l__tag_loglevel_int } > {0}
568     {
569         \msg_note:nnn { tag / debug } {struct-end} {inserted}
570         \seq_log:N \g__tag_struct_tag_stack_seq
571     }
572 }
573 \cs_new_protected:Npn \__tag_debug_struct_end_ignore:
574 {
575     \int_compare:nNnT { \l__tag_loglevel_int } > {0}
576     {
577         \msg_note:nnn { tag / debug } {struct-end } {ignored}
578     }
579 }
580 \cs_new_protected:Npn \__tag_debug_struct_end_check:n #1
581 {
582     \int_compare:nNnT { \l__tag_loglevel_int } > {0}
583     {
584         \seq_get:NNT \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
585         {
586             \str_if_eq:eeF
587             {#1}
588             {\exp_last_unbraced:No \use_i:nn { \l__tag_tmpa_tl }}
589             {
590                 \msg_warning:nnee { tag/debug } {struct-end-wrong }
591                 {#1}
592                 {\exp_last_unbraced:No \use_i:nn { \l__tag_tmpa_tl }}
593             }
594         }
595     }
596 }

```

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

```

597 \msg_new:nnn { tag / debug } {tag-suspend}
598 {

```

```

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

```

6.6 Benchmarks

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

```

614 <*package>
615 \cs_new_protected:Npn \__tag_check_benchmark_tic:{}
616 \cs_new_protected:Npn \__tag_check_benchmark_toc:{}
617 \cs_new_protected:Npn \tag_check_benchmark_on:
618   {
619     \cs_if_exist:NT \benchmark_tic:
620     {
621       \cs_set_eq:NN \__tag_check_benchmark_tic: \benchmark_tic:
622       \cs_set_eq:NN \__tag_check_benchmark_toc: \benchmark_toc:
623     }
624   }
625 </package>

```

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

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

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

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

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

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

2 Commands related to mc-chunks

<code>\tagmcbegin</code>	<code>\tagmcbegin{\key-val}</code>
<code>\tagmcend</code>	<code>\tagmcend</code>
<code>\tagmcuse</code>	<code>\tagmcuse{\label}</code>

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

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

This is a wrapper around `\tag_mc_if_in:TF`. and tests if an mc is open or not. It is mostly of importance for 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.

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.

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

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

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

5.3 Header and footer

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

`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

<code>\tag_socket_use:n</code>	<code>\tag_socket_use:n {<socket name>}</code>
<code>\tag_socket_use:nn</code>	<code>\tag_socket_use:nn {<socket name>} {<socket argument>}</code>
<code>\UseTaggingSocket</code>	<code>\tag_socket_use:nnn {<socket name>} {<socket argument>} {<socket argument>}</code>

```
\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} {2025-05-16} {0.99q}
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 40.)

`\tag_tool:n`
`\tagtool`

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

```

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

```

(End of definition for `\tag_tool:n` and `\tagtool`. These functions are documented on page 40.)

9 Commands for the mc-chunks

```

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

```

(End of definition for `\tagmcbegin`, `\tagmcend`, and `\tagmcuse`. These functions are documented on page 40.)

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

```

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

```

(End of definition for `\tagmcifinTF`. This function is documented on page 40.)

10 Commands for the structure

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

```

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

```

```

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

```

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

11 Socket support

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

```

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

```

```

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

72 <*package>
73 \cs_set_protected:Npn \tag_socket_use:n #1
74 {
75   \bool_if:NT \l__tag_active_socket_bool
76   { \socket_use:n {tagsupport/#1} }
77 }
78 \cs_set_protected:Npn \tag_socket_use:nn #1#2
79 {
80   \bool_if:NT \l__tag_active_socket_bool
81   { \socket_use:nn {tagsupport/#1} {#2} }
82 }
83 \cs_set_protected:Npn \tag_socket_use:nnn #1#2#3
84 {
85   \bool_if:NTF \l__tag_active_socket_bool
86   { \socket_use:nnn {tagsupport/#1} {#2} {#3} }
87   { #3 }
88 }
89 \cs_set:Npn \tag_socket_use_expandable:n #1
90 {
91   \bool_if:NT \l__tag_active_socket_bool
92   { \socket_use_expandable:n {tagsupport/#1} }
93 }

```

```

94 \cs_set_protected:Npn \UseTaggingSocket #1
95 {
96   \bool_if:NTF \l__tag_active_socket_bool
97   { \socket_use:nw {tagsupport/#1} }
98   {
99     \int_case:nnF
100     { \int_use:c { c__socket_tagsupport/#1_args_int } }
101     {
102       0 \prg_do_nothing:
103       1 \use_none:n
104       2 \use_ii:nn

```

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

```

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

```

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

```

120     }
121     \ERRORusetaggingsocket
122   }
123 }
124 </package>

```

(End of definition for \tag_socket_use:n and others. These functions are documented on page 43.)

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 41.)

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

```

131 \keys_define:nn { __tag / show }
132 {
133   mc-data .code:n =
134   {
135     \bool_if:NT \g__tag_mode_lua_bool
136     {
137       \lua_now:e{ltx.__tag.trace.show_all_mc_data(#1,\__tag_get_mc_abs_cnt:,0)}
138     }
139   }
140   ,mc-data .default:n = 1
141 }
142

```

(End of definition for mc-data (show-key). This function is documented on page 41.)

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

```

143 \keys_define:nn { __tag / show }
144 { mc-current .code:n =
145   {
146     \bool_if:NTF \g__tag_mode_lua_bool
147     {
148       \int_compare:nNnTF
149       { -2147483647 }
150       =
151       {
152         \lua_now:e
153         {
154           tex.print
155           (tex.getattribute
156            (luatexbase.attributes.g__tag_mc_cnt_attr))
157         }
158       }
159     }
160     \lua_now:e
161     {
162       ltx.__tag.trace.log
163       (
164         "mc-current:~no-MC~open,~current~absent
165         =\__tag_get_mc_abs_cnt:"
166         ,0
167       )
168       texio.write_nl("")
169     }
170   }
171   {
172     \lua_now:e
173     {
174       ltx.__tag.trace.log
175       (
176         "mc-current:~absent=\__tag_get_mc_abs_cnt:=="
177         ..

```



```

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

```

(End of definition for mc-current (show-key). This function is documented on page 41.)

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).

```

202 \keys_define:nn { __tag / show }
203 {
204     mc-marks .choice: ,
205     mc-marks / show .code:n =
206     {
207         \__tag_mc_get_marks:
208         \__tag_check_if_mc_in_galley:TF
209         {
210             \iow_term:n {Marks~from~this~page:~}
211         }
212         {
213             \iow_term:n {Marks~from~a~previous~page:~}
214         }
215         \seq_show:N \l__tag_mc_firstmarks_seq
216         \seq_show:N \l__tag_mc_botmarks_seq
217         \__tag_check_if_mc_tmb_missing:T
218         {
219             \iow_term:n {BDC~missing~on~this~page!}
220         }
221         \__tag_check_if_mc_tme_missing:T
222         {
223             \iow_term:n {EMC~missing~on~this~page!}
224         }
225     },
226     mc-marks / use .code:n =

```

```

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

```

(End of definition for mc-marks (show-key). This function is documented on page 41.)

struct-stack (show-key)

```

245 \keys_define:nn { __tag / show }
246 {
247   struct-stack .choice:
248   ,struct-stack / log .code:n = \seq_log:N \g__tag_struct_tag_stack_seq
249   ,struct-stack / show .code:n = \seq_show:N \g__tag_struct_tag_stack_seq
250   ,struct-stack .default:n = show
251 }
252 \</package>

```

(End of definition for struct-stack (show-key). This function is documented on page 41.)

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.

```

253 <*debug>
254 \keys_define:nn { __tag / show }
255 {
256   ,debug/structures .code:n =
257   {
258     \int_step_inline:nnn{#1}{\c@g__tag_struct_abs_int}
259     {
260       \msg_term:nneeee
261       { tag/debug } { show-struct }
262       { ##1 }
263       {
264         \prop_map_function:cN
265         {g__tag_struct_debug_##1_prop}
266         \msg_show_item_unbraced:nn
267       }
268       { } { }
269       \msg_term:nneeee
270       { tag/debug } { show-kids }
271       { ##1 }

```

```

272         {
273             \seq_map_function:cN
274             {g__tag_struct_debug_kids_##1_seq}
275             \msg_show_item_unbraced:n
276         }
277         { } { }
278     }
279 }
280 ,debug/structures .default:n = 1
281 }
282 </debug>

```

(End of definition for `debug/structures` (`show-key`). This function is documented on page 41.)

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.

```

283 <*package>

```

13.1 Document structure

```

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

```

(End of definition for `\g__tag_root_default_tl`, `activate` (`setup-key`), and `activate/socket` (`setup-key`). These functions are documented on page 40.)

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.

```

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

```

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

```

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

```

(End of definition for \pdffakespace. This function is documented on page 42.)

13.4 Paratagging

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

```

\l__tag_para_bool
\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.

```

321 </package>
322 <base>\bool_new:N \l__tag_para_flattened_bool
323 <base>\bool_new:N \l__tag_para_bool
324 <*package>
325 \int_new:N \g__tag_para_begin_int
326 \int_new:N \g__tag_para_end_int
327 \int_new:N \g__tag_para_main_begin_int
328 \int_new:N \g__tag_para_main_end_int
329 \tl_new:N \g__tag_para_main_struct_tl
330 \tl_gset:Nn \g__tag_para_main_struct_tl {1}
331 \tl_new:N \l__tag_para_tag_default_tl
332 \tl_set:Nn \l__tag_para_tag_default_tl { text }
333 \tl_new:N \l__tag_para_tag_tl
334 \tl_set:Nn \l__tag_para_tag_tl { \l__tag_para_tag_default_tl }
335 \tl_new:N \l__tag_para_main_tag_tl
336 \tl_set:Nn \l__tag_para_main_tag_tl {text-unit}

```

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

this is perhaps already defined by the block code

```

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

```

(End of definition for \l__tag_para_bool and others.)

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

```

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

```

(End of definition for __tag_gincr_para_main_begin_int: and others.)

```

\__tag_start_para_ints:
\__tag_stop_para_ints:
356 \cs_new_protected:Npn \__tag_start_para_ints:
357 {
358   \cs_set_protected:Npn \__tag_gincr_para_main_begin_int:
359   {
360     \int_gincr:N \g__tag_para_main_begin_int
361   }
362   \cs_set_protected:Npn \__tag_gincr_para_begin_int:
363   {
364     \int_gincr:N \g__tag_para_begin_int
365   }
366   \cs_set_protected:Npn \__tag_gincr_para_main_end_int:
367   {
368     \int_gincr:N \g__tag_para_main_end_int
369   }
370   \cs_set_protected:Npn \__tag_gincr_para_end_int:
371   {
372     \int_gincr:N \g__tag_para_end_int
373   }
374 }
375 \cs_new_protected:Npn \__tag_stop_para_ints:
376 {
377   \cs_set_eq:NN \__tag_gincr_para_main_begin_int:\prg_do_nothing:
378   \cs_set_eq:NN \__tag_gincr_para_begin_int:\prg_do_nothing:
379   \cs_set_eq:NN \__tag_gincr_para_main_end_int:\prg_do_nothing:
380   \cs_set_eq:NN \__tag_gincr_para_end_int:\prg_do_nothing:
381 }

```

(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:`

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

(End of definition for `__tag_para_main_store_struct:.`)

temporary adaption for the block module:

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

para/tagging (setup-key)

para/tag (setup-key)

para/maintag (setup-key)

para/tagging (tool-key)

para/tag (tool-key)

para/maintag (tool-key)

para/flattened (tool-key)

unittag (deprecated)

para-flattened (deprecated)

paratagging (deprecated)

paratagging-show (deprecated)

paratag (deprecated)

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

```
393 \keys_define:nn { __tag / setup }
394 {
395   para/tagging      .bool_set:N = \l__tag_para_bool,
396   debug/show/para   .code:n = {\bool_set_true:N \l__tag_para_show_bool},
397   debug/show/paraOff .code:n = {\bool_set_false:N \l__tag_para_show_bool},
398   para/tag          .tl_set:N   = \l__tag_para_tag_tl,
399   para/maintag      .tl_set:N   = \l__tag_para_main_tag_tl,
400   para/flattened    .bool_set:N = \l__tag_para_flattened_bool
401 }
402 \keys_define:nn { tag / tool}
403 {
404   para/tagging      .bool_set:N = \l__tag_para_bool,
405   para/tag          .tl_set:N   = \l__tag_para_tag_tl,
406   para/maintag      .tl_set:N   = \l__tag_para_main_tag_tl,
407   para/flattened    .bool_set:N = \l__tag_para_flattened_bool
408 }
```

the deprecated names

```
409 \keys_define:nn { __tag / setup }
410 {
411   paratagging      .bool_set:N = \l__tag_para_bool,
412   paratagging-show .bool_set:N = \l__tag_para_show_bool,
413   paratag          .tl_set:N   = \l__tag_para_tag_tl
414 }
415 \keys_define:nn { tag / tool}
416 {
417   para      .bool_set:N = \l__tag_para_bool,
```

```

418     paratag .tl_set:N = \l__tag_para_tag_tl,
419     unittag .tl_set:N = \l__tag_para_main_tag_tl,
420     para-flattened .bool_set:N = \l__tag_para_flattened_bool
421 }

```

(End of definition for para/tagging (setup-key) and others. These functions are documented on page 42.)

Helper command for debugging:

```

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

```

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.

```

443 \str_if_exist:cF { l__socket_tagssupport/para/begin_plug_str }
444 {
445   \socket_new:nn      {tagssupport/para/begin}{0}
446   \socket_new:nn      {tagssupport/para/end}{0}
447
448   \socket_new_plug:nnn{tagssupport/para/begin}{plain}
449   {
450     \bool_if:NT \l__tag_para_bool
451     {
452       \bool_if:NF \l__tag_para_flattened_bool
453       {
454         \__tag_gincr_para_main_begin_int:
455         \tag_struct_begin:n
456         {
457           tag=\l__tag_para_main_tag_tl,
458         }
459         \__tag_para_main_store_struct:
460       }
461       \__tag_gincr_para_begin_int:

```

```

462         \tag_struct_begin:n {tag=\l__tag_para_tag_tl}
463         \__tag_check_para_begin_show:nn {green}{}
464         \tag_mc_begin:n {}
465     }
466 }
467 \socket_new_plug:nnn{tagsupport/para/begin}{block}
468 {
469     \bool_if:NT \l__tag_para_bool
470     {
471         \legacy_if:nF { @inlabel }
472         {
473             \__tag_check_typeout_v:n
474             {==>~ @endpe = \legacy_if:nTF { @endpe }{true}{false} \on@line }
475             \legacy_if:nF { @endpe }
476             {
477                 \bool_if:NF \l__tag_para_flattened_bool
478                 {
479                     \__tag_gincr_para_main_begin_int:
480                     \tag_struct_begin:n
481                     {
482                         tag=\l__tag_para_main_tag_tl,
483                         attribute-class=\l__tag_para_main_attr_class_tl,
484                     }
485                     \__tag_para_main_store_struct:
486                 }
487             }
488             \__tag_gincr_para_begin_int:
489             \__tag_check_typeout_v:n {==>~increment~ P \on@line }
490             \tag_struct_begin:n
491             {
492                 tag=\l__tag_para_tag_tl
493                 ,attribute-class=\l__tag_para_attr_class_tl
494             }
495             \__tag_check_para_begin_show:nn {green}{\PARALABEL}
496             \tag_mc_begin:n {}
497         }
498     }
499 }

```

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

```

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

```



```

513         }
514     }
515 }
516 }

```

By default we assign the plain plug:

```

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

```

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

```

519 \AddToHook{para/begin}{ \socket_use:n { tagsupport/para/begin }
520 }
521 \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.

```

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

```

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.

```

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

```

```

556         {para-hook-count-wrong}
557         {\int_use:N\g__tag_para_begin_int}
558         {\int_use:N\g__tag_para_end_int}
559         {text}
560     }
561 }

```

13.5 output routine stuff

We need at least the new-or-1 code. In generic mode we also must insert the code to finish the MC-chunks This part here can go in June 2025

```

562 \ifpackageloaded{footmisc}
563   {\PackageWarning{tagpdf}{tagpdf~has~been~loaded~too~late!}} %
564   {\RequirePackage{latex-lab-testphase-new-or-1}}
565
566 \AddToHook{begindocument/before}
567 {
568   \bool_if:NF \g__tag_mode_lua_bool
569   {
570     \cs_if_exist:NT \@kernel@before@footins
571     {
572       \tl_put_right:Nn \@kernel@before@footins
573       { \tag_mc_add_missing_to_stream:Nn \footins {footnote} }
574       \tl_put_right:Nn \@kernel@tagsupport@@makecol
575       {
576         \__tag_check_typeout_v:n {====>~In~\token_to_str:N \@makecol\c_space_tl\the\c@
577         \tag_mc_add_missing_to_stream:Nn \@outputbox {main}
578       }
579     }
580   }
581 }
582

```

If the new OR is there, we use it

```

583 \str_if_exist:cT { l_socket_tagsupport/build/column/outputbox_plug_str }
584 {
585   \NewSocketPlug{tagsupport/build/column/outputbox}{tagpdf}
586   {
587     \__tag_check_typeout_v:n {====>~In~\token_to_str:N \@makecol
588     \c_space_tl\the\c@page }
589     \tag_mc_add_missing_to_stream:Nn \@outputbox {main}
590   }
591   \NewSocketPlug{tagsupport/build/column/footins}{tagpdf}
592   { \tag_mc_add_missing_to_stream:Nn \footins {footnote} }
593
594   \bool_if:NF \g__tag_mode_lua_bool
595   {
596     \AssignSocketPlug{tagsupport/build/column/outputbox}{tagpdf}
597     \AssignSocketPlug{tagsupport/build/column/footins}{tagpdf}
598   }
599 }
600 \end{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}`

```

601 <base>\newcommand\tagpdfparaOn {}
602 <base>\newcommand\tagpdfparaOff{}
603 <*package>
604 \renewcommand\tagpdfparaOn {\bool_set_true:N \l__tag_para_bool}
605 \renewcommand\tagpdfparaOff{\bool_set_false:N \l__tag_para_bool}

```

(End of definition for `\tagpdfparaOn` and `\tagpdfparaOff`. These functions are documented on page 42.)

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

```

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

606 \NewDocumentCommand\tagpdfsuppressmarks{m}
607 {{\use:c{__tag_mc_disable_marks:} #1}}

```

(End of definition for `\tagpdfsuppressmarks`. This function is documented on page 42.)

13.6 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)`

```

608 \keys_define:nn { __tag / setup }
609 {
610   text / lang .tl_set:N = \l__tag_struct_lang_tl
611 }

```

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

13.7 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.

```

612 \cs_new_protected:Npn\__tag_hook_kernel_before_head:{}
613 \cs_new_protected:Npn\__tag_hook_kernel_after_head:{}
614 \cs_new_protected:Npn\__tag_hook_kernel_before_foot:{}
615 \cs_new_protected:Npn\__tag_hook_kernel_after_foot:{}

```

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

```
616 \AddToHook{begindocument}
617 {
618   \cs_if_exist:NT \@kernel@before@head
619   {
620     \tl_put_right:Nn \@kernel@before@head {\__tag_hook_kernel_before_head:}
621     \tl_put_left:Nn \@kernel@after@head {\__tag_hook_kernel_after_head:}
622     \tl_put_right:Nn \@kernel@before@foot {\__tag_hook_kernel_before_foot:}
623     \tl_put_left:Nn \@kernel@after@foot {\__tag_hook_kernel_after_foot:}
624   }
625 }
```

If the new page sockets exist, we use them.

```
626 \str_if_exist:cT { l__socket_tagsupport/build/page/footer_plug_str }
627 {
628   \NewSocketPlug{tagsupport/build/page/header}{tagpdf}
629   {
630     \__tag_hook_kernel_before_head:
631     #2
632     \__tag_hook_kernel_after_head:
633   }
634
635   \AssignSocketPlug{tagsupport/build/page/header}{tagpdf}
636   \NewSocketPlug{tagsupport/build/page/footer}{tagpdf}
637   {
638     \__tag_hook_kernel_before_foot:
639     #2
640     \__tag_hook_kernel_after_foot:
641   }
642   \AssignSocketPlug{tagsupport/build/page/footer}{tagpdf}
643 }
644
645 \bool_new:N \g__tag_saved_in_mc_bool
646 \cs_new_protected:Npn \__tag_exclude_headfoot_begin:
647 {
648   \bool_set_false:N \l__tag_para_bool
649   \bool_if:NTF \g__tag_mode_lua_bool
650   {
651     \tag_mc_end_push:
652   }
653   {
654     \bool_gset_eq:NN \g__tag_saved_in_mc_bool \g__tag_in_mc_bool
655     \bool_gset_false:N \g__tag_in_mc_bool
656   }
657   \tag_mc_begin:n {artifact}
658   \tag_suspend:n{headfoot}
659 }
660 \cs_new_protected:Npn \__tag_exclude_headfoot_end:
661 {
662   \tag_resume:n{headfoot}
663   \tag_mc_end:
664   \bool_if:NTF \g__tag_mode_lua_bool
665   {
666     \tag_mc_begin_pop:n{}
```

```

667     }
668     {
669         \bool_gset_eq:NN \g__tag_in_mc_bool\g__tag_saved_in_mc_bool
670     }
671 }

```

This version allows to use an Artifact structure

```

672 \__tag_attr_new_entry:nn {__tag/attr/pagination}{/0/Artifact/Type/Pagination}
673 \cs_new_protected:Npn \__tag_exclude_struct_headfoot_begin:n #1
674 {
675     \bool_set_false:N \l__tag_para_bool
676     \bool_if:NTF \g__tag_mode_lua_bool
677     {
678         \tag_mc_end_push:
679     }
680     {
681         \bool_gset_eq:NN \g__tag_saved_in_mc_bool \g__tag_in_mc_bool
682         \bool_gset_false:N \g__tag_in_mc_bool
683     }
684     \tag_struct_begin:n{tag=Artifact,attribute-class=__tag/attr/#1}
685     \tag_mc_begin:n {artifact=#1}
686     \tag_suspend:n{headfoot}
687 }
688
689 \cs_new_protected:Npn \__tag_exclude_struct_headfoot_end:
690 {
691     \tag_resume:n{headfoot}
692     \tag_mc_end:
693     \tag_struct_end:
694     \bool_if:NTF \g__tag_mode_lua_bool
695     {
696         \tag_mc_begin_pop:n{ }
697     }
698     {
699         \bool_gset_eq:NN \g__tag_in_mc_bool\g__tag_saved_in_mc_bool
700     }
701 }

```

And now the keys

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

exclude-header-footer (deprecated)

```

702 \keys_define:nn { __tag / setup }
703 {
704     page/exclude-header-footer .choice:,
705     page/exclude-header-footer / true .code:n =
706     {
707         \cs_set_eq:NN \__tag_hook_kernel_before_head: \__tag_exclude_headfoot_begin:
708         \cs_set_eq:NN \__tag_hook_kernel_before_foot: \__tag_exclude_headfoot_begin:
709         \cs_set_eq:NN \__tag_hook_kernel_after_head: \__tag_exclude_headfoot_end:
710         \cs_set_eq:NN \__tag_hook_kernel_after_foot: \__tag_exclude_headfoot_end:
711     },
712     page/exclude-header-footer / pagination .code:n =
713     {
714         \cs_set:Nn \__tag_hook_kernel_before_head: { \__tag_exclude_struct_headfoot_begin:n {p
715         \cs_set:Nn \__tag_hook_kernel_before_foot: { \__tag_exclude_struct_headfoot_begin:n {p

```

```

716     \cs_set_eq:NN \__tag_hook_kernel_after_head: \__tag_exclude_struct_headfoot_end:
717     \cs_set_eq:NN \__tag_hook_kernel_after_foot: \__tag_exclude_struct_headfoot_end:
718   },
719   page/exclude-header-footer / false .code:n =
720   {
721     \cs_set_eq:NN \__tag_hook_kernel_before_head: \prg_do_nothing:
722     \cs_set_eq:NN \__tag_hook_kernel_before_foot: \prg_do_nothing:
723     \cs_set_eq:NN \__tag_hook_kernel_after_head: \prg_do_nothing:
724     \cs_set_eq:NN \__tag_hook_kernel_after_foot: \prg_do_nothing:
725   },
726   page/exclude-header-footer .default:n = true,
727   page/exclude-header-footer .initial:n = true,
  deprecated name
728   exclude-header-footer .meta:n = { page/exclude-header-footer = {#1} }
729 }

```

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

13.8 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; this is added for normal links by the generic hyperref driver.

```

730 \hook_gput_code:nnn
731 {pdfannot/link/URI/before}
732 {tagpdf}
733 {
734   \tag_mc_end_push:
735   \tag_struct_begin:n { tag=Link }
736   \tag_mc_begin:n { tag=Link }
737   \pdfannot_dict_put:nne
738     { link/URI }
739     { StructParent }
740     { \tag_struct_parent_int: }
741 }
742
743 \hook_gput_code:nnn
744 {pdfannot/link/URI/after}
745 {tagpdf}
746 {
747   \tag_struct_insert_annot:ee {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
748   \tag_mc_end:
749   \tag_struct_end:
750   \tag_mc_begin_pop:n{ }
751 }
752
753 \hook_gput_code:nnn
754 {pdfannot/link/GoTo/before}
755 {tagpdf}
756 {
757   \tag_mc_end_push:
758   \tag_struct_begin:n{tag=Link}

```

```

759     \tag_mc_begin:n{tag=Link}
760     \pdfannot_dict_put:nne
761       { link/GoTo }
762       { StructParent }
763       { \tag_struct_parent_int: }
764   }
765
766   \hook_gput_code:nnn
767     {pdfannot/link/GoTo/after}
768     {tagpdf}
769   {
770     \tag_struct_insert_annot:ee {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
771     \tag_mc_end:
772     \tag_struct_end:
773     \tag_mc_begin_pop:n{ }
774   }
775
776   \hook_gput_code:nnn
777     {pdfannot/link/GoToR/before}
778     {tagpdf}
779   {
780     \tag_mc_end_push:
781     \tag_struct_begin:n{tag=Link}
782     \tag_mc_begin:n{tag=Link}
783     \pdfannot_dict_put:nne
784       { link/GoToR }
785       { StructParent }
786       { \tag_struct_parent_int: }
787   }
788
789   \hook_gput_code:nnn
790     {pdfannot/link/GoToR/after}
791     {tagpdf}
792   {
793     \tag_struct_insert_annot:ee {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
794     \tag_mc_end:
795     \tag_struct_end:
796     \tag_mc_begin_pop:n{ }
797   }

```

13.9 Attaching css-files for derivation

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

Such CSS style definitions can be given in two ways:

- In files with the extension `.css`. Such files should contain only CSS style definitions. ngpdf will store these files and include them with an `<link rel=stylesheet href=...>` in the head of the html.
- In files with the extension `.html`. Such files should contain CSS style definitions inside one (or more) `<style>...</style>` html tags. The content of these files are copied by ngpdf directly into the head of the derived html.

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

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

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

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

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

The files in this list are attached at the end of the compilation (and so normally after the files attached with `root-supplemental-file`) but you shouldn't rely on this or on a specific order of the embedding in the html.

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

```

798 \prop_new:N \g__tag_css_prop
799 \prop_gset_from_keyval:Nn \g__tag_css_prop
800 {
801   latex-list-css.html=true,
802   latex-align-css.html=true
803 }
804
805
806 \bool_new:N \g__tag_css_bool
807 \bool_gset_true:N \g__tag_css_bool
808
809 \hook_gput_code:nnn{tagpdf/finish/before}{tagpdf/css}
810 {
811   \bool_lazy_and:nnT { \g__tag_css_bool }{ \tag_if_active_p: }
812   {
813     \prop_map_inline:Nn \g__tag_css_prop
814     {
815       \keys_set:nn { __tag / setup }{ root-supplemental-file= {#1} }
816     }
817   }
818 }
819
820 \keys_define:nn { __tag / setup }
821 {
822   attach-css .bool_gset:N = \g__tag_css_bool,
823   css-list .code:n =
824   {
825     \tl_if_empty:nTF{#1}
826     { \prop_gclear:N \g__tag_css_prop }
827     { \prop_gput:Nnn \g__tag_css_prop { #1 }{true} }
828   },
829   css-list-add .code:n = { \prop_gput:Nnn \g__tag_css_prop { #1 }{true} },

```



```
830     css-list-remove .code:n = { \prop_gremove:Nn \g__tag_css_prop { #1 } },
831   }
</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} {2025-05-16} {0.99q}
4 {part of tagpdf - code related to writing trees and dictionaries to the pdf}
5 </header>
```

1 Trees, pdfmanagement and finalization code

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

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

1.1 Check structure

__tag_tree_final_checks:

```
20 \cs_new_protected:Npn \__tag_tree_final_checks:
21 {
22   \int_compare:nNnF {\seq_count:N\g__tag_struct_stack_seq}={1}
23   {
24     \msg_warning:nn {tag}{tree-struct-still-open}
25     \int_step_inline:nnn{2}{\seq_count:N\g__tag_struct_stack_seq}
26     {\tag_struct_end:}
27   }
28   \socket_use:n { tag/check/parent-child-end }
29   \msg_note:nn {tag}{tree-statistic}
30 }
```

(End of definition for __tag_tree_final_checks:.)

1.2 Catalog: MarkInfo and StructTreeRoot and OpenAction

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

```
--tag/struct/1 This is the object for the root object, the StructTreeRoot
31 \pdf_object_new_indexed:nn { __tag/struct }{ 1 }
(End of definition for __tag/struct/1.)
```

```
\g__tag_tree_openaction_struct_tl We need a variable that indicates which structure is wanted in the OpenAction. By
default we use 2 (the Document structure).
32 \tl_new:N \g__tag_tree_openaction_struct_tl
33 \tl_gset:Nn \g__tag_tree_openaction_struct_tl { 2 }
(End of definition for \g__tag_tree_openaction_struct_tl.)
```

```
viewer/startstructure (setup-key) We also need an option to setup the start structure. So we setup a key which sets the
variable to the current structure. This still requires hyperref to do most of the job, this
should perhaps be changed.
34 \keys_define:nn { __tag / setup }
35 {
36   viewer/startstructure .code:n =
37   {
38     \tl_gset:Nn \g__tag_tree_openaction_struct_tl {#1}
39   }
40   ,viewer/startstructure .default:n = { \int_use:N \c@g__tag_struct_abs_int }
41 }
```

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

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

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

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

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

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

```

58             \int_compare:nNnTF{ \seq_count:N \l__tag_tmpa_seq } > {1}
59             { /\seq_item:Nn\l__tag_tmpa_seq{2} }
60             { ] }
61         >>
62     }
63 }
64 }
65 }
66 \hook_gput_code:nnn{shipout/lastpage}{tagpdf}
67 {
68     \bool_if:NT \g__tag_active_tree_bool
69     {
70         \pdfmanagement_add:nnn { Catalog / MarkInfo } { Marked } { true }
71         \pdfmanagement_add:nne
72         { Catalog }
73         { StructTreeRoot }
74         { \pdf_object_ref_indexed:nn { __tag/struct } { 1 } }
75         \__tag_tree_update_openaction:
76     }
77 }

```

1.3 Writing the IDtree

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

```

\g__tag_tree_id_pad_int
78 \int_new:N\g__tag_tree_id_pad_int
(End of definition for \g__tag_tree_id_pad_int.)
Now we get the needed padding
79 \cs_generate_variant:Nn \tl_count:n {e}
80 \hook_gput_code:nnn{begindocument}{tagpdf}
81 {
82     \int_gset:Nn\g__tag_tree_id_pad_int
83     {\tl_count:e { \__tag_property_ref_lastpage:nn{tagstruct}{1000}}+1}
84 }
85

```

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

```

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

```

```

95         {
96         \__tag_struct_get_id:n{##1}~\pdf_object_ref_indexed:nn {\__tag/struct}{##1}~
97         }
98     \int_compare:nNnF {\l__tag_tmpa_int}<{50} %
99     {
100         \pdf_object_unnamed_write:ne {dict}
101         { /Limits~[\__tag_struct_get_id:n{##1}~\l__tag_tmpa_int+1}~\__tag_struct_get_id:
102         /Names~[\l__tag_tmpa_tl]
103         }
104         \tl_put_right:Ne\l__tag_tmpb_tl {\pdf_object_ref_last:\c_space_tl}
105         \int_zero:N \l__tag_tmpa_int
106         \tl_clear:N \l__tag_tmpa_tl
107     }
108 }
109 \tl_if_empty:NF \l__tag_tmpa_tl
110 {
111     \pdf_object_unnamed_write:ne {dict}
112     {
113         /Limits~
114         [\__tag_struct_get_id:n{\c@g__tag_struct_abs_int~\l__tag_tmpa_int+1}~
115         \__tag_struct_get_id:n{\c@g__tag_struct_abs_int}]
116         /Names~[\l__tag_tmpa_tl]
117     }
118     \tl_put_right:Ne\l__tag_tmpb_tl {\pdf_object_ref_last:}
119 }
120 \pdf_object_unnamed_write:ne {dict}{/Kids~[\l__tag_tmpb_tl]}
121 \__tag_prop_gput:cne
122 { g__tag_struct_1_prop }
123 { IDTree }
124 { \pdf_object_ref_last: }
125 }

```

1.4 Writing structure elements

The following commands are needed to write out the structure.

`__tag_tree_write_structtreeroot:` This writes out the root object.

```

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

```

```

143         \l__tag_tmpa_tl
144     }

```

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

```

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

```

(End of definition for __tag_tree_write_structtreeroot:.)

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

```

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

```

(End of definition for __tag_tree_write_structelems:.)

1.5 ParentTree

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

```

154 \pdf_object_new:n { __tag/tree/parenttree }

```

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

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

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

```

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

```

(End of definition for \c@g__tag_parenttree_obj_int.)

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

\g__tag_parenttree_objr_tl

```

162 \tl_new:N \g__tag_parenttree_objr_tl

```

(End of definition for \g__tag_parenttree_objr_tl.)

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

```

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

```

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

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

```

170 \tl_new:N \l__tag_parenttree_content_tl

```

(End of definition for `\l__tag_parenttree_content_tl`.)

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

```

171 \cs_new_protected:Npn \_tag_tree_parenttree_rerun_msg: {}
172 \cs_new_protected:Npn \_tag_tree_fill_parenttree:
173 {
174   \int_step_inline:nnnn{1}{1}{\_tag_property_ref_lastpage:nn{abspage}{-1}} %not quite clear
175   { %page ##1
176     \prop_clear:N \l__tag_tmpa_prop
177     \int_step_inline:nnnn{1}{1}{\_tag_property_ref_lastpage:nn{tagmcabs}{-
178       1}}
179     {
180       %mcid####1
181       \int_compare:nT
182       {\property_ref:enn{mcid-####1}{tagabspage}{-1}=##1} %mcid is on current page
183       {% yes
184         \prop_get:NnNT
185         \g__tag_mc_parenttree_prop
186         {####1}
187         \l__tag_tmpa_tl
188         {
189           \prop_put:Nee
190           \l__tag_tmpa_prop
191           {\property_ref:enn{mcid-####1}{tagmcid}{-1}}
192           {\l__tag_tmpa_tl}
193         }
194       }
195       \tl_put_right:Ne\l__tag_parenttree_content_tl
196       {
197         \int_eval:n {##1-1}\c_space_tl
198         [\c_space_tl %]
199       }
200       \int_step_inline:nnnn %####1
201       {0}
202       {1}
203       { \prop_count:N \l__tag_tmpa_prop -1 }
204       {

```

```

205 \prop_get:NnNTF \l__tag_tmpa_prop {###1} \l__tag_tmpa_tl
206 {% page#1:mcid##1:\l__tag_tmpa_tl :content
207 \tl_put_right:Ne \l__tag_parenttree_content_tl
208 {
209 \prop_if_exist:cTF { g__tag_struct_ \l__tag_tmpa_tl _prop }
210 {
211 \pdf_object_ref_indexed:nn { __tag/struct }{ \l__tag_tmpa_tl }
212 }
213 {
214 null
215 }
216 \c_space_tl
217 }
218 }
219 {
220 \cs_set_protected:Npn \__tag_tree_parenttree_rerun_msg:
221 {
222 \msg_warning:nn { tag } {tree-mcid-index-wrong}
223 }
224 }
225 }
226 \tl_put_right:Nn
227 \l__tag_parenttree_content_tl
228 {%[
229 ]^^J
230 }
231 }
232 }

```

(End of definition for __tag_tree_fill_parenttree:.)

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

```

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

```

(End of definition for __tag_tree_lua_fill_parenttree:.)

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

```

246 \cs_new_protected:Npn \__tag_tree_write_parenttree:
247 {
248 \bool_if:NTF \g__tag_mode_lua_bool
249 {

```



```

250     \__tag_tree_lua_fill_parenttree:
251   }
252   {
253     \__tag_tree_fill_parenttree:
254   }
255   \__tag_tree_parenttree_rerun_msg:
256   \tl_put_right:No \l__tag_parenttree_content_tl { \g__tag_parenttree_objr_tl }
257   \pdf_object_write:nne { __tag/tree/parenttree }{dict}
258   {
259     /Nums\c_space_tl [\l__tag_parenttree_content_tl]
260   }
261 }

```

(End of definition for __tag_tree_write_parenttree:.)

1.6 Rolemap dictionary

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

`__tag/tree/rolemap` At first we reserve again an object. Rolemap is also used in PDF 2.0 as a fallback.

```

262 \pdf_object_new:n { __tag/tree/rolemap }

```

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

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

```

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

```

(End of definition for __tag_tree_write_rolemap:.)

1.7 Classmap dictionary

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

```
\__tag_tree_write_classmap:
```

```
286 \cs_new_protected:Npn \__tag_tree_write_classmap:
287 {
288   \tl_clear:N \l__tag_tmpa_tl
289   \seq_map_inline:Nn \g__tag_attr_class_used_seq
290   {
291     \prop_gput:Nnn \g__tag_attr_class_used_prop {##1}{ }
292   }
293   \prop_map_inline:Nn \g__tag_attr_class_used_prop
294   {
295     \prop_get:NnNT \g__tag_attr_entries_prop {##1} \l__tag_tmpb_tl
296     {
297       \tl_put_right:Ne \l__tag_tmpa_tl
298       {
299         ##1\c_space_tl
300         <<
301         \l__tag_tmpb_tl
302         >>
303         \iow_newline:
304       }
305     }
306   }
307   \tl_if_empty:NF
308   \l__tag_tmpa_tl
309   {
310     \pdf_object_new:n { __tag/tree/classmap }
311     \pdf_object_write:nne
312     { __tag/tree/classmap }
313     {dict}
314     { \l__tag_tmpa_tl }
315     \__tag_prop_gput:cne
316     { g__tag_struct_1_prop }
317     { ClassMap }
318     { \pdf_object_ref:n { __tag/tree/classmap } }
319   }
320 }
```

(End of definition for __tag_tree_write_classmap:.)

1.8 Namespaces

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

```
__tag/tree/namespaces
```

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

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

`__tag_tree_write_namespaces:`

```
322 \cs_new_protected:Npn \__tag_tree_write_namespaces:
323 {
324   \pdf_version_compare:NnF < {2.0}
325   {
326     \prop_map_inline:Nn \g__tag_role_NS_prop
327     {
328       \pdfdict_if_empty:NF {g__tag_role/RoleMapNS/##1_dict}
329       {
330         \pdf_object_write:nne {__tag/RoleMapNS/##1}{dict}
331         {
332           \pdfdict_use:n {g__tag_role/RoleMapNS/##1_dict}
333         }
334         \pdfdict_gput:nne{g__tag_role/RoleMapNS/##1}{dict}
335         {RoleMapNS}{\pdf_object_ref:n {__tag/RoleMapNS/##1}}
336       }
337       \pdf_object_write:nne{tag/NS/##1}{dict}
338       {
339         \pdfdict_use:n {g__tag_role/RoleMapNS/##1_dict}
340       }
341     }
342     \pdf_object_write:nne {__tag/tree/namespaces}{array}
343     {
344       \prop_map_tokens:Nn \g__tag_role_NS_prop{\use_ii:nn}
345     }
346   }
347 }
```

(End of definition for `__tag_tree_write_namespaces:.`)

1.9 Finishing the structure

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

`__tag_finish_structure:`

```
348 \hook_new:n {tagpdf/finish/before}
349 \cs_new_protected:Npn \__tag_finish_structure:
350 {
351   \bool_if:NT\g__tag_active_tree_bool
352   {
353     \hook_use:n {tagpdf/finish/before}
354     \__tag_tree_final_checks:
355     \iow_term:n{Package~tagpdf~Info:~writing~ParentTree}
356     \__tag_check_benchmark_tic:
357     \__tag_tree_write_parenttree:
358     \__tag_check_benchmark_toc:
359     \iow_term:n{Package~tagpdf~Info:~writing~IDTree}
360     \__tag_check_benchmark_tic:
361     \__tag_tree_write_idtree:
362     \__tag_check_benchmark_toc:
363     \iow_term:n{Package~tagpdf~Info:~writing~RoleMap}
```

```

364     \__tag_check_benchmark_tic:
365     \__tag_tree_write_rolemap:
366     \__tag_check_benchmark_toc:
367     \iow_term:n{Package~tagpdf~Info:~writing~ClassMap}
368     \__tag_check_benchmark_tic:
369     \__tag_tree_write_classmap:
370     \__tag_check_benchmark_toc:
371     \iow_term:n{Package~tagpdf~Info:~writing~NameSpaces}
372     \__tag_check_benchmark_tic:
373     \__tag_tree_write_namespaces:
374     \__tag_check_benchmark_toc:
375     \iow_term:n{Package~tagpdf~Info:~writing~StructElems}
376     \__tag_check_benchmark_tic:
377     \__tag_tree_write_structelements: %this is rather slow!!
378     \__tag_check_benchmark_toc:
379     \iow_term:n{Package~tagpdf~Info:~writing~Root}
380     \__tag_check_benchmark_tic:
381     \__tag_tree_write_structtreeroot:
382     \__tag_check_benchmark_toc:
383   }
384 }
385 \end{package}

```

(End of definition for __tag_finish_structure:.)

1.10 StructParents entry for Page

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

```

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

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

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

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

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

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

New: 2019-11-20

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

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

New: 2021-04-22

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

<code>\tag_mc_if_in_p: *</code>	<code>\tag_mc_if_in:TF {<true code>} {<false code>}</code>
<code>\tag_mc_if_in:TF *</code>	

Determines if a mc-chunk is open.

<code>\tag_mc_reset_box:N</code> ★	<code>\tag_mc_reset_box:N</code> $\langle box \rangle$
------------------------------------	--

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

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

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.

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

New: 2024-11-18	This declares the interface needed to handle a new stream with <code>\tag_mc_add_missing_to_stream:Nn</code> . Predeclared are the names <code>main</code> , <code>footnote</code> and <code>multicol</code> .
-----------------	--

2 Public keys

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

<code>tag</code> (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).
---------------------------	---

<code>artifact</code> (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 <code>pagination</code> , <code>layout</code> , <code>page</code> , <code>background</code> and <code>notype</code> (this is the default).
--------------------------------	---

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

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

<code>actualtext</code> (mc-key)	This key inserts an <code>/ActualText</code> 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)	<p>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.</p> <p>The code is split into three parts: code shared by all engines, code specific to luamode and code not used by luamode.</p>
-----------------------	---

3 Marked content code – shared

```

1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-mc-code-shared} {2025-05-16} {0.99q}
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__tag_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 cname 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 77.)

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

`\tag_mc_artifact_group_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 77.)

\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 78.)

\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 77.)

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

```

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

```

This records if logging is on

```

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

```

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

```

156     \int_compare:nNnT {\l__tag_parent_child_check_tl} = { \c__tag_role_rule_checkparent_tl }
157     {
158         \__tag_struct_get_role:nnNN

```

```

159         {#1}
160         {parentrole}
161         \l__tag_get_parent_tmpa_tl
162         \l__tag_get_parent_tmppb_tl
163         \__tag_role_check_parent_child:ooooN
164         { \l__tag_get_parent_tmpa_tl }
165         { \l__tag_get_parent_tmppb_tl }
166         { MC } %
167         { } %
168         \l__tag_parent_child_check_tl
169     }
170     \__tag_check_forbidden_parent_child:nnee
171     {\l__tag_parent_child_check_tl}
172     {#1}
173     {
174         \l__tag_get_parent_tmppb_tl : \l__tag_get_parent_tmpa_tl
175     }
176     {
177         MC~(real content)
178     }
179 }
180 \cs_generate_variant:Nn \__tag_mc_check_parent_child:n {o}

```

(End of definition for __tag_mc_check_parent_child:n.)

3.3 Keys

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

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

```

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

```

```

202  __artifact-type / page      .code:n    =
203  {
204      \tl_set:Nn \l__tag_mc_artifact_type_tl { Page }
205  },
206  __artifact-type / background .code:n    =
207  {
208      \tl_set:Nn \l__tag_mc_artifact_type_tl { Background }
209  },
210  __artifact-type / notype     .code:n    =
211  {
212      \tl_set:Nn \l__tag_mc_artifact_type_tl {}
213  },
214  __artifact-type /           .code:n    =
215  {
216      \tl_set:Nn \l__tag_mc_artifact_type_tl {}
217  },
218  }

```

(End of definition for `stash (mc-key)`, `__artifact-bool`, and `__artifact-type`. This function is documented on page 79.)

219 `</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} {2025-05-16} {0.99q}
4 {part of tagpdf - code related to marking chunks - generic mode}
5 </generic>
6 <*debug>
7 \ProvidesExplPackage {tagpdf-debug-generic} {2025-05-16} {0.99q}
8 {part of tagpdf - debugging code related to marking chunks - generic mode}
9 </debug>
```

1.1 Variables

```
10 <*generic>
```

`\l__tag_mc_ref_abspace_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_abspace_tl
```

(End of definition for `\l__tag_mc_ref_abspace_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 78.)

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

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

```

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

```

(End of definition for __tag_mc_disable_marks:.)

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

```

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

```

(End of definition for __tag_mc_get_marks:.)

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

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

```

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

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

```

108 \cs_new_protected:Npn \__tag_mc_insert_extra_tmb:n #1 % #1 stream: e.g. main or footnote
109 {
110   \__tag_check_typeout_v:n {>~ first~ \seq_use:Nn \l__tag_mc_firstmarks_seq {,~}}
111   \__tag_check_typeout_v:n {>~ bot~ \seq_use:Nn \l__tag_mc_botmarks_seq {,~}}
112   \__tag_check_if_mc_tmb_missing:TF
113   {
114     \__tag_check_typeout_v:n {>~ TMB~ ~ missing~ --- inserted}
115     %test if artifact
116     \int_compare:nNnTF { \seq_item:cn { g__tag_mc_#1_marks_seq } {3} } = {-
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       }
132     }
133   }
134 }

```

```

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 78.)

`_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:` 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.
`\tag_mc_if_in:TF`

```

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 77.)

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

```

234 % #1 tag, #2 properties
235 \cs_set_eq:NN \_tag_mc_bmc:n \pdf_bmc:n
236 \cs_set_eq:NN \_tag_mc_emc: \pdf_emc:
237 \cs_set_eq:NN \_tag_mc_bdc:nn \pdf_bdc:nn
238 \cs_set_eq:NN \_tag_mc_bdc_shipout:ee \pdf_bdc_shipout:ee

```

(End of definition for `_tag_mc_bmc:n`, `_tag_mc_emc:`, and `_tag_mc_bdc:nn`.)

`_tag_mc_bdc_mcid:nn` 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.
`_tag_mc_bdc_mcid:n`
`_tag_mc_handle_mcid:nn`
`_tag_mc_handle_mcid:oo`

```

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

```

```

253     }
254 }
255 \cs_new_protected:Npn \__tag_mc_bdc_mcid:n #1
256 {
257     \__tag_mc_bdc_mcid:nn {#1} {}
258 }
259
260 \cs_new_protected:Npn \__tag_mc_handle_mcid:nn #1 #2 % #1 tag, #2 properties
261 {
262     \__tag_mc_bdc_mcid:nn {#1} {#2}
263 }
264
265 \cs_generate_variant:Nn \__tag_mc_handle_mcid:nn {oo}

```

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

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

```

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

```

(End of definition for __tag_mc_handle_stash:n.)

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

```

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

```

(End of definition for `__tag_mc_bmc_artifact:`, `__tag_mc_bmc_artifact:n`, and `__tag_mc_handle_artifact:N`.)

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

```
293 \cs_new:Nn __tag_get_data_mc_tag: { \g__tag_mc_key_tag_tl }
294 \</generic>
```

(End of definition for `__tag_get_data_mc_tag:.`)

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

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

set default MC tags to structure:

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

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

```

333         \bool_if:NF \l__tag_mc_key_stash_bool
334         {
335             \socket_use:nn{tag/check/parent-child}
336             {
337                 \__tag_mc_check_parent_child:o
338                 { \g__tag_struct_stack_current_tl }
339             }
340             \__tag_mc_handle_stash:e { \int_use:N \c@g__tag_MCID_abs_int }
341         }
342     }
343 }
344 \group_end:
345 }
346 <*debug>
347 {
348     \__tag_debug_mc_begin_ignore:n { #1 }
349 }
350 </debug>
351 }
352 <*generic>
353 \cs_set_protected:Nn \tag_mc_end:
354 {
355     \__tag_check_if_active_mc:T
356     {
357 </generic>
358 <*debug>
359 \cs_set_protected:Nn \tag_mc_end:
360 {
361     \__tag_check_if_active_mc:TF
362     {
363         \__tag_debug_mc_end_insert:
364 </debug>
365         \__tag_check_mc_if_open:
366         \bool_gset_false:N \g__tag_in_mc_bool
367         \tl_gset:Nn \g__tag_mc_key_tag_tl { }
368         \__tag_mc_emc:
369         \__tag_mc_end_marks:
370     }
371 <*debug>
372 {
373     \__tag_debug_mc_end_ignore:
374 }
375 </debug>
376 }
377 </generic | debug>

```

(End of definition for \tag_mc_begin:n and \tag_mc_end:. These functions are documented on page 77.)

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) 378 <*generic>
alt (mc-key) 379 \keys_define:nn { __tag / mc }
actualtext (mc-key) 380 {
label (mc-key) 381   tag .code:n = % the name (H,P,Span) etc
artifact (mc-key) 382   {
383     \tl_set:Nc \l__tag_mc_key_tag_tl { #1 }
384     \tl_gset:Nc \g__tag_mc_key_tag_tl { #1 }
385   },
386   raw .code:n =
387   {
388     \tl_put_right:Nc \l__tag_mc_key_properties_tl { #1 }
389   },
390   alt .code:n = % Alt property
391   {
392     \str_set_convert:Nc
393       \l__tag_tmpa_str
394       { #1 }
395       { default }
396       { utf16/hex }
397     \tl_put_right:Nc \l__tag_mc_key_properties_tl { /Alt~< }
398     \tl_put_right:Nc \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
399   },
400   alttext .meta:n = {alt=#1},
401   actualtext .code:n = % ActualText property
402   {
403     \tl_if_empty:oF{#1}
404     {
405       \str_set_convert:Nc
406         \l__tag_tmpa_str
407         { #1 }
408         { default }
409         { utf16/hex }
410       \tl_put_right:Nc \l__tag_mc_key_properties_tl { /ActualText~< }
411       \tl_put_right:Nc \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
412     }
413   },
414   label .tl_set:N = \l__tag_mc_key_label_tl,
415   artifact .code:n =
416   {
417     \exp_args:Nne
418     \keys_set:nn
419       { __tag / mc }
420       { __artifact-bool, __artifact-type=#1 }
421   },
422   artifact .default:n = {notype}
423 }
424 </generic>

```

(End of definition for tag (mc-key) and others. These functions are documented on page 78.)

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

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

`tag` : the type (a string)

`raw` : more properties (string)

`label`: a string.

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

`kids`: a array of tables

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

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

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

```
1 <@@=tag>
2 <*luamode>
3 \ProvidesExplPackage {tagpdf-mc-code-lua} {2025-05-16} {0.99q}
4   {tagpdf - mc code only for the luamode }
5 </luamode>
6 <*debug>
7 \ProvidesExplPackage {tagpdf-debug-lua} {2025-05-16} {0.99q}
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 78.)

`_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 77.)

`_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:N\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 <debug>     MC~#1%
157 <debug>   }
158   \lua_now:e
159   {
160     ltx.__tag.func.store_struct_mcab
161     (
162       \g__tag_struct_stack_current_tl,#1
163     )
164   }
165 }
166 </luamode | debug>
167 <*luamode>
168 \cs_generate_variant:Nn \__tag_mc_handle_stash:n { e }

```

(End of definition for __tag_mc_handle_stash:n.)

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

```

169 \cs_set_protected:Nn \tag_mc_begin:n
170 {
171   \__tag_check_if_active_mc:T
172   {
173     \group_begin:
174     %\__tag_check_mc_if_nested:
175     \bool_gset_true:N \g__tag_in_mc_bool
176     \bool_set_false:N \l__tag_mc_artifact_bool
177     \tl_clear:N \l__tag_mc_key_properties_tl
178     \int_gincr:N \c@g__tag_MCID_abs_int

```

set the default tag to the structure:

```

179     \tl_set_eq:NN \l__tag_mc_key_tag_tl \g__tag_struct_tag_tl
180     \tl_gset_eq:NN \g__tag_mc_key_tag_tl \g__tag_struct_tag_tl
181     \lua_now:e
182     {
183       ltx.__tag.func.store_mc_data(\__tag_get_mc_abs_cnt:,"tag","\g__tag_struct_tag_tl"
184     }
185     \keys_set:nn { __tag / mc }{ label={}, #1 }
186     %check that a tag or artifact has been used
187     \__tag_check_mc_tag:N \l__tag_mc_key_tag_tl
188     %set the attributes:
189     \__tag_mc_lua_set_mc_type_attr:o { \l__tag_mc_key_tag_tl }

```

```

190 \bool_if:NF \l__tag_mc_artifact_bool
191 { % store the absolute num name in a label:
192   \tl_if_empty:NF {\l__tag_mc_key_label_tl}
193   {
194     \__tag_mc_handle_mc_label:e { \l__tag_mc_key_label_tl }
195   }
196   % if not stashed record the absolute number
197   \bool_if:NF \l__tag_mc_key_stash_bool
198   {
199     \socket_use:nn{tag/check/parent-child}
200     {
201       \__tag_mc_check_parent_child:o
202       { \g__tag_struct_stack_current_tl }
203     }
204     \__tag_mc_handle_stash:e { \__tag_get_mc_abs_cnt: }
205   }
206 }
207 \group_end:
208 }
209 }

```

(End of definition for \tag_mc_begin:n. This function is documented on page 77.)

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

```

210 \cs_set_protected:Nn \tag_mc_end:
211 {
212   \__tag_check_if_active_mc:T
213   {
214     %\__tag_check_mc_if_open:
215     \bool_gset_false:N \g__tag_in_mc_bool
216     \bool_set_false:N \l__tag_mc_artifact_bool
217     \__tag_mc_lua_unset_mc_type_attr:
218     \tl_set:Nn \l__tag_mc_key_tag_tl { }
219     \tl_gset:Nn \g__tag_mc_key_tag_tl { }
220   }
221 }

```

(End of definition for \tag_mc_end:. This function is documented on page 77.)

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

```

222 \cs_set_protected:Npn \tag_mc_reset_box:N #1
223 {
224   \lua_now:e
225   {
226     local~type=tex.getattribute(luatexbase.attributes.g__tag_mc_type_attr)
227     local~mc=tex.getattribute(luatexbase.attributes.g__tag_mc_cnt_attr)
228     ltx.__tag.func.update_mc_attributes(tex.getbox(\int_use:N #1),mc,type)
229   }
230 }

```

(End of definition for \tag_mc_reset_box:N. This function is documented on page 78.)

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

```

231 \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)  232 \keys_define:nn { __tag / mc }
alt (mc-key)  233 {
actualtext (mc-key) 234   tag .code:n = %
label (mc-key)  235   {
artifact (mc-key) 236     \tl_set:Nc \l__tag_mc_key_tag_tl { #1 }
                237     \tl_gset:Nc \g__tag_mc_key_tag_tl { #1 }
                238     \lua_now:e
                239     {
                240       ltx.__tag.func.store_mc_data(\__tag_get_mc_abs_cnt:,"tag","#1")
                241     }
                242   },
                243   raw .code:n =
                244   {
                245     \tl_put_right:Nc \l__tag_mc_key_properties_tl { #1 }
                246     \lua_now:e
                247     {
                248       ltx.__tag.func.store_mc_data(\__tag_get_mc_abs_cnt:,"raw","#1")
                249     }
                250   },
                251   alt .code:n      = % Alt property
                252   {
                253     \tl_if_empty:oF{#1}
                254     {
                255       \str_set_convert:Nc
                256       \l__tag_tmpa_str
                257       { #1 }
                258       { default }
                259       { utf16/hex }
                260       \tl_put_right:Nc \l__tag_mc_key_properties_tl { /Alt~< }
                261       \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
                262       \lua_now:e
                263       {
                264         ltx.__tag.func.store_mc_data
                265         (
                266           \__tag_get_mc_abs_cnt:,"alt","/Alt~<\str_use:N \l__tag_tmpa_str>"
                267         )
                268       }
                269     }
                270   },
                271   alttext .meta:n = {alt=#1},
                272   actualtext .code:n      = % Alt property
                273   {
                274     \tl_if_empty:oF{#1}
                275     {
                276       \str_set_convert:Nc
                277       \l__tag_tmpa_str
                278       { #1 }
                279       { default }
                280       { utf16/hex }
```



```

281         \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
282         \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
283         \lua_now:e
284         {
285             ltx.__tag.func.store_mc_data
286             (
287                 \__tag_get_mc_abs_cnt:,
288                 "actualtext",
289                 "/ActualText~<\str_use:N \l__tag_tmpa_str>"
290             )
291         }
292     }
293 },
294 label .code:n =
295 {
296     \tl_set:Nn\l__tag_mc_key_label_tl { #1 }
297     \lua_now:e
298     {
299         ltx.__tag.func.store_mc_data
300         (
301             \__tag_get_mc_abs_cnt:,"label","#1"
302         )
303     }
304 },
305 __artifact-store .code:n =
306 {
307     \lua_now:e
308     {
309         ltx.__tag.func.store_mc_data
310         (
311             \__tag_get_mc_abs_cnt:,"artifact","#1"
312         )
313     }
314 },
315 artifact .code:n      =
316 {
317     \exp_args:Nne
318     \keys_set:nn
319     { __tag / mc}
320     { __artifact-bool, __artifact-type=#1, tag=Artifact }
321     \exp_args:Nne
322     \keys_set:nn
323     { __tag / mc }
324     { __artifact-store=\l__tag_mc_artifact_type_tl }
325 },
326 artifact .default:n    = { notype }
327 }
328
329 </luamode>

```

(End of definition for tag (mc-key) and others. These functions are documented on page 78.)

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.

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

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

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

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

2 Public keys

2.1 Keys for the structure commands

- tag** (*struct key*) This is required. The value of the key is normally one of the standard types listed in the main tagpdf documentation. It is possible to setup new tags/types. The value can also be of the form `type/NS`, where NS is the shorthand of a declared name space. Currently the names spaces `pdf`, `pdf2`, `mathml` and `user` are defined. This allows to use a different name space than the one connected by default to the tag. But normally this should not be needed.
- stash** (*struct key*) Normally a new structure inserts itself as a kid into the currently active structure. This key prohibits this. The structure is nevertheless from now on “the current active structure” and parent for following marked content and structures.
- label** (*struct key*) This key sets a label by which one can refer to the structure. It is e.g. used by `\tag_struct_use:n` (where a real label is actually not needed as you can only use structures already defined), and by the `ref` key (which can refer to future structures). Internally the label name will start with `tagpdfstruct-` and it stores the two attributes `tagstruct` (the structure number) and `tagstructobj` (the object reference).
- parent** (*struct key*) By default a structure is added as kid to the currently active structure. With the parent key one can choose another parent. The value is a structure number which must refer to an already existing, previously created structure. Such a structure number can for example be have been stored with `\tag_get:n`, but one can also use a label on the parent structure and then use `\property_ref:nn{tagpdfstruct-label}{tagstruct}` to retrieve it.
- firstkid** (*struct key*) If this key is used the structure is added at the left of the kids of the parent structure (if the structure is not stashed). This means that it will be the first kid of the structure (unless some later structure uses the key too).
- title** (*struct key*) This keys allows to set the dictionary entry `/Title` in the structure object. The value
- title-o** (*struct key*) is handled as verbatim string and hex encoded. Commands are not expanded. `title-o` will expand the value once.

- alt** (*struct key*) This key inserts an `/Alt` value in the dictionary of structure object. The value is handled as verbatim string and hex encoded. The value will be expanded first once. If it is empty, nothing will happen.
- actualtext** (*struct key*) This key inserts an `/ActualText` value in the dictionary of structure object. The value is handled as verbatim string and hex encoded. The value will be expanded first once. If it is empty, nothing will happen.
- lang** (*struct key*) This key allows to set the language for a structure element. The value should be a bcp-identifier, e.g. `de-De`.
- ref** (*struct key*) This key allows to add references to other structure elements, it adds the `/Ref` array to the structure. The value should be a comma separated list of structure labels set with the `label` key. e.g. `ref={label1,label2}`.
- E** (*struct key*) This key sets the `/E` key, the expanded form of an abbreviation or an acronym (I couldn't think of a better name, so I stucked to E).
- AF** (*struct key*) These keys handle associated files in the structure element.
- AFref** (*struct key*)
- AFinline** (*struct key*) **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} {2025-05-16} {0.99q}
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>\int_new:N \c@g__tag_struct_abs_int
7 <base>\int_gset:Nn \c@g__tag_struct_abs_int { 1 }
```

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

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

```
8 <*package>
9 \__tag_seq_new:N \g__tag_struct_objR_seq
```

(End of definition for `\g__tag_struct_objR_seq`.)

`\c__tag_struct_null_tl` In lua mode we have to test if the kids a null
`10 \tl_const:Nn\c__tag_struct_null_tl {null}`

(End of definition for `\c__tag_struct_null_tl`.)

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

`11 __tag_prop_new:N \g__tag_struct_cont_mc_prop`

(End of definition for `\g__tag_struct_cont_mc_prop`.)

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

`12 \seq_new:N \g__tag_struct_stack_seq`
`13 \seq_gpush:Nn \g__tag_struct_stack_seq {1}`

(End of definition for `\g__tag_struct_stack_seq`.)

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

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

(End of definition for `\g__tag_struct_tag_stack_seq`.)

`\g__tag_struct_stack_current_tl` The global variable will hold the current structure number. It is already defined in `tagpdf-base`.
`\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
\l__tag_struct_parenttag_tl
\l__tag_struct_parenttag_NS_tl

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

```
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
62 \tl_new:N \l__tag_struct_parenttag_tl
```

```

63 \tl_set:Nn \l__tag_struct_parenttag_tl {STASHED}
64 \tl_new:N \l__tag_struct_parenttag_NS_tl
65 \tl_set:Nn \l__tag_struct_parenttag_NS_tl {\latex}

```

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

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

```

66 \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

```

67 \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**.

```

68 \tl_new:N \l__tag_struct_addkid_tl
69 \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.

```

70 \</package>
71 \<base>\prop_new_linked:N \g__tag_struct_dest_num_prop
72 \<*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.

```

73 \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
74 \cs_new:Npn \__tag_struct_output_prop_aux:nn #1 #2 %#1 num, #2 key
75 {
76   \prop_if_in:cnT
77     { g__tag_struct_#1_prop }
78     { #2 }
79   {
80     \c_space_tl/#2~ \prop_item:cn{ g__tag_struct_#1_prop } { #2 }
81   }
82 }
83
84 \cs_new_protected:Npn \__tag_new_output_prop_handler:n #1
85 {
86   \cs_new:cn { __tag_struct_output_prop_#1:n }
87   {
88     \__tag_struct_output_prop_aux:nn {#1}{##1}
89   }
90 }
91 \</package>

```

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

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

```

92 \<package| debug>
93 \<package>\cs_new_protected:Npn \__tag_struct_prop_gput:nnn #1 #2 #3
94 \<debug>\cs_set_protected:Npn \__tag_struct_prop_gput:nnn #1 #2 #3
95 {
96   \__tag_prop_gput:cnn
97     { g__tag_struct_#1_prop }{#2}{#3}
98 \<debug>\prop_gput:cnn { g__tag_struct_debug_#1_prop } {#2} {#3}
99 }
100 \cs_generate_variant:Nn \__tag_struct_prop_gput:nnn {onn,nne,nee,nnn}
101 \</package| debug>

```

(End of definition for __tag_struct_prop_gput:nnn.)

4.1 Initialization of the StructTreeRoot

The first structure element, the StructTreeRoot is special, so created manually. The underlying object is @@/struct/1 which is currently created in the tree code (TODO move it here). The ParentTree and RoleMap entries are added at begin document in the tree code as they refer to object which are setup in other parts of the code. This avoid timing issues.

```

102 \<package>
103 \tl_gset:Nn \g__tag_struct_stack_current_tl {1}

\__tag_pdf_name_e:n
104 \cs_new:Npn \__tag_pdf_name_e:n #1{\pdf_name_from_unicode_e:n{#1}}
105 \</package>

```

(End of definition for __tag_pdf_name_e:n.)

```

g__tag_struct_1_prop
g__tag_struct_kids_1_seq
106 <*package>
107 \__tag_prop_new:c { g__tag_struct_1_prop }
108 \__tag_new_output_prop_handler:n {1}
109 \__tag_seq_new:c { g__tag_struct_kids_1_seq }
110
111 \__tag_struct_prop_gput:nne
112 { 1 }
113 { Type }
114 { \pdf_name_from_unicode_e:n {StructTreeRoot} }
115
116 \__tag_struct_prop_gput:nne
117 { 1 }
118 { S }
119 { \pdf_name_from_unicode_e:n {StructTreeRoot} }
120
121 \__tag_struct_prop_gput:nne
122 { 1 }
123 { tag }
124 { {StructTreeRoot}{pdf} }
125
126 \__tag_struct_prop_gput:nne
127 { 1 }
128 { rolemap }
129 { {StructTreeRoot}{pdf} }
130
131 \__tag_struct_prop_gput:nne
132 { 1 }
133 { parentrole }
134 { {StructTreeRoot}{pdf} }
135

```

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

```

136 \pdf_version_compare:NnF < {2.0}
137 {
138   \__tag_struct_prop_gput:nne
139   { 1 }
140   { Namespaces }
141   { \pdf_object_ref:n { __tag/tree/namespaces } }
142 }
143 </package>

```

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

```

144 <debug>\prop_new:c { g__tag_struct_debug_1_prop }
145 <debug>\seq_new:c { g__tag_struct_debug_kids_1_seq }
146 <debug>\prop_gset_eq:cc { g__tag_struct_debug_1_prop }{ g__tag_struct_1_prop }
147 <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`

```

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

```

(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.

```

159 \pdf_version_compare:NnTF < {2.0}
160 {
161   \cs_new_protected:Npn __tag_struct_set_tag_info:nnn #1 #2 #3
162     %#1 structure number, #2 tag, #3 NS
163   {
164     __tag_struct_prop_gput:nne
165       { #1 }
166       { S }
167       { \pdf_name_from_unicode_e:n {#2} } %
168     __tag_struct_prop_gput:nnn
169       { #1 }
170       { tag }
171       { {#2} {} }
172   }
173 }
174 {
175   \cs_new_protected:Npn __tag_struct_set_tag_info:nnn #1 #2 #3
176   {
177     __tag_struct_prop_gput:nne
178       { #1 }
179       { S }
180       { \pdf_name_from_unicode_e:n {#2} } %
181     \prop_get:NnNT \g__tag_role_NS_prop {#3} \l__tag_get_tmpc_tl
182     {
183       __tag_struct_prop_gput:nne
184         { #1 }
185         { NS }
186         { \l__tag_get_tmpc_tl } %
187     }
188     __tag_struct_prop_gput:nnn
189       { #1 }
190       { tag }
191       { {#2} {#3} }
192   }
193 }

```

```
194 \cs_generate_variant:Nn \__tag_struct_set_tag_info:nnn {eoo}
```

(End of definition for __tag_struct_set_tag_info:nnn.)

```
\__tag_struct_get_role:nnNN
```

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

```
195 \cs_new_protected:Npn \__tag_struct_get_role:nnNN #1 #2 #3 #4
196   %#1 :struct num,
197   %#2 :rolemap or parentrole
198   %#3 :tlvar for tag (rolemapped)
199   %#4 :tlvar for NS (rolemapped, so standard or empty or UNKNOWN)
200   {
201     \prop_get:cnNTF
202     { g__tag_struct_#1_prop }
203     { #2 }
204     \l__tag_get_tmpc_tl
205     {
206       \tl_set:Ne #3{\exp_last_unbraced:No\use_i:nn { \l__tag_get_tmpc_tl }}
207       \tl_set:Ne #4{\exp_last_unbraced:No\use_ii:nn { \l__tag_get_tmpc_tl }}
208     }
209     {
210       \tl_clear:N#3
211       \tl_clear:N#4
212     }
213   }
214 \cs_generate_variant:Nn \__tag_struct_get_role:nnNN {enNN}
```

(End of definition for __tag_struct_get_role:nnNN.)

4.4 Handlings kids

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

```
\__tag_struct_kid_mc_gput_right:nn
\__tag_struct_kid_mc_gput_right:ne
```

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

```
215 \cs_new:Npn \__tag_struct_mcid_dict:n #1 %#1 MCID absnum
216   {
217     <<
218     /Type \c_space_tl /MCR \c_space_tl
219     /Pg
220     \c_space_tl
221     \pdf_pageobject_ref:n { \property_ref:enn{mcid-#1}{tagabspage}{1} }
222     /MCID \c_space_tl \property_ref:enn{mcid-#1}{tagmcid}{1}
223     >>
224   }
225 (/package)
```

```

226 <*package| debug>
227 <package>\cs_new_protected:Npn \__tag_struct_kid_mc_gput_right:nn #1 #2
228 <debug>\cs_set_protected:Npn \__tag_struct_kid_mc_gput_right:nn #1 #2
229 %%#1 structure num, #2 MCID absnum%
230 {
231   \__tag_seq_gput_right:ce
232   { g__tag_struct_kids_#1_seq }
233   {
234     \__tag_struct_mcid_dict:n {#2}
235   }
236 <debug>   \seq_gput_right:cn
237 <debug>   { g__tag_struct_debug_kids_#1_seq }
238 <debug>   {
239     MC~#2
240 <debug>   }
241   \__tag_seq_gput_right:cn
242   { g__tag_struct_kids_#1_seq }
243   {
244     \prop_item:Nn \g__tag_struct_cont_mc_prop {#2}
245   }
246 }
247 <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.

```

248 <package>\cs_new_protected:Npn \__tag_struct_kid_struct_gput_right:nn #1 #2
249 <debug>\cs_set_protected:Npn \__tag_struct_kid_struct_gput_right:nn #1 #2
250 %%#1 num of parent struct, #2 kid struct
251 {
252   \__tag_seq_gput_right:ce
253   { g__tag_struct_kids_#1_seq }
254   {
255     \pdf_object_ref_indexed:nn { __tag/struct }{ #2 }
256   }
257 <debug>   \seq_gput_right:cn
258 <debug>   { g__tag_struct_debug_kids_#1_seq }
259 <debug>   {
260     Struct~#2
261 <debug>   }
262 }
263 <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.

```

264 <package>\cs_new_protected:Npn \__tag_struct_kid_struct_gput_left:nn #1 #2
265 <debug>\cs_set_protected:Npn \__tag_struct_kid_struct_gput_left:nn #1 #2
266 %%#1 num of parent struct, #2 kid struct
267 {
268   \__tag_seq_gput_left:ce
269   { g__tag_struct_kids_#1_seq }
270   {

```

```

271     \pdf_object_ref_indexed:nn { __tag/struct }{ #2 }
272   }
273   <debug> \seq_gput_left:cn
274   <debug> { g__tag_struct_debug_kids_#1_seq }
275   <debug> {
276     <debug> Struct~#2
277   <debug> }
278   }
279   <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

```

280 <package>\cs_new_protected:Npn \__tag_struct_kid_OBJR_gput_right:nnn #1 #2 #3
281 <package>
282 <package>
283 <debug>\cs_set_protected:Npn \__tag_struct_kid_OBJR_gput_right:nnn #1 #2 #3
284 %%#1 num of parent struct,#2 obj reference,#3 page object reference
285 {
286   \pdf_object_unnamed_write:nn
287   { dict }
288   {
289     /Type/OBJR/Obj~#2/Pg~#3
290   }
291   \__tag_seq_gput_right:ce
292   { g__tag_struct_kids_#1_seq }
293   {
294     \pdf_object_ref_last:
295   }
296   <debug> \seq_gput_right:ce
297   <debug> { g__tag_struct_debug_kids_#1_seq }
298   <debug> {
299     <debug> OBJR~reference
300   <debug> }
301   }
302 </package|debug>
303 <*package>
304 \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.

```

305 \cs_new_protected:Npn \__tag_struct_exchange_kid_command:N #1 %%#1 = seq var
306 {
307   \seq_gpop_left:NN #1 \l__tag_tmpa_tl
308   \tl_replace_once:Nnn \l__tag_tmpa_tl
309   { \__tag_mc_insert_mcid_kids:n }
310   { \__tag_mc_insert_mcid_single_kids:n }
311   \seq_gput_left:No #1 { \l__tag_tmpa_tl }

```

```

312 }
313
314 \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.

```

315 \cs_new_protected:Npn \__tag_struct_fill_kid_key:n #1 % #1 is the struct num
316 {
317   \bool_if:NF \g__tag_mode_lua_bool
318   {
319     \seq_clear:N \l__tag_tmpa_seq
320     \seq_map_inline:cn { g__tag_struct_kids_#1_seq }
321     { \seq_put_right:Ne \l__tag_tmpa_seq { ##1 } }
322     %\seq_show:c { g__tag_struct_kids_#1_seq }
323     %\seq_show:N \l__tag_tmpa_seq
324     \seq_remove_all:Nn \l__tag_tmpa_seq {}
325     %\seq_show:N \l__tag_tmpa_seq
326     \seq_gset_eq:cN { g__tag_struct_kids_#1_seq } \l__tag_tmpa_seq
327   }
328
329   \int_case:nnF
330   {
331     \seq_count:c
332     {
333       g__tag_struct_kids_#1_seq
334     }
335   }
336   {
337     { 0 }
338     { } %no kids, do nothing
339     { 1 } % 1 kid, insert
340     {
341       % in this case we need a special command in
342       % luamode to get the array right. See issue #13
343       \sys_if_engine luatex:TF
344       {
345         \__tag_struct_exchange_kid_command:c
346         {g__tag_struct_kids_#1_seq}

```

check if we get null

```

347     \tl_set:N\l__tag_tmpa_tl
348     {\use:ef\seq_item:cn {g__tag_struct_kids_#1_seq} {1}}}
349     \tl_if_eq:NNF\l__tag_tmpa_tl \c__tag_struct_null_tl
350     {
351       \__tag_struct_prop_gput:nne
352       {#1}
353       {K}
354       {
355         \seq_item:cn
356         {
357           g__tag_struct_kids_#1_seq
358         }

```

```

359         {1}
360     }
361 }
362 }
363 {
364     \_tag_struct_prop_gput:nne
365     {#1}
366     {K}
367     {
368         \seq_item:cn
369         {
370             g__tag_struct_kids_#1_seq
371         }
372         {1}
373     }
374 }
375 } %
376 }
377 { %many kids, use an array
378     \_tag_struct_prop_gput:nne
379     {#1}
380     {K}
381     {
382         [
383             \seq_use:cn
384             {
385                 g__tag_struct_kids_#1_seq
386             }
387             {
388                 \c_space_tl
389             }
390         ]
391     }
392 }
393 }
394

```

(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.

```

395 \cs_new_protected:Npn \_tag_struct_get_dict_content:nN #1 #2 %#1: structure num
396 {
397     \tl_clear:N #2
398     \prop_map_inline:cn { g__tag_struct_#1_prop }
399     {

```

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.

```

400     \cs_if_exist_use:cTF {\_tag_struct_format_##1:nnN}
401     {
402         {##1}{##2}#2

```



```

403     }
404     {
405         \tl_put_right:Ne #2 { \c_space_tl/##1~##2 }
406     }
407 }
408 }

```

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

```

\__tag_struct_format_rolemap:nnN
\__tag_struct_format_parentrole:nnN
\__tag_struct_format_P:nnN
\__tag_struct_format_tag:nnN

```

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

```

409 \cs_new:Nn\__tag_struct_format_rolemap:nnN{}
410 \cs_new:Nn\__tag_struct_format_parentrole:nnN{}
411 \cs_new:Nn\__tag_struct_format_tag:nnN{}

```

(End of definition for `__tag_struct_format_rolemap:nnN` and others.)

```
\__tag_struct_format_parentnum:nnN
```

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

```

412 \cs_new_protected:Nn\__tag_struct_format_parentnum:nnN
413 {
414     \tl_put_right:Ne #3 { ~/P~\pdf_object_ref_indexed:nn { __tag/struct} { #2 } }
415 }

```

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

```
\__tag_struct_format_Ref:nnN
```

Ref is an array, we store values as a list of commands that must be executed here, the formatting has to add also brackets.

```

416 \cs_new_protected:Nn\__tag_struct_format_Ref:nnN
417 {
418     \tl_put_right:Nn #3 { ~/#1~[ ] %]
419     \clist_map_inline:nn{ #2 }
420     {
421         ##1 #3
422     }
423     \tl_put_right:Nn #3
424     { %[
425         \c_space_tl]
426     }
427 }

```

(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.

```

428 \cs_new_protected:Npn \__tag_struct_write_obj:n #1 % #1 is the struct num
429 {
430     \prop_if_exist:cTF { g__tag_struct_#1_prop }
431     {

```

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.

```

432     \prop_get:cnNF { g__tag_struct_#1_prop } {parentnum}\l__tag_tmpb_tl
433     {
434     %         \prop_gput:cne { g__tag_struct_#1_prop } {P}
435     %         {\pdf_object_ref_indexed:nn { __tag/struct }{1}}

```

```

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

```

(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:

```

```

462 \cs_new_protected:Npn \__tag_struct_insert_annot:nn #1 #2
463   {%#1 object reference to the annotation/xform
464   %#2 structparent number

```

```

465 {
466   \bool_if:NT \g__tag_active_struct_bool
467   {
468     %get the number of the parent structure:
469     \seq_get:NNF
470     \g__tag_struct_stack_seq
471     \l__tag_struct_stack_parent_tmpa_tl
472     {
473       \msg_error:nn { tag } { struct-faulty-nesting }
474     }
475     %put the obj number of the annot in the kid entry, this also creates
476     %the OBJR object
477     \__tag_property_record:nn {@tag@objr@page@#2 }{ tagabspage }
478     \__tag_struct_kid_OBJR_gput_right:eee
479     {
480       \l__tag_struct_stack_parent_tmpa_tl
481     }
482     {
483       #1 %
484     }
485     {
486       \pdf_pageobject_ref:n
487       { \property_ref:nnn {@tag@objr@page@#2 }{ tagabspage }{1} }
488     }
489     % add the parent obj number to the parent tree:
490     \exp_args:Nne
491     \__tag_parenttree_add_objr:nn
492     {
493       #2
494     }
495     {
496       \pdf_object_ref_indexed:nn
497       { __tag/struct }{ \l__tag_struct_stack_parent_tmpa_tl }
498     }
499     % increase the int:
500     \int_gincr:N \c@g__tag_parenttree_obj_int
501   }
502 }

```

(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**.

```

503 \cs_new:Npn \__tag_get_data_struct_tag:
504 {
505   \exp_args:Ne
506   \tl_tail:n
507   {
508     \prop_item:cn {g__tag_struct_\g__tag_struct_stack_current_tl _prop}{S}
509   }
510 }

```

(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`.

```

511 \cs_new:Npn \__tag_get_data_struct_id:
512 {
513   \__tag_struct_get_id:n {\g__tag_struct_stack_current_tl}
514 }
515 \endpackage

```

(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

```

516 \begin{base}
517 \cs_new:Npn \__tag_get_data_struct_num:
518 {
519   \g__tag_struct_stack_current_tl
520 }
521 \end{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.

```

522 \begin{base}
523 \cs_new:Npn \__tag_get_data_struct_counter:
524 {
525   \int_use:N \c@g__tag_struct_abs_int
526 }
527 \end{base}

```

(End of definition for `__tag_get_data_struct_counter:.`)

4.6 Commands for the parent-child checks

`__tag_struct_check_parent_child_aux:nnnnN`

```

528 \begin{package}
529 \cs_new_protected:Npn \__tag_struct_check_parent_child_aux:nnnnN #1#2#3#4#5
530 {
531   % #1 structure number of parent
532   % #2 key to use to retrieve role of parent (either rolemap or parentrole field)
533   % #3 structure number of parent
534   % #4 key to use to retrieve role of child (either rolemap or parentrole field)
535   % #5 tl for return value

```

get parent rolemap

```

536   \__tag_struct_get_role:nnNN
537   {#1}
538   {#2}
539   \l__tag_get_parent_tmpa_tl
540   \l__tag_get_parent_tmpb_tl

```

get child rolemap

```

541 \__tag_struct_get_role:nnNN
542 {#3}
543 {#4}
544 \l__tag_get_child_tmpa_tl
545 \l__tag_get_child_tmpb_tl

```

check

```

546 \__tag_role_check_parent_child:ooooN
547 { \l__tag_get_parent_tmpa_tl } % rolemapped from above
548 { \l__tag_get_parent_tmpb_tl } % rolemapped from above
549 { \l__tag_get_child_tmpa_tl } %
550 { \l__tag_get_child_tmpb_tl } %
551 #5
552 }

```

(End of definition for __tag_struct_check_parent_child_aux:nnnnN.)

__tag_struct_check_parent_child:nn

When comparing the relation between structures we use the structure numbers.

```

553 \cs_new_protected:Npn \__tag_struct_check_parent_child:nn #1 #2
554 % #1 structure number of parent
555 % #2 structure number of child. %
556 % This assumes that the fields rolemap/parentrole has already been filled.
557 {

```

This records if logging is on

```

558 \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
559 {
560 \prop_get:cnN{g__tag_struct_#1_prop}{tag}\l__tag_get_parent_tmpa_tl
561 \prop_get:cnN{g__tag_struct_#2_prop}{tag}\l__tag_get_parent_tmpb_tl
562 \msg_note:nnee
563 { tag }
564 { role-parent-child-check }
565 {
566 \quark_if_no_value:NTF \l__tag_get_parent_tmpa_tl
567 {??}
568 {
569 \exp_last_unbraced:No\use_ii:nn
570 { \l__tag_get_parent_tmpa_tl }
571 :
572 \exp_last_unbraced:No\use_i:nn
573 { \l__tag_get_parent_tmpa_tl }
574 }
575 }
576 {
577 \quark_if_no_value:NTF \l__tag_get_parent_tmpb_tl
578 {??}
579 {
580 \exp_last_unbraced:No\use_ii:nn
581 { \l__tag_get_parent_tmpb_tl }
582 :
583 \exp_last_unbraced:No\use_i:nn
584 { \l__tag_get_parent_tmpb_tl }
585 }
586 }

```

```

587     }
588     \_tag_struct_check_parent_child_aux:nnnnN
589     {#1}
590     {rolemap}
591     {#2}
592     {rolemap}
593     \l__tag_parent_child_check_tl

```

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

```

594     \int_compare:nNnT {\l__tag_parent_child_check_tl} = { \c__tag_role_rule_checkparent_tl }
595     {
596         \_tag_struct_check_parent_child_aux:nnnnN
597         {#1}
598         {parentrole}
599         {#2}
600         {rolemap}
601         \l__tag_parent_child_check_tl
602     }
603     \_tag_check_struct_forbidden_parent_child:onn
604     {\l__tag_parent_child_check_tl}
605     {#1}
606     {#2}
607 }
608 \cs_generate_variant:Nn \_tag_struct_check_parent_child:nn {oo}

```

(End of definition for _tag_struct_check_parent_child:nn.)

_tag_struct_use_check_parent_child:nn

A similar command is needed if a structure is stashed and used. The child can be - a normal tag (e.g. H1) then rolemap = parentrole = H1pdf2 and we should test rolemap (parent) and rolemap (child) if = 7 parentrole (parent) and rolemap (child) That is the normal check above.

- Part/Div/Nonstruct then rolemap = Partpdf2 and parentrole = STASHEDlatex or target parentNS

If parentrole =STASHED we can't test if the child fits here. If parentrole is not STASHED, then would should test if target parent= rolemap (parent) or parentrole (parent) and if yet then test rolemap (child) against rolemap (parent) and if =7 rolemap(child) against parentrole(parent). that is again the normal check.

```

609 \cs_new_protected:Npn \_tag_struct_use_check_parent_child:nn #1 #2
610 % #1 structure number of parent
611 % #2 structure number of child. %
612 {
613     \_tag_struct_get_role:enNN
614     {#2}
615     {rolemap}
616     \l__tag_get_child_tmpa_tl
617     \l__tag_get_child_tmpb_tl
618     \str_case:onTF {\l__tag_get_child_tmpa_tl }
619     {
620         {Part} {}
621         {Div} {}
622         {NonStruct} {}
623     }
624     { %child=Part etc
625         \_tag_struct_get_role:enNN

```

```

626     {#2}
627     {parentrole}
628     \l__tag_get_child_tmpa_tl
629     \l__tag_get_child_tmpb_tl
630     \str_if_eq:noTF
631     {STASHED}{\l__tag_get_child_tmpa_tl}
632     {
633       % warn about unknown relationship
634     }
635     {
636       % test if
637       \__tag_struct_get_role:enNN
638       {#1}
639       {parentrole}
640       \l__tag_get_parent_tmpa_tl
641       \l__tag_get_parent_tmpb_tl
642       \tl_if_eq:NNTF\l__tag_get_parent_tmpa_tl \l__tag_get_child_tmpa_tl
643       {
644         \__tag_struct_check_parent_child:nn {#1}{#2}
645       }
646       {
647         %warn that parent-tag was misused.
648       }
649     }
650   }
651   {
652     %child not Part etc, normal parent child test.
653     \__tag_struct_check_parent_child:nn {#1}{#2}
654   }
655 }
656 \cs_generate_variant:Nn { \__tag_struct_use_check_parent_child:nn }{oo}

```

(End of definition for __tag_struct_use_check_parent_child:nn.)

5 Keys

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

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

```

657 \socket_new:nn { tag/struct/tag }{1}
658 \socket_new_plug:nnn { tag/struct/tag }{ latex-tags }
659 {
660   \prop_get:NeNTF \g__tag_role_tags_NS_prop {#1} \l__tag_tmp_unused_tl
661   {
662     \seq_set_split:Nne \l__tag_tmpa_seq { / }
663     {#1/\l__tag_tmp_unused_tl}
664   }
665   {
666     \seq_set_split:Nne \l__tag_tmpa_seq { / }
667     {#1/}
668   }
669   \tl_gset:Ne \g__tag_struct_tag_tl { \seq_item:Nn\l__tag_tmpa_seq {1} }

```

```

670 \tl_gset:Ne \g__tag_struct_tag_NS_tl{ \seq_item:Nn\l__tag_tmpa_seq {2} }
671 \__tag_check_structure_tag:N \g__tag_struct_tag_tl
672 }
673
674 \socket_new_plug:nnn { tag/struct/tag }{ pdf-tags }
675 {
676   \prop_get:NeNTF \g__tag_role_tags_NS_prop {#1} \l__tag_tmp_unused_tl
677   {
678     \seq_set_split:Nne \l__tag_tmpa_seq { / }
679     {#1/\l__tag_tmp_unused_tl}
680   }
681   {
682     \seq_set_split:Nne \l__tag_tmpa_seq { / }
683     {#1/}
684   }
685   \tl_gset:Ne \g__tag_struct_tag_tl { \seq_item:Nn\l__tag_tmpa_seq {1} }
686   \tl_gset:Ne \g__tag_struct_tag_NS_tl{ \seq_item:Nn\l__tag_tmpa_seq {2} }
687   \__tag_role_get:ooNN
688   { \g__tag_struct_tag_tl }
689   { \g__tag_struct_tag_NS_tl}
690   \l__tag_tmpa_tl
691   \l__tag_tmpb_tl
692   \tl_gset:Ne \g__tag_struct_tag_tl {\l__tag_tmpa_tl}
693   \tl_gset:Ne \g__tag_struct_tag_NS_tl{\l__tag_tmpb_tl}
694   \__tag_check_structure_tag:N \g__tag_struct_tag_tl
695 }
696 \socket_assign_plug:nn { tag/struct/tag } {latex-tags}

```

```

label (struct key)
stash (struct key) 697 \keys_define:nn { __tag / struct }
parent (struct key) 698 {
firstkid (struct key) 699   label .code:n      =
tag (struct key) 700   {
title (struct key) 701     \prop_gput:Nee\g__tag_struct_label_num_prop
title-o (struct key) 702     {#1}{\int_use:N \c@g__tag_struct_abs_int}
alt (struct key) 703     \__tag_property_record:eo
actualtext (struct key) 704     {tagpdfstruct-#1}
lang (struct key) 705     { \c__tag_property_struct_clist }
ref (struct key) 706   },
E (struct key) 707   stash .bool_set:N    = \l__tag_struct_elem_stash_bool,
phoneme (struct key) 708   parent .code:n      =
709   {
710     \bool_lazy_and:nnTF
711     {
712       \prop_if_exist_p:c { g__tag_struct\_int_eval:n {#1}_prop }
713     }
714     {
715       \int_compare_p:nNn {#1}<{\c@g__tag_struct_abs_int}
716     }
717     { \tl_set:Ne \l__tag_struct_stack_parent_tmpa_tl { \int_eval:n {#1} } }
718     {
719       \msg_warning:nnee { tag } { struct-unknown }
720       { \int_eval:n {#1} }
721       { parent~key~ignored }

```



```

722     }
723 },
724 parent .default:n      = {-1},
725 parent-tag .code:n =
726 {
727     \prop_get:NeNTF \g__tag_role_tags_NS_prop {#1} \l__tag_tmp_unused_tl
728     {
729         \seq_set_split:Nne \l__tag_tmpa_seq { / }
730         {#1/\l__tag_tmp_unused_tl}
731     }
732     {
733         \seq_set_split:Nne \l__tag_tmpa_seq { / }
734         {#1/}
735     }
736     \tl_set:Ne \l__tag_struct_parenttag_tl { \seq_item:Nn\l__tag_tmpa_seq {1} }
737     \tl_set:Ne \l__tag_struct_parenttag_NS_tl{ \seq_item:Nn\l__tag_tmpa_seq {2} }
738     \__tag_role_get:ooNN
739     { \l__tag_struct_parenttag_tl }
740     { \l__tag_struct_parenttag_NS_tl}
741     \l__tag_tmpa_tl
742     \l__tag_tmpb_tl
743     \tl_set:No \l__tag_struct_parenttag_tl { \l__tag_tmpa_tl}
744     \tl_set:No \l__tag_struct_parenttag_NS_tl{ \l__tag_tmpb_tl}
745     \__tag_check_structure_tag:N \l__tag_struct_parenttag_tl
746 },
747 firstkid .code:n = { \tl_set:Nn \l__tag_struct_addkid_tl {left} },
748 tag .code:n      = % S property
749 {
750     \socket_use:nn { tag/struct/tag }{#1}
751 },
752 title .code:n    = % T property
753 {
754     \str_set_convert:Nnnn
755     \l__tag_tmpa_str
756     { #1 }
757     { default }
758     { utf16/hex }
759     \__tag_struct_prop_gput:nne
760     { \int_use:N \c@g__tag_struct_abs_int }
761     { T }
762     { <\l__tag_tmpa_str> }
763 },
764 title-o .code:n  = % T property
765 {
766     \str_set_convert:Nonn
767     \l__tag_tmpa_str
768     { #1 }
769     { default }
770     { utf16/hex }
771     \__tag_struct_prop_gput:nne
772     { \int_use:N \c@g__tag_struct_abs_int }
773     { T }
774     { <\l__tag_tmpa_str> }
775 },

```

```

776 alt .code:n      = % Alt property
777 {
778     \tl_if_empty:oF{#1}
779     {
780         \str_set_convert:Noon
781         \l__tag_tmpa_str
782         { #1 }
783         { default }
784         { utf16/hex }
785         \__tag_struct_prop_gput:nne
786         { \int_use:N \c@g__tag_struct_abs_int }
787         { Alt }
788         { <\l__tag_tmpa_str> }
789     }
790 },
791 alttext .meta:n = {alt=#1},
792 actualtext .code:n = % ActualText property
793 {
794     \tl_if_empty:oF{#1}
795     {
796         \str_set_convert:Noon
797         \l__tag_tmpa_str
798         { #1 }
799         { default }
800         { utf16/hex }
801         \__tag_struct_prop_gput:nne
802         { \int_use:N \c@g__tag_struct_abs_int }
803         { ActualText }
804         { <\l__tag_tmpa_str>}
805     }
806 },
807 phoneme .code:n = % Phoneme property
808 {
809     \tl_if_empty:oF{#1}
810     {
811         \str_set_convert:Noon
812         \l__tag_tmpa_str
813         { #1 }
814         { default }
815         { utf16/hex }
816         \__tag_struct_prop_gput:nne
817         { \int_use:N \c@g__tag_struct_abs_int }
818         { Phoneme }
819         { <\l__tag_tmpa_str>}
820     }
821 },
822 lang .code:n      = % Lang property
823 {
824     \__tag_struct_prop_gput:nne
825     { \int_use:N \c@g__tag_struct_abs_int }
826     { Lang }
827     { (#1) }
828 },
829 }

```

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` this command is a helper command that is 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

```

830 \cs_new_protected:Npn \__tag_struct_Ref_obj:nN #1 #2 %#1 a object reference
831 {
832   \tl_put_right:Ne#2
833   {
834     \c_space_tl#1
835   }
836 }
837
838 \cs_new_protected:Npn \__tag_struct_Ref_label:nN #1 #2 %#1 a label
839 {
840   \prop_get:NnNTF \g__tag_struct_label_num_prop {#1} \l__tag_tmpb_tl
841   {
842     \tl_put_right:Ne#2
843     {
844       \c_space_tl\tag_struct_object_ref:e{ \l__tag_tmpb_tl }
845     }
846   }
847   {
848     \msg_warning:nnn {tag}{struct-Ref-unknown}{Label~'#1'}
849   }
850 }
851 \cs_new_protected:Npn \__tag_struct_Ref_dest:nN #1 #2 %#1 a dest name
852 {
853   \prop_get:NnNTF \g__tag_struct_dest_num_prop {#1} \l__tag_tmpb_tl
854   {
855     \tl_put_right:Ne#2
856     {
857       \c_space_tl\tag_struct_object_ref:e{ \l__tag_tmpb_tl }
858     }
859   }
860   {
861     \msg_warning:nnn {tag}{struct-Ref-unknown}{Destination~'#1'}
862   }
863 }
864 \cs_new_protected:Npn \__tag_struct_Ref_num:nN #1 #2 %#1 a structure number
865 {
866   \tl_put_right:Ne#2
867   {
868     \c_space_tl\tag_struct_object_ref:e{ #1 }
869   }
870 }
871

```

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

ref (struct key)
E (struct key)

```

872 \keys_define:nn { __tag / struct }
873 {
874   ref .code:n          = % ref property
875   {
876     \clist_map_inline:on {#1}
877     {
878       \tag_struct_gput:nne
879       { \int_use:N \c@g__tag_struct_abs_int } { ref_label } { ##1 }
880     }
881   },
882   E .code:n          = % E property
883   {
884     \str_set_convert:Nnon
885     \l__tag_tmpa_str
886     { #1 }
887     { default }
888     { utf16/hex }
889     \__tag_struct_prop_gput:nne
890     { \int_use:N \c@g__tag_struct_abs_int }
891     { E }
892     { <\l__tag_tmpa_str> }
893   },
894 }

```

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

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

```

895 \int_new:N \g__tag_struct_AFobj_int
(End of definition for \g__tag_struct_AFobj_int.)

896 \cs_generate_variant:Nn \pdf_file_embed_stream:nnN {neN}
897 \cs_new_protected:Npn \__tag_struct_add_inline_AF:nn #1 #2
898 % #1 content, #2 extension
899 {
900   \tl_if_empty:nF{#1}
901   {
902     \group_begin:
903     \int_gincr:N \g__tag_struct_AFobj_int
904     \pdf_file_embed_stream:neN
905     {#1}
906     {tag-AFfile\int_use:N \g__tag_struct_AFobj_int.#2}
907     \l__tag_tmpa_tl
908     \__tag_struct_add_AF:ee
909     { \int_use:N \c@g__tag_struct_abs_int }
910     { \l__tag_tmpa_tl }
911     \__tag_struct_prop_gput:nne
912     { \int_use:N \c@g__tag_struct_abs_int }
913     { AF }
914   }

```

```

915         [
916             \tl_use:c
917             { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
918         ]
919     }
920     \group_end:
921 }
922 }
923
924 \cs_generate_variant:Nn \__tag_struct_add_inline_AF:nn {on}
925 \cs_new_protected:Npn \__tag_struct_add_AF:nn #1 #2
926 % #1 struct num #2 object reference
927 {
928     \tl_if_exist:cTF
929     {
930         g__tag_struct_#1_AF_tl
931     }
932     {
933         \tl_gput_right:ce
934         { g__tag_struct_#1_AF_tl }
935         { \c_space_tl #2 }
936     }
937     {
938         \tl_new:c
939         { g__tag_struct_#1_AF_tl }
940         \tl_gset:ce
941         { g__tag_struct_#1_AF_tl }
942         { #2 }
943     }
944 }
945 \cs_generate_variant:Nn \__tag_struct_add_AF:nn {en,ee}
946 \keys_define:nn { __tag / struct }
947 {
948     AF .code:n          = % AF property
949     {
950         \pdf_object_if_exist:eTF {#1}
951         {
952             \__tag_struct_add_AF:ee
953             { \int_use:N \c@g__tag_struct_abs_int }{\pdf_object_ref:e {#1}}
954             \__tag_struct_prop_gput:nne
955             { \int_use:N \c@g__tag_struct_abs_int }
956             { AF }
957             {
958                 [
959                     \tl_use:c
960                     { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
961                 ]
962             }
963         }
964         {
965             % message?
966         }
967     },
968     AFref .code:n      = % AF property

```

```

969 {
970   \tl_if_empty:eF {#1}
971   {
972     \__tag_struct_add_AF:ee { \int_use:N \c@g__tag_struct_abs_int }{#1}
973     \__tag_struct_prop_gput:nne
974       { \int_use:N \c@g__tag_struct_abs_int }
975       { AF }
976       {
977         [
978           \tl_use:c
979           { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
980         ]
981       }
982     }
983   },
984   ,AFinline .code:n =
985   {
986     \__tag_struct_add_inline_AF:nn {#1}{txt}
987   }
988   ,AFinline-o .code:n =
989   {
990     \__tag_struct_add_inline_AF:on {#1}{txt}
991   }
992   ,texsource .code:n =
993   {
994     \group_begin:
995     \pdfdict_put:nnn { l_pdffile/Filespec } {Desc}{(TeX~source)}
996     \pdfdict_put:nnn { l_pdffile/Filespec }{AFRelationship} { /Source }
997     \__tag_struct_add_inline_AF:on {#1}{tex}
998     \group_end:
999   }
1000   ,mathml .code:n =
1001   {
1002     \group_begin:
1003     \pdfdict_put:nnn { l_pdffile/Filespec } {Desc}{(mathml~representation)}
1004     \pdfdict_put:nnn { l_pdffile/Filespec }{AFRelationship} { /Supplement }
1005     \pdfdict_put:nne { l_pdffile }{Subtype}
1006       { \pdf_name_from_unicode_e:n{application/mathml+xml} }
1007     \__tag_struct_add_inline_AF:on {#1}{xml}
1008     \group_end:
1009   }
1010 }

```

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!

```

1011 \keys_define:nn { __tag / setup }
1012 {
1013   root-AF .code:n =
1014   {
1015     \pdf_object_if_exist:nTF {#1}
1016     {
1017       \__tag_struct_add_AF:ee { 1 }{\pdf_object_ref:n {#1}}
1018       \__tag_struct_prop_gput:nne
1019       { 1 }

```

```

1020         { AF }
1021     {
1022         [
1023             \tl_use:c
1024             { g__tag_struct_1_AF_tl }
1025         ]
1026     }
1027 }
1028 {
1029
1030 }
1031 },
1032 }

```

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

```

1033 \keys_define:nn { __tag / setup }
1034 {
1035     root-supplemental-file .code:n =
1036     {
1037         \group_begin:
1038         \pdfdict_put:nnn {l_pdffile/Filespec} {AFRelationship}{/Supplement}
1039         \int_gincr:N \g__tag_unique_cnt_int
1040         \pdffile_embed_file:eee
1041         {#1}
1042         {#1}
1043         {__tag_latex_css_\int_use:N\g__tag_unique_cnt_int}
1044         \keys_set:nn
1045         {__tag / setup}
1046         {root-AF={__tag_latex_css_\int_use:N\g__tag_unique_cnt_int}}
1047         \group_end:
1048     }
1049 }

```

6 User commands

We allow to set a language by default

`\l__tag_struct_lang_tl`

```

1050 \tl_new:N \l__tag_struct_lang_tl
1051 \</package>

```

(End of definition for `\l__tag_struct_lang_tl`.)

`\tag_struct_begin:n`

`\tag_struct_end:`

```

1052 <base>\cs_new_protected:Npn \tag_struct_begin:n #1 {\int_gincr:N \c@g__tag_struct_abs_int}
1053 <base>\cs_new_protected:Npn \tag_struct_end:{}
1054 <base>\cs_new_protected:Npn \tag_struct_end:n{
1055 <*package| debug>
1056 <package>\cs_set_protected:Npn \tag_struct_begin:n #1 %#1 key-val
1057 <debug>\cs_set_protected:Npn \tag_struct_begin:n #1 %#1 key-val
1058 {
1059 <package>\__tag_check_if_active_struct:T

```

```

1060 <debug>\__tag_check_if_active_struct:TF
1061     {
1062         \group_begin:
1063         \int_gincr:N \c@g__tag_struct_abs_int
1064         \__tag_prop_new:c { g__tag_struct\_int_eval:n { \c@g__tag_struct_abs_int }_prop }
1065 <debug>         \prop_new:c { g__tag_struct_debug\_int_eval:n { \c@g__tag_struct_abs_int }_prop }
1066         \__tag_new_output_prop_handler:n { \int_eval:n { \c@g__tag_struct_abs_int } }
1067         \__tag_seq_new:c { g__tag_struct_kids\_int_eval:n { \c@g__tag_struct_abs_int }_seq }
1068 <debug>         \seq_new:c { g__tag_struct_debug_kids\_int_eval:n { \c@g__tag_struct_abs_int }_seq }
1069         \pdf_object_new_indexed:nn { __tag/struct }
1070         { \c@g__tag_struct_abs_int }
1071         \__tag_struct_prop_gput:nnn
1072         { \int_use:N \c@g__tag_struct_abs_int }
1073         { Type }
1074         { /StructElem }
1075         \tl_if_empty:NF \l__tag_struct_lang_tl
1076         {
1077             \__tag_struct_prop_gput:nne
1078             { \int_use:N \c@g__tag_struct_abs_int }
1079             { Lang }
1080             { (\l__tag_struct_lang_tl) }
1081         }
1082         \__tag_struct_prop_gput:nnn
1083         { \int_use:N \c@g__tag_struct_abs_int }
1084         { Type }
1085         { /StructElem }
1086
1087         \tl_set:Nn \l__tag_struct_stack_parent_tmpa_tl {-1}
1088         \keys_set:nn { __tag / struct } { #1 }
1089
1090         \__tag_struct_set_tag_info:eoo
1091         { \int_use:N \c@g__tag_struct_abs_int }
1092         { \g__tag_struct_tag_tl }
1093         { \g__tag_struct_tag_NS_tl }
1094         \__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.

```

1094         \int_compare:nNnT { \l__tag_struct_stack_parent_tmpa_tl } = { -1 }
1095         {
1096             \seq_get:NNF
1097             \g__tag_struct_stack_seq
1098             \l__tag_struct_stack_parent_tmpa_tl
1099             {
1100                 \msg_error:nn { tag } { struct-faulty-nesting }
1101             }
1102         }
1103         \seq_gpush:NV \g__tag_struct_stack_seq \c@g__tag_struct_abs_int
1104         \__tag_role_get:ooNN
1105         { \g__tag_struct_tag_tl }
1106         { \g__tag_struct_tag_NS_tl }
1107         \l__tag_struct_roletag_tl
1108         \l__tag_struct_roletag_NS_tl

```

We push the role tag on the stack:


```

1109     \seq_gpush:Ne \g__tag_struct_tag_stack_seq
1110     {{\g__tag_struct_tag_t1}{\l__tag_struct_roletag_t1}}
1111     \tl_gset:NV \g__tag_struct_stack_current_t1 \c@g__tag_struct_abs_int
1112     %\seq_show:N \g__tag_struct_stack_seq

```

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

```

1113     \__tag_struct_prop_gput:nne
1114     { \int_use:N \c@g__tag_struct_abs_int }
1115     { rolemap }
1116     {
1117         {\l__tag_struct_roletag_t1}{\l__tag_struct_roletag_NS_t1}
1118     }

```

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

```

1119     \str_case:onTF { \l__tag_struct_roletag_t1 }
1120     {
1121         {Part} {}
1122         {Div} {}
1123         {NonStruct} {}
1124     }
1125     {
1126         \bool_if:NTF \l__tag_struct_elem_stash_bool
1127         {
1128             \__tag_struct_prop_gput:nne
1129             { \int_use:N \c@g__tag_struct_abs_int }
1130             { parentrole }
1131             {
1132                 {\l__tag_struct_parenttag_t1}{\l__tag_struct_parenttag_NS_t1}
1133             }
1134         }
1135         {
1136             \prop_get:cnNT
1137             { g__tag_struct_ \l__tag_struct_stack_parent_tmpa_t1 _prop }
1138             { parentrole }
1139             \l__tag_get_tmpc_t1
1140             {
1141                 \__tag_struct_prop_gput:nno
1142                 { \int_use:N \c@g__tag_struct_abs_int }
1143                 { parentrole }
1144                 {
1145                     \l__tag_get_tmpc_t1
1146                 }
1147             }
1148         }
1149     }
1150     {
1151         \__tag_struct_prop_gput:nne
1152         { \int_use:N \c@g__tag_struct_abs_int }
1153         { parentrole }

```

```

1154         {
1155             {\l__tag_struct_roletag_t1}{\l__tag_struct_roletag_NS_t1}
1156         }
1157     }
1158     \bool_if:NF
1159     \l__tag_struct_elem_stash_bool
1160     {

```

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.

```

1161         \socket_use:nn{tag/check/parent-child}
1162         {
1163             \__tag_struct_check_parent_child:oo
1164             { \l__tag_struct_stack_parent_tmpa_t1 }
1165             { \int_use:N \c@g__tag_struct_abs_int }
1166         }

```

Set the Parent structure number.

```

1167         \__tag_struct_prop_gput:nne
1168         { \int_use:N \c@g__tag_struct_abs_int }
1169         { parentnum }
1170         {
1171             \l__tag_struct_stack_parent_tmpa_t1
1172         }
1173         %record this structure as kid:
1174         %\tl_show:N \g__tag_struct_stack_current_t1
1175         %\tl_show:N \l__tag_struct_stack_parent_tmpa_t1
1176         \use:c { __tag_struct_kid_struct_gput_ \l__tag_struct_addkid_t1 :ee }
1177         { \l__tag_struct_stack_parent_tmpa_t1 }
1178         { \g__tag_struct_stack_current_t1 }
1179         %\prop_show:c { g__tag_struct_\g__tag_struct_stack_current_t1 _prop }
1180         %\seq_show:c {g__tag_struct_kids_\l__tag_struct_stack_parent_tmpa_t1 _seq}
1181     }

```

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.

```

1182 <debug>         \prop_gset_eq:cc
1183 <debug>         { g__tag_struct_debug_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
1184 <debug>         { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
1185 <debug>         \prop_gput:cne
1186 <debug>         { g__tag_struct_debug_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
1187 <debug>         { parentnum }
1188 <debug>         {
1189 <debug>             \bool_if:NTF \l__tag_struct_elem_stash_bool
1190 <debug>             {no~parent:~stashed}
1191 <debug>             {
1192 <debug>                 \l__tag_struct_stack_parent_tmpa_t1\c_space_t1 =~
1193 <debug>                 \prop_item:cn{ g__tag_struct_\l__tag_struct_stack_parent_tmpa_t1 _p
1194 <debug>             }
1195 <debug>         }
1196 <debug>         \prop_gput:cne
1197 <debug>         { g__tag_struct_debug_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
1198 <debug>         { NS }

```

```

1199 <debug>                { \g__tag_struct_tag_NS_t1 }
1200                %\prop_show:c { g__tag_struct_\g__tag_struct_stack_current_t1 _prop }
1201                %\seq_show:c {g__tag_struct_kids_\l__tag_struct_stack_parent_tmpa_t1 _seq}
1202 <debug> \__tag_debug_struct_begin_insert:n { #1 }
1203                \group_end:
1204        }
1205 <debug>{ \__tag_debug_struct_begin_ignore:n { #1 }}
1206        }
1207 <package>\cs_set_protected:Nn \tag_struct_end:
1208 <debug>\cs_set_protected:Nn \tag_struct_end:
1209 { %take the current structure num from the stack:
1210   %the objects are written later, lua mode hasn't all needed info yet
1211   %\seq_show:N \g__tag_struct_stack_seq
1212 <package>\__tag_check_if_active_struct:T
1213 <debug>\__tag_check_if_active_struct:TF
1214 {
1215     \seq_gpop:NN    \g__tag_struct_tag_stack_seq \l__tag_tmpa_t1
1216     \seq_gpop:NNTF \g__tag_struct_stack_seq \l__tag_tmpa_t1
1217     {
1218         \__tag_check_info_closing_struct:o { \g__tag_struct_stack_current_t1 }
1219     }
1220     { \__tag_check_no_open_struct: }
1221     % get the previous one, shouldn't be empty as the root should be there
1222     \seq_get:NNTF \g__tag_struct_stack_seq \l__tag_tmpa_t1
1223     {
1224         \tl_gset:No \g__tag_struct_stack_current_t1 { \l__tag_tmpa_t1 }
1225     }
1226     {
1227         \__tag_check_no_open_struct:
1228     }
1229     \seq_get:NNT \g__tag_struct_tag_stack_seq \l__tag_tmpa_t1
1230     {
1231         \tl_gset:Ne \g__tag_struct_tag_t1
1232         { \exp_last_unbraced:No\use_i:nn { \l__tag_tmpa_t1 } }
1233         \prop_get:NoNT\g__tag_role_tags_NS_prop { \g__tag_struct_tag_t1} \l__tag_tmpa_t1
1234         {
1235             \tl_gset:Ne \g__tag_struct_tag_NS_t1 { \l__tag_tmpa_t1 }
1236         }
1237     }
1238 <debug>\__tag_debug_struct_end_insert:
1239        }
1240 <debug>{\__tag_debug_struct_end_ignore:}
1241        }
1242
1243 \cs_set_protected:Npn \tag_struct_end:n #1
1244 {
1245 <debug>    \__tag_check_if_active_struct:T{\__tag_debug_struct_end_check:n{#1}}
1246    \tag_struct_end:
1247 }
1248 </package | debug>

```

(End of definition for \tag_struct_begin:n and \tag_struct_end:. These functions are documented on page 106.)

`\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.

```

1249 <base>\cs_new_protected:Npn \tag_struct_use:n #1 {}
1250 <*package|debug>
1251 \cs_set_protected:Npn \tag_struct_use:n #1 %#1 is the label
1252 {
1253   \__tag_check_if_active_struct:T
1254   {
1255     \prop_if_exist:cTF
1256     { g__tag_struct_\property_ref:enn{tagpdfstruct-#1}{tagstruct}{unknown}_prop } %
1257     {
1258       \__tag_check_struct_used:n {#1}
1259       \tl_set:Nx \l__tag_get_child_tmpa_tl
1260       { \property_ref:enn{tagpdfstruct-#1}{tagstruct}{1} }

```

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

```

1261       \__tag_struct_kid_struct_gput_right:ee
1262       { g__tag_struct_stack_current_tl }
1263       { \l__tag_get_child_tmpa_tl }

```

add the current structure to the labeled one as parents

```

1264       \__tag_prop_gput:cne
1265       { g__tag_struct_\l__tag_get_child_tmpa_tl _prop }
1266       { parentnum }
1267       {
1268         \g__tag_struct_stack_current_tl
1269       }

```

debug code

```

1270 <debug>          \prop_gput:cne
1271 <debug>          { g__tag_struct_debug_\l__tag_get_child_tmpa_tl _prop }
1272 <debug>          { parentnum }
1273 <debug>          {
1274 <debug>          \g__tag_struct_stack_current_tl\c_space_tl=~
1275 <debug>          \g__tag_struct_tag_tl
1276 <debug>          }

```

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

```

1277       \socket_use:nn{tag/check/parent-child}
1278       {
1279         \__tag_struct_use_check_parent_child:oo
1280         { g__tag_struct_stack_current_tl }
1281         { \l__tag_get_child_tmpa_tl }
1282       }
1283     }
1284   {
1285     \msg_warning:nnn{ tag }{struct-label-unknown}{#1}
1286   }
1287 }
1288 }
1289 </package|debug>

```

(End of definition for \tag_struct_use:n. This function is documented on page 106.)

\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.

```

1290 <base>\cs_new_protected:Npn \tag_struct_use_num:n #1 {}
1291 <*package|debug>
1292 \cs_set_protected:Npn \tag_struct_use_num:n #1 %#1 is structure number
1293 {
1294   \__tag_check_if_active_struct:T
1295   {
1296     \prop_if_exist:cTF
1297     { g__tag_struct_#1_prop } %
1298     {
1299       \prop_get:cnNT
1300       {g__tag_struct_#1_prop}
1301       {parentnum}
1302       \l__tag_tmpa_tl
1303       {
1304         \msg_warning:nnn { tag } {struct-used-twice} {#1}
1305       }
1306     }

```

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

```

1306   \__tag_struct_kid_struct_gput_right:ee
1307   { \g__tag_struct_stack_current_tl }
1308   { #1 }

```

add the current structure to #1 as parent

```

1309   \__tag_struct_prop_gput:nne
1310   { #1 }
1311   { parentnum }
1312   {
1313     \g__tag_struct_stack_current_tl
1314   }
1315 <debug> \prop_gput:cne
1316 <debug> { g__tag_struct_debug_#1_prop }
1317 <debug> { parentnum }
1318 <debug> {
1319 <debug> \g__tag_struct_stack_current_tl\c_space_tl=~
1320 <debug> \g__tag_struct_tag_tl
1321 <debug> }

```

check if the tag is allowed as child.

```

1322   \socket_use:nn{tag/check/parent-child}
1323   {
1324     \__tag_struct_use_check_parent_child:oo
1325     {\g__tag_struct_stack_current_tl}
1326     {#1}
1327   }
1328 }
1329 {
1330   \msg_warning:nnn{ tag }{struct-label-unknown}{#1}
1331 }
1332 }
1333 }

```

```
1334 </package | debug>
```

(End of definition for `\tag_struct_use_num:n`. This function is documented on page 106.)

`\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.

```
1335 <*package>
1336 \cs_new:Npn \tag_struct_object_ref:n #1
1337 {
1338   \pdf_object_ref_indexed:nn {__tag/struct}{ #1 }
1339 }
1340 \cs_generate_variant:Nn \tag_struct_object_ref:n {e}
1341 </package>
```

(End of definition for `\tag_struct_object_ref:n`. This function is documented on page 106.)

`\tag_struct_gput:nnn` This is a command that allows to update the data of a structure. This often can't done simply by replacing the value, as we have to preserve and extend existing content. We use therefore dedicated functions adjusted to the key in question. The first argument is the number of the structure, the second a keyword referring to a function, the third the value. Currently the existing keywords are mostly related to the Ref key (an array). The keyword `ref` takes as value an explicit object reference to a structure. The keyword `ref_label` expects as value a label name (from a label set in a `\tagstructbegin` command). The keyword `ref_dest` expects a destination name set with `\MakeLinkTarget`. It then will refer to the structure in which this `\MakeLinkTarget` was used. The keyword `ref_num` expects a structure number. At last there is the keyword `attribute` which allows to add or extend the /A key of the structure. The value is the content of one attribute dictionary, so for example /O /Layout /BBox [10 10 50 50]. The content is stored in an object and the object reference is then added to the /A.

```
1342 <base>\cs_new_protected:Npn \tag_struct_gput:nnn #1 #2 #3{
1343 <*package>
1344 \cs_set_protected:Npn \tag_struct_gput:nnn #1 #2 #3
1345 {
1346   \cs_if_exist_use:cF {__tag_struct_gput_data_#2:nn}
1347   { %warning??
1348     \use_none:nn
1349   }
1350   {#1}{#3}
1351 }
1352 \cs_generate_variant:Nn \tag_struct_gput:nnn {ene,nne}
1353 </package>
```

(End of definition for `\tag_struct_gput:nnn`. This function is documented on page 107.)

`__tag_struct_gput_data_ref_aux:nnn`

```
1354 <*package>
1355 \cs_new_protected:Npn __tag_struct_gput_data_ref_aux:nnn #1 #2 #3
1356 % #1 receiving struct num, #2 key word #3 value
1357 {
1358   \prop_get:cnNTF
1359   { g__tag_struct_#1_prop }
1360   {Ref}
```

```

1361     \l__tag_get_tmpc_tl
1362     {
1363         \tl_put_right:No \l__tag_get_tmpc_tl
1364         {\cs:w __tag_struct_Ref_#2:nN \cs_end: {#3},}
1365     }
1366     {
1367         \tl_set:No \l__tag_get_tmpc_tl
1368         {\cs:w __tag_struct_Ref_#2:nN \cs_end: {#3},}
1369     }
1370     \__tag_struct_prop_gput:nno
1371     { #1 }
1372     { Ref }
1373     { \l__tag_get_tmpc_tl }
1374 }
1375 \cs_new_protected:Npn \__tag_struct_gput_data_ref:nn #1 #2
1376 {
1377     \__tag_struct_gput_data_ref_aux:nnn {#1}{obj}{#2}
1378 }
1379 \cs_new_protected:Npn \__tag_struct_gput_data_ref_label:nn #1 #2
1380 {
1381     \__tag_struct_gput_data_ref_aux:nnn {#1}{label}{#2}
1382 }
1383 \cs_new_protected:Npn \__tag_struct_gput_data_ref_dest:nn #1 #2
1384 {
1385     \__tag_struct_gput_data_ref_aux:nnn {#1}{dest}{#2}
1386 }
1387 \cs_new_protected:Npn \__tag_struct_gput_data_ref_num:nn #1 #2
1388 {
1389     \__tag_struct_gput_data_ref_aux:nnn {#1}{num}{#2}
1390 }
1391
1392 \cs_generate_variant:Nn \__tag_struct_gput_data_ref:nn {ee,no}
(End of definition for \__tag_struct_gput_data_ref_aux:nnn.)

\__tag_struct_gput_data_attribute:nn
1393 \cs_new_protected:Npn \__tag_struct_gput_data_attribute:nn #1 #2
1394 {
1395     \pdf_object_unnamed_write:nn {dict} {#2}
1396     \prop_get:cnNTF { g__tag_struct_#1_prop }{A} \l__tag_tmpa_tl
1397     {
1398         \tl_remove_once:Nn\l__tag_tmpa_tl{[]}
1399         \tl_remove_once:Nn\l__tag_tmpa_tl{[] }
1400         \__tag_prop_gput:cne { g__tag_struct_#1_prop }
1401         { A }
1402         {
1403             [ \l__tag_tmpa_tl \c_space_tl \pdf_object_ref_last: ]
1404         }
1405     }
1406     {
1407         \__tag_prop_gput:cne { g__tag_struct_#1_prop }
1408         { A }
1409         { \pdf_object_ref_last: }
1410     }
1411 }

```

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

```

\tag_struct_insert_annot:nn This are the user command to insert annotations. They must be used together to get the
\tag_struct_insert_annot:ee numbers right. They use a counter to the StructParent and \tag_struct_insert_-
\tag_struct_insert_annot:ee annot:nn increases the counter given back by \tag_struct_parent_int:.
\tag_struct_parent_int: It must be used together with \tag_struct_parent_int: to insert an annotation.
                          TODO: decide how it should be guarded if tagging is deactivated.

1412 \cs_new_protected:Npn \tag_struct_insert_annot:nn #1 #2 %#1 should be an object reference
1413                                     %#2 struct parent num
1414 {
1415   \__tag_check_if_active_struct:T
1416   {
1417     \__tag_struct_insert_annot:nn {#1}{#2}
1418   }
1419 }
1420
1421 \cs_generate_variant:Nn \tag_struct_insert_annot:nn {xx,ee}
1422 \cs_new:Npn \tag_struct_parent_int: {\int_use:c { c@g__tag_parenttree_obj_int }}
1423
1424 \</package>
1425

```

(End of definition for `\tag_struct_insert_annot:nn` and `\tag_struct_parent_int:`. These functions are documented on page 106.)

7 Attributes and attribute classes

```

1426 \<header>
1427 \ProvidesExplPackage {tagpdf-attr-code} {2025-05-16} {0.99q}
1428 {part of tagpdf - code related to attributes and attribute classes}
1429 \</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
\g__tag_attr_objref_prop name. \l_@@_attr_value_tl is used to build the attribute array or key. Every time an
\l__tag_attr_value_tl 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

```

```

1430 \<package>
1431 \prop_new:N \g__tag_attr_entries_prop
1432 \prop_new_linked:N \g__tag_attr_class_used_prop
1433 \tl_new:N \l__tag_attr_value_tl
1434 \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.

```

1435 \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.
`newattribute (deprecated)`

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},
}

1436 \cs_new_protected:Npn \__tag_attr_new_entry:nn #1 #2 %#1:name, #2: content
1437 {
1438   \prop_gput:Nen \g__tag_attr_entries_prop
1439   {\pdf_name_from_unicode_e:n{#1}}{#2}
1440 }
1441
1442 \cs_generate_variant:Nn \__tag_attr_new_entry:nn {ee}
1443 \keys_define:nn { __tag / setup }
1444 {
1445   role/new-attribute .code:n =
1446   {
1447     \__tag_attr_new_entry:nn #1
1448   }
1449
  deprecated name
1449   ,newattribute .code:n =
1450   {
1451     \__tag_attr_new_entry:nn #1
1452   },
1453 }

(End of definition for \__tag_attr_new_entry:nn, role/new-attribute (setup-key), and newattribute
(deprecated). These functions are documented on page 109.)
```

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

```
1454 \keys_define:nn { __tag / struct }
1455 {
1456   attribute-class .code:n =
1457   {
1458     \clist_set:Ne \l__tag_tmpa_clist { #1 }
1459     \seq_set_from_clist:NN \l__tag_tmpb_seq \l__tag_tmpa_clist
1460
  we convert the names into pdf names with slash
1460     \seq_set_map_e:NNn \l__tag_tmpa_seq \l__tag_tmpb_seq
1461     {
1462       \pdf_name_from_unicode_e:n {##1}
1463     }
1464     \seq_map_inline:Nn \l__tag_tmpa_seq
1465     {
```

```

1466     \prop_get:NnNF \g__tag_attr_entries_prop {##1}\l__tag_tmpa_tl
1467     {
1468       \msg_error:nnn { tag } { attr-unknown } { ##1 }
1469     }
1470     \prop_gput:Nnn\g__tag_attr_class_used_prop { ##1} {}
1471   }
1472   \tl_set:Ne \l__tag_tmpa_tl
1473   {
1474     \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}
1475     \seq_use:Nn \l__tag_tmpa_seq { \c_space_tl }
1476     \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}
1477   }
1478   \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 0 }
1479   {
1480     \__tag_struct_prop_gput:nne
1481     { \int_use:N \c@g__tag_struct_abs_int }
1482     { C }
1483     { \l__tag_tmpa_tl }
1484     %\prop_show:c { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
1485   }
1486 }
1487 }

```

attribute (struct key)

```

1488 \keys_define:nn { __tag / struct }
1489 {
1490   attribute .code:n = % A property (attribute, value currently a dictionary)
1491   {
1492     \clist_set:Ne          \l__tag_tmpa_clist { #1 }
1493     \clist_if_empty:NF \l__tag_tmpa_clist
1494     {
1495       \seq_set_from_clist:NN \l__tag_tmppb_seq \l__tag_tmpa_clist
1496       \seq_set_map_e:NNn \l__tag_tmpa_seq \l__tag_tmppb_seq
1497       {
1498         \pdf_name_from_unicode_e:n {##1}
1499       }
1500       \tl_set:Ne \l__tag_attr_value_tl
1501       {
1502         \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[}%]
1503       }
1504       \seq_map_inline:Nn \l__tag_tmpa_seq
1505       {
1506         \prop_get:NnNF \g__tag_attr_entries_prop {##1}\l__tag_tmp_unused_tl
1507         {
1508           \msg_error:nnn { tag } { attr-unknown } { ##1 }
1509         }
1510         \prop_get:NnNF \g__tag_attr_objref_prop {##1}\l__tag_tmpa_tl
1511         {%\prop_show:N \g__tag_attr_entries_prop
1512          \pdf_object_unnamed_write:ne
1513          { dict }
1514          {
1515            \prop_item:Nn\g__tag_attr_entries_prop {##1}

```

```

1516         }
1517         \prop_gput:Nne \g__tag_attr_objref_prop {##1} {\pdf_object_ref_last:}
1518     }
1519     \tl_put_right:Ne \l__tag_attr_value_tl
1520     {
1521         \c_space_tl
1522         \prop_item:Nn \g__tag_attr_objref_prop {##1}
1523     }
1524     % \tl_show:N \l__tag_attr_value_tl
1525     }
1526     \tl_put_right:Ne \l__tag_attr_value_tl
1527     { %[
1528         \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{]}%
1529     }
1530     % \tl_show:N \l__tag_attr_value_tl
1531     \__tag_struct_prop_gput:nne
1532     { \int_use:N \c@g__tag_struct_abs_int }
1533     { A }
1534     { \l__tag_attr_value_tl }
1535 }
1536 },
1537 }
1538 \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} {2025-05-16} {0.99q}
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 \cs_set_protected:Npn \__tag_seq_gput_right:Nn #1 #2
36 {
37   \seq_gput_right:Nn #1 { #2 }
38   \lua_now:e { table.insert(ltx.__tag.tables['\cs_to_str:N#1'], "#2") }
39 }

```

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

```

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

```

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

```

71 </luatex>

```

The module declaration

```

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

```

```

78     version      = "0.99q",          --TAGVERSION
79     date         = "2025-05-16", --TAGDATE
80     description   = "tagpdf lua code",
81     license       = "The LATEX Project Public License 1.3c"
82 }
83
84 if luatexbase and luatexbase.provides_module then
85     luatexbase.provides_module (ProvidesLuaModule)
86 end
87
88 --[[
89 The code has quite probably a number of problems
90 - more variables should be local instead of global
91 - the naming is not always consistent due to the development of the code
92 - the traversing of the shipout box must be tested with more complicated setups
93 - it should probably handle more node types
94 -
95 --]]
96

```

Some comments about the lua structure.

```

97 --[[
98 the main table is named ltx.__tag. It contains the functions and also the data
99 collected during the compilation.
100
101 ltx.__tag.mc      will contain mc connected data.
102 ltx.__tag.role    will contain data related to parent-child relations.
103 ltx.__tag.struct  will contain structure related data.
104 ltx.__tag.page    will contain page data
105 ltx.__tag.tables  contains also data from mc and struct (from older code). This needs cleaning
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.showspace : boolean
136
137 ltx.tag.get_structnum: number, shows the current structure number
138 ltx.tag.get_structnum_next: number, shows the next structure number
139 --]]
140

```

This set-ups the main attribute registers. The mc_type attribute stores the type (P, Span etc) encoded as a num, The mc_cnt attribute stores the absolute number and allows so to see if a node belongs to the same mc-chunk.

The interwordspace attr is set by the function @@_mark_spaces, and marks the place where spaces should be inserted. The interwordfont attr is set by the function @@_mark_spaces too and stores the font, so that we can decide which font to use for the real space char. The interwordspaceOff attr allows to locally suppress the insertion of real space chars, e.g. when they are inserted by other means (e.g. with \char).

```

141 local mctypeattributeid = luatexbase.new_attribute ("g__tag_mc_type_attr")
142 local mcntattributeid   = luatexbase.new_attribute ("g__tag_mc_cnt_attr")
143 local 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.role = ltx.__tag.role or { } -- parent-child data
188 ltx.__tag.role.states = ltx.__tag.role.states or { } -- the states
189 ltx.__tag.role.index = ltx.__tag.role.index or { } -- standard types to index
190                                     --- numbers
191 ltx.__tag.role.matrix = ltx.__tag.role.matrix or { } -- implements the matrix
192 ltx.__tag.struct     = ltx.__tag.struct or { } -- struct data
193 ltx.__tag.tables     = ltx.__tag.tables or { } -- tables created with new prop and new seq.
194                                     -- wasn't a so great idea ...
195                                     -- g__tag_role_tags_seq used by tag<-> is in this table
196                                     -- used for pure lua tables too now!
197 ltx.__tag.page      = ltx.__tag.page or { } -- page data, currently only i->{0->mcnum,1->mcnum}
198 ltx.__tag.trace     = ltx.__tag.trace or { } -- show commands
199 ltx.__tag.func      = ltx.__tag.func or { } -- functions
200 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.

\

```

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

```



```

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

```

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

```

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

```

239 local __tag_pairs_prop =
240   function (prop)
241     local a = {}
242     for n in pairs(prop) do tableinsert(a, n) end
243     table.sort(a)
244     local i = 0           -- iterator variable
245     local iter = function () -- iterator function
246       i = i + 1

```

```

247         if a[i] == nil then return nil
248         else return a[i], prop[a[i]]
249         end
250     end
251     return iter
252 end
253
254
255 function ltx.__tag.trace.show_prop (prop)
256 if (type(prop) == "table") then
257     for i,v in __tag_pairs_prop (prop) do
258         __tag_log ("[" .. i .. "] => " .. tostring(v),1)
259     end
260 else
261     __tag_log ("prop " .. tostring(prop) .. " not found or not a table",1)
262 end
263 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

```

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

```

279 function ltx.__tag.trace.show_all_mc_data (min,max,loglevel)
280 for i = min, max do
281     ltx.__tag.trace.show_mc_data (i,loglevel)
282 end
283 texio.write_nl("")
284 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.

```

285 function ltx.__tag.trace.show_struct_data (num)
286 if ltx.__tag and ltx.__tag.struct and ltx.__tag.struct[num] then
287     for k,v in ipairs(ltx.__tag.struct[num]) do

```

```

288 __tag_log ("struct "..num..": "..tostring(k).."=>"..tostring(v),1)
289 end
290 else
291 __tag_log ("struct "..num.." not found ",1)
292 end
293 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.

```

294 local __tag_get_mc_cnt_type_tag = function (n)
295   local mcnt      = nodegetattribute(n,mcntattributeid) or -1
296   local mctype    = nodegetattribute(n,mctypeattributeid) or -1
297   local tag       = ltx.__tag.func.get_tag_from(mctype)
298   return mcnt,mctype,tag
299 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.

```

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

```

308 ltx.__tag.tables.role_tag_attribute = {}
309 ltx.__tag.tables.role_attribute_tag = {}

```

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

`ltx.__tag.func.alloctag`

```

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

```

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

```

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

```

353 function ltx.__tag.func.store_mc_data (num,key,data)
354   ltx.__tag.mc[num] = ltx.__tag.mc[num] or { }

```

```

355 ltx.__tag.mc[num][key] = data
356 __tag_log ("INFO TEX-STORE-MC-DATA: "..num.." => "..tostring(key).. => "..tostring(data),3)
357 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.

```

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

```

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

```

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

```

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

4.2 Functions to insert the pdf literals

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

```

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

```

```

388     return emcnode
389   end
390 end
391 else -- pdf mode
392   function __tag_backend_create_emc_node ()
393     local emcnode = nodenew("whatsit","pdf_literal")
394     emcnode.data = "EMC"
395     emcnode.mode=1
396     return emcnode
397   end
398 end
399
400 local function __tag_insert_emc_node (head,current)
401   local emcnode= __tag_backend_create_emc_node()
402   head = node.insert_before(head,current,emcnode)
403   return head
404 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

```

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

```

434 local __tag_backend_create_bdc_node
435
436 if tex.outputmode == 0 then
437   if token.get_macro("c_sys_backend_str") == "dvipdfmx" then
438     function __tag_backend_create_bdc_node (tag,dict)
439       local bdcnode = nodenew("whatsit","special")
440       bdcnode.data = "pdf:code /"..tag.."<<"..dict..">> BDC"
441       return bdcnode
442     end
443   else -- assume a dvips variant
444     function __tag_backend_create_bdc_node (tag,dict)
445       local bdcnode = nodenew("whatsit","special")
446       bdcnode.data = "ps:SDict begin mark/"..tag.."<<"..dict..">> /BDC pdfmark end"
447       return bdcnode
448     end
449   end
450 else -- pdf mode
451   function __tag_backend_create_bdc_node (tag,dict)
452     local bdcnode = nodenew("whatsit","pdf_literal")
453     bdcnode.data = "/"..tag.."<<"..dict..">> BDC"
454     bdcnode.mode=1
455     return bdcnode
456   end
457 end
458
459 local function __tag_insert_bdc_node (head,current,tag,dict)
460   bdcnode= __tag_backend_create_bdc_node (tag,dict)
461   head = node.insert_before(head,current,bdcnode)
462   return head
463 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.

```

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

```

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

```

(End of definition for __tag_show_spacemark.)

__tag_fakespace This is used to define a lua version of \pdf_fakespace
ltx.__tag.func.fakespace

```

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

```

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

```



```

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

```

(End of definition for __tag_mark_spaces.)

```

__tag_activate_mark_space These functions add/remove the function which marks the spaces to the callbacks
ltx.__tag.func.markspaceon pre_linebreak_filter and hpack_filter
ltx.__tag.func.markspaceoff
555 local function __tag_activate_mark_space ()
556 if not luatexbase.in_callback ("pre_linebreak_filter","markspaces") then
557     luatexbase.add_to_callback("pre_linebreak_filter",__tag_mark_spaces,"markspaces")
558     luatexbase.add_to_callback("hpack_filter",__tag_mark_spaces,"markspaces")
559 end
560 end
561
562 ltx.__tag.func.markspaceon=__tag_activate_mark_space
563
564 local function __tag_deactivate_mark_space ()
565 if luatexbase.in_callback ("pre_linebreak_filter","markspaces") then
566     luatexbase.remove_from_callback("pre_linebreak_filter","markspaces")

```

```

567 luatexbase.remove_from_callback("hpack_filter","markspaces")
568 end
569 end
570
571 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.

```

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

```

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

```

577 local function __tag_font_has_space (fontid)
578   t= fonts.hashes.identifiers[fontid]
579   if luaotfload.aux.slot_of_name(fontid,"space")
580     or t.characters and t.characters[32] and t.characters[32]["unicode"]==32
581   then
582     return true
583   else
584     return false
585   end
586 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.

```

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

```

```

612         space_char.font=curfont
613     end
614     head, space = node.insert_before(head, n, space_char) --
615     n.width      = n.width - space.width
616     space.attr   = n.attr
617 end
618 end
619 end
620 box.head = head
621 end
622 end
623
624 function ltx.__tag.func.space_chars_shipout (box)
625     __tag_space_chars_shipout (box)
626 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.

```

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

```

```

656 __tag_log("WARN TEX-MC-INSERT-MISSING: "..mcnum.." doesn't exist",0)
657 end
658 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.

```

659 function ltx.__tag.func.store_struct_mcabs (structnum,mcnum)
660 ltx.__tag.struct[structnum]=ltx.__tag.struct[structnum] or { }
661 ltx.__tag.struct[structnum]["mc"]=ltx.__tag.struct[structnum]["mc"] or { }
662 -- a structure can contain more than on mc chunk, the content should be ordered
663 tableinsert(ltx.__tag.struct[structnum]["mc"],mcnum)
664 __tag_log("INFO TEX-MC-INTO-STRUCT: "..
665           mcnum.." inserted in struct "..structnum,3)
666 -- but every mc can only be in one structure
667 ltx.__tag.mc[mcnum]= ltx.__tag.mc[mcnum] or { }
668 ltx.__tag.mc[mcnum]["parent"] = structnum
669 end
670

```

(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.

```

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

```

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

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

```

743     mcpopen,mcpagecnt,mccntprev,mctypeprev=
744     ltx.__tag.func.mark_page_elements (n,mcpagecnt,mccntprev,mcpopen,"INTERNAL VLIST",mctype
745 elseif n.id == GLUE and not n.leader then -- at glue real space chars are inserted, but t
746                                     -- been done if the previous shipout wandering, so here it
747 elseif n.id == LOCAL_PAR then -- local_par is ignored
748 elseif n.id == PENALTY then -- penalty is ignored
749 elseif n.id == KERN then -- kern is ignored
750     __tag_log ("INFO TAG-KERN-SUBTYPE: "..
751         node.type(node.getid(n)).." "..n.subtype,4)
752 else
753     -- math is currently only logged.
754     -- we could mark the whole as math
755     -- for inner processing the mlist_to_hlist callback is probably needed.
756 if n.id == MATH then
757     __tag_log("INFO TAG-MATH-SUBTYPE: "..
758         node.type(node.getid(n)).." "..__tag_get_mathsubtype(n),4)
759 end
760 -- endmath
761 __tag_log("INFO TAG-MC-COMPARE: current "..
762     mccnt.." prev "..mccntprev,4)
763 if mccnt~=mccntprev then -- a new mc chunk
764     __tag_log ("INFO TAG-NEW-MC-NODE: "..
765         node.type(node.getid(n))..
766         " MC"..tostring(mccnt)..
767         " <=> PREVIOUS "..tostring(mccntprev),4)
768 if mcpopen~=0 then -- there is a chunk open, close it (hope there is only one ...
769     box.list=__tag_insert_emc_node (box.list,n)
770     mcpopen = mcpopen - 1
771     __tag_log ("INFO TAG-INSERT-EMC: " ..
772         mcpagecnt .. " MCOPEN = " .. mcpopen,3)
773 if mcpopen ~=0 then
774     __tag_log ("WARN TAG-OPEN-MC: " .. mcpopen,1)
775 end
776 end
777 if ltx.__tag.mc[mccnt] then
778 if ltx.__tag.mc[mccnt]["artifact"] then
779     __tag_log("INFO TAG-INSERT-ARTIFACT: "..
780         tostring(ltx.__tag.mc[mccnt]["artifact"]),3)
781 if ltx.__tag.mc[mccnt]["artifact"] == "" then
782     box.list = __tag_insert_bmc_node (box.list,n,"Artifact")
783 else
784     box.list = __tag_insert_bdc_node (box.list,n,"Artifact", "/Type /"..ltx.__tag.mc[mccnt]
785 end
786 else
787     __tag_log("INFO TAG-INSERT-TAG: "..
788         tostring(tag),3)
789 mcpagecnt = mcpagecnt +1
790 __tag_log ("INFO TAG-INSERT-BDC: "..mcpagecnt,3)
791 local dict= "/MCID "..mcpagecnt
792 if ltx.__tag.mc[mccnt]["raw"] then
793     __tag_log("INFO TAG-USE-RAW: "..
794         tostring(ltx.__tag.mc[mccnt]["raw"]),3)
795     dict= dict .. " " .. ltx.__tag.mc[mccnt]["raw"]
796 end

```

```

797     if ltx.__tag.mc[mccnt]["alt"] then
798         __tag_log("INFO TAG-USE-ALT: "..
799             tostring(ltx.__tag.mc[mccnt]["alt"]),3)
800         dict= dict .. " " .. ltx.__tag.mc[mccnt]["alt"]
801     end
802     if ltx.__tag.mc[mccnt]["actualtext"] then
803         __tag_log("INFO TAG-USE-ACTUALTEXT: "..
804             tostring(ltx.__tag.mc[mccnt]["actualtext"]),3)
805         dict= dict .. " " .. ltx.__tag.mc[mccnt]["actualtext"]
806     end
807     box.list = __tag_insert_bdc_node (box.list,n,tag, dict)
808     ltx.__tag.func.store_mc_kid (mccnt,mcpagecnt,abspage)
809     ltx.__tag.func.store_mc_in_page(mccnt,mcpagecnt,abspage)
810     ltx.__tag.trace.show_mc_data (mccnt,3)
811 end
812 mcopen = mcopen + 1
813 else
814     if tagunmarkedbool.mode == truebool.mode then
815         __tag_log("INFO TAG-NOT-TAGGED: this has not been tagged, using artifact",2)
816         box.list = __tag_insert_bmc_node (box.list,n,"Artifact")
817         mcopen = mcopen + 1
818     else
819         __tag_log("WARN TAG-NOT-TAGGED: this has not been tagged",1)
820     end
821 end
822 mccntprev = mccnt
823 end
824 end -- end if
825 end -- end for
826 if head then
827     mccnthead, mctypehead,taghead = __tag_get_mc_cnt_type_tag (head)
828     __tag_log ("INFO TAG-ENDHEAD: " ..
829         node.type(node.getid(head))..
830         " MC"..tostring(mccnthead)..
831         " => TAG "..tostring(mctypehead)..
832         " => "..tostring(taghead),4)
833 else
834     __tag_log ("INFO TAG-ENDHEAD: ".. tostring(head),4)
835 end
836 __tag_log ("INFO TAG-QUITTING-BOX "..
837     tostring(name)..
838     " TYPE ".. node.type(node.getid(box)),4)
839 return mcopen,mcpagecnt,mccntprev,mctypeprev
840 end
841

```

(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.

```

842 function ltx.__tag.func.mark_shipout (box)
843     mcopen = ltx.__tag.func.mark_page_elements (box,-1,-100,0,"Shipout",-1)
844     if mcopen~=0 then -- there is a chunk open, close it (hope there is only one ...

```

```

845 local emcnode = __tag_backend_create_emc_node ()
846 local list = box.list
847 if list then
848     list = node.insert_after (list,node.tail(list),emcnode)
849     mcopen = mcopen - 1
850     __tag_log ("INFO SHIPOUT-INSERT-LAST-EMC: MCOPEN " .. mcopen,3)
851 else
852     __tag_log ("WARN SHIPOUT-UPS: this shouldn't happen",0)
853 end
854 if mcopen ~=0 then
855     __tag_log ("WARN SHIPOUT-MC-OPEN: " .. mcopen,1)
856 end
857 end
858 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.

```

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

```



```

891     --local objref    = ltx.__tag.tables[propname]["objref"] or "XXXX"
892     local objref = __tag_pdf_object_ref('__tag/struct',structnum)
893     numsentry = numsentry .. " " .. objref
894     end
895     numsentry = numsentry .. "]" "
896     __tag_log("INFO PARENTTREE-NUMENTRY: page " ..
897     page.. " num entry = " .. numsentry,3)
898     end
899     else
900         __tag_log ("INFO PARENTTREE-NO-DATA: page "..page,3)
901         numsentry = pdfpage.." ["
902     end
903     return numsentry
904 end
905
906 function ltx.__tag.func.output_parenttree (abspage)
907 for i=1,abspage do
908     line = ltx.__tag.func.fill_parent_tree_line (i) .. "^^J"
909     tex.sprint(catlatex,line)
910 end
911 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.

```

912 do
913     local properties = node.get_properties_table()
914     local is_soft_hyphen_prop = 'tagpdf.rewrite-softhyphen.is_soft_hyphen'
915     local hyphen_char = 0x2D
916     local soft_hyphen_char = 0xAD
917
918     A lookup table to test if the font supports the soft hyphen glyph.
919
920     local softhyphen_fonts = setmetatable({}, {_index = function(t, fid)
921         local fdir = identifiers[fid]
922         local format = fdir and fdir.format
923         local result = (format == 'opentype' or format == 'truetype')
924         local characters = fdir and fdir.characters
925         result = result and (characters and characters[soft_hyphen_char]) ~= nil
926         t[fid] = result
927         return result
928     end})
929
930     A pre shaping callback to mark hyphens as being hyphenation hyphens. This runs
931     before shaping to avoid affecting hyphens moved into discretionaries during shaping.
932
933     local function process_softhyphen_pre(head, _context, _dir)
934         if softhyphenbool.mode ~= truebool.mode then return true end
935         for disc, sub in node.traverse_id(DISC, head) do
936             if sub == explicit_disc or sub == regular_disc then
937                 for n, _ch, _f in node.traverse_char(disc.pre) do
938                     local props = properties[n]
939                     if not props then
940                         props = {}
941                         properties[n] = props
942                     end
943                 end
944             end
945         end
946     end
947
948     end
949
950 end

```

```

936         props[is_soft_hyphen_prop] = true
937     end
938 end
939 end
940 return true
941 end
942

```

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

```

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

```

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

8 parent-child rules

role_get_parent_child_rule

ltx.__tag.func.role_get_parent_child_rule

```

960 local function role_get_parent_child_rule (parent,child)
961     local state=
962     ltx.__tag.role.matrix[ltx.__tag.role.index[parent]]
963     and ltx.__tag.role.matrix[ltx.__tag.role.index[parent]][ltx.__tag.role.index[child]] or 0
964     return state
965 end
966 ltx.__tag.func.role_get_parent_child_rule=role_get_parent_child_rule

```

(End of definition for role_get_parent_child_rule and ltx.__tag.func.role_get_parent_child_rule. This function is documented on page ??.)

check_update_stashed

check_parent_child_rules

ltx.__tag.func.check_parent_child_rules

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

```

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

```

```

1015     string.match(ltx.__tag.tables['g__tag_struct_'.parent..'__prop']['rolemap'], "{(-
    )}{(-)}")
1016     local pp,ppNS=
1017     string.match(ltx.__tag.tables['g__tag_struct_'.parent..'__prop']['parentrole'], "{(-
    )}{(-)}")
1018     if pp == 'STASHED' then
1019         check_update_stashed (parent,loglevel,0)
1020         pp,ppNS=
1021         string.match(ltx.__tag.tables['g__tag_struct_'.parent..'__prop']['parentrole'], "{(-
    )}{(-)}")
1022     end
1023     __tag_log (parent..'': '.. pt..'': '..ptNS,loglevel)
1024     __tag_log (parent..'': '.. pr..'': '..prNS,loglevel)
1025     __tag_log (parent..'': '.. pp..'': '..ppNS,loglevel)
1026     -- now check the rule.
1027     -- at first rolemap of child against rolemap of parent.
1028     local state=ltx.__tag.func.role_get_parent_child_rule (pr,r)
1029     __tag_log ('rule of '..pr.."-">"..r.." is '..state,loglevel)
1030     -- if the state is 7 we check against parentrole of the parent
1031     if state == 7 then
1032         state=ltx.__tag.func.role_get_parent_child_rule (pp,r)
1033         __tag_log ('Parent-Child relation '..pp.."-">"..r.." is '..state,loglevel)
1034     end
1035     if state == 0 then
1036         __tag_log
1037         ('Warning: Parent-Child relation '
1038         '..ptNS..'': '..pt..' -> '..tNS..'': '..t..' is unknown',0)
1039         __tag_log
1040         ('Structure ' ..parent..' -> '..i,0)
1041     end
1042     if state == -1 then
1043         __tag_log
1044         ('Warning: Parent-Child relation '
1045         '..ptNS..'': '..pt..' -> '..tNS..'': '..t..' is not allowed',0)
1046         __tag_log
1047         ('Structure ' ..parent..' -> '..i,0)
1048     end
1049     __tag_log('=====',loglevel)
1050 end
1051 end -- end for
1052 end
1053
1054 ltx.__tag.func.check_parent_child_rules=check_parent_child_rules

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

1055 </lua>

```

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} {2025-05-16} {0.99q}
4 {part of tagpdf - code related to roles and structure names}
5 </header>
```

1 Code related to roles and structure names

⁶ `<*package>`

1.1 Variables

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

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

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

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

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

This are the main variables used by the code:

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

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

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

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

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

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

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

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

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

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

⁷ `\prop_new:N \g__tag_role_tags_NS_prop`

(End of definition for `\g__tag_role_tags_NS_prop`.)

`\g__tag_role_tags_class_prop` With pdf 2.0 we store the class in the NS dependent props. With pdf <2.0 we store for now the type(s) of a tag in a common prop. Tags that are rolemapped should get the type from the target.

`\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/dft>

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

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

`\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.

`\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.

`\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`

`\tl_new:N \l__tag_role_tag_tmpa_tl`

`\l__tag_role_tag_namespace_tmpb_tl` %

`\tl_new:N \l__tag_role_tag_namespace_tmpa_tl`

`\l__tag_role_role_tmpa_tl`

`\tl_new:N \l__tag_role_tag_namespace_tmpb_tl`

`\l__tag_role_role_namespace_tmpa_tl`

`\tl_new:N \l__tag_role_role_tmpa_tl`

`\l__tag_role_tmpa_seq`

`\tl_new:N \l__tag_role_role_namespace_tmpa_tl`

`\seq_new:N \l__tag_role_tmpa_seq`

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

1.2 Namespaces

The following commands setups a name space. With pdf version <2.0 this is only a prop with the rolemap. With pdf 2.0 a dictionary must be set up. Such a name space dictionaries can contain an optional /Schema and /RoleMapNS entry. We only reserve the objects but delay the writing to the finish code, where we can test if the keys and the name spaces are actually needed. This commands setups objects for the name space and its rolemap. It also initialize a dict to collect the rolemaps if needed, and a property with the tags of the name space and their rolemapping for loops. It is unclear if a reference to a schema file will be ever needed, but it doesn't harm

`g__tag_role/RoleMap_dict`
`\g__tag_role_rolemap_prop`

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

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

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

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

`__tag_role_NS_new:nnn`

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



```

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

```

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

We need an id for the user space. For the tests it should be possible to set it to a fix value. So we use random numbers which can be fixed by setting a seed. We fake a sort of GUID but do not try to be really exact as it doesn't matter ...

`\c__tag_role_userNS_id_str`

```

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

```

(End of definition for `\c__tag_role_userNS_id_str`.)

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

```

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

```

1.3 Adding a new tag

Both when reading the files and when setting up a tag manually we have to store data in various places.

`__tag_role_alloctag:nnn`

This command allocates a new tag without role mapping. In the lua backend it will also record the attribute value.

```

81 \pdf_version_compare:NnTF < {2.0}
82 {
83     \sys_if_engine luatex:TF
84     {

```

```

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

```

(End of definition for __tag_role_alloctag:nnn.)

1.3.1 pdf 1.7 and earlier

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

```

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

```

checks and messages

```

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

```

now the addition

```

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

```

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

```

142  \tl_if_empty:nF { #2 }
143  {
144    \prop_get:NnNTF \g__tag_role_rolemap_prop {#2}\l__tag_tpa_tl
145    {
146      \__tag_prop_gput:Nno \g__tag_role_rolemap_prop {#1}{\l__tag_tpa_tl}
147    }
148    {
149      \__tag_prop_gput:Nne \g__tag_role_rolemap_prop {#1}{\tl_to_str:n{#2}}
150    }
151  }
152  }
153  \cs_generate_variant:Nn \__tag_role_add_tag:nn {oo,ne}

```

(End of definition for __tag_role_add_tag:nn.)

For the parent-child test we must be able to get the role. We use the same number of arguments as for the 2.0 command. If there is no role, we assume a standard tag. Note: this is quite fast and a move to lua doesn't improve speed.

__tag_role_get:nnNN

```

154  \pdf_version_compare:NnT < {2.0}
155  {
156    \cs_new:Npn \__tag_role_get:nnNN #1#2#3#4 {%#1 tag, #2 NS, #3 tlvar which hold the role tag
157    {
158      \prop_get:NnNF \g__tag_role_rolemap_prop {#1}#3
159      {
160        \tl_set:Nn #3 {#1}
161      }
162      \tl_set:Nn #4 {}
163    }
164    \cs_generate_variant:Nn \__tag_role_get:nnNN {ooNN}
165  }
166

```

(End of definition for __tag_role_get:nnNN.)

1.3.2 The pdf 2.0 version

`__tag_role_add_tag:nnnn` The pdf 2.0 version takes four arguments: tag/namespace/role/namespace

```

167 \cs_new_protected:Nn \__tag_role_add_tag:nnnn %tag/namespace/role/namespace
168 {
169   \__tag_check_add_tag_role:nnn {#1/#2}{#3}{#4}
170   \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
171   {
172     \msg_info:nnn { tag }{new-tag}{#1}
173   }
174   \prop_if_exist:cTF
175   { g__tag_role_NS_#4_class_prop }
176   {
177     \prop_get:cnN { g__tag_role_NS_#4_class_prop } {#3}\l__tag_tmpa_tl
178     \quark_if_no_value:NT \l__tag_tmpa_tl
179     {
180       \tl_set:Nn\l__tag_tmpa_tl{--UNKNOWN--}
181     }
182   }
183   { \tl_set:Nn\l__tag_tmpa_tl{--UNKNOWN--} }
184   \__tag_role_alloctag:nno {#1}{#2}{ \l__tag_tmpa_tl }

```

Do not remap standard tags. TODO add warning?

```

185   \tl_if_in:nnF {-pdf-pdf2-mathml-}{-#2-}
186   {
187     \pdfdict_gput:nne {g__tag_role/RoleMapNS_#2_dict}{#1}
188     {
189       [
190         \pdf_name_from_unicode_e:n{#3}
191         \c_space_tl
192         \pdf_object_ref:n {tag/NS/#4}
193       ]
194     }
195   }

```

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

```

196   \tl_if_empty:nF { #2 }
197   {
198     \prop_get:cnN { g__tag_role_NS_#4_prop } {#3}\l__tag_tmpa_tl
199     \quark_if_no_value:NTF \l__tag_tmpa_tl
200     {
201       \__tag_prop_gput:cne { g__tag_role_NS_#2_prop } {#1}
202       {\tl_to_str:n{#3}}{\tl_to_str:n{#4}}
203     }
204     {
205       \__tag_prop_gput:cno { g__tag_role_NS_#2_prop } {#1}{\l__tag_tmpa_tl}
206     }
207   }

```

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

```

208   \bool_if:NT \l__tag_role_update_bool
209   {
210     \tl_if_empty:nF { #3 }

```

```

211     {
212       \tl_if_eq:nnF{#1}{#3}
213       {
214         \prop_get:NnN \g__tag_role_rolemap_prop {#3}\l__tag_tmpa_tl
215         \quark_if_no_value:NTF \l__tag_tmpa_tl
216         {
217           \__tag_prop_gput:Nne \g__tag_role_rolemap_prop {#1}{\tl_to_str:n{#3}}
218         }
219         {
220           \__tag_prop_gput:Nno \g__tag_role_rolemap_prop {#1}{\l__tag_tmpa_tl}
221         }
222       }
223     }
224   }
225 }
226 \cs_generate_variant:Nn \__tag_role_add_tag:nnnn {oooo}

```

(End of definition for __tag_role_add_tag:nnnn.)

For the parent-child test we must be able to get the role. We use the same number of arguments as for the <2.0 command. Note: this is quite fast and a move to lua doesn't improve speed.

__tag_role_get:nnNN

```

227 \pdf_version_compare:NnF < {2.0}
228 {
229   \cs_new:Npn \__tag_role_get:nnNN #1#2#3#4
230     {%#1 tag, #2 NS,
231      %#3 tlvar which hold the role tag
232      %#4 tlvar which hold the name of the target NS
233     {
234       \prop_if_exist:cTF {g__tag_role_NS_#2_prop}
235       {
236         \prop_get:cnNTF {g__tag_role_NS_#2_prop} {#1}\l__tag_get_tmpc_tl
237         {
238           \tl_set:Ne #3 {\exp_last_unbraced:No\use_i:nn {\l__tag_get_tmpc_tl}}
239           \tl_set:Ne #4 {\exp_last_unbraced:No\use_ii:nn {\l__tag_get_tmpc_tl}}
240         }
241         {
242           \msg_warning:nnn { tag } {role-unknown-tag} { #1 }
243           \tl_set:Nn #3 {#1}
244           \tl_set:Nn #4 {#2}
245         }
246       }
247       {
248         \msg_warning:nnn { tag } {role-unknown-NS} { #2 }
249         \tl_set:Nn #3 {#1}
250         \tl_set:Nn #4 {#2}
251       }
252     }
253   \cs_generate_variant:Nn \__tag_role_get:nnNN {ooNN}
254 }

```

(End of definition for __tag_role_get:nnNN.)

1.4 Helper command to read the data from files

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

`__tag_role_read_namespace_line:nw`

This command will process a line in the name space file. The first argument is the name of the name space. The definition differ for pdf 2.0. as we have proper name spaces there. With pdf<2.0 special name spaces shouldn't update the default role or add to the rolemap again, they only store the values for later uses. We use a boolean here.

```

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

```

```

301         {
302         \tl_set:Nn \l__tag_tmpa_tl {--UNKNOWN--}
303         }
304     }
305     {
306     \tl_set:Nn \l__tag_tmpa_tl {#5}
307     }
308     \__tag_role_alloctag:nno {#2} {#1} { \l__tag_tmpa_tl }
309     \bool_lazy_and:nnT
310     { ! \tl_if_empty_p:n {#3} } { ! \str_if_eq_p:nn {#1}{pdf2} }
311     {
312     \__tag_role_add_tag:nnnn {#2}{#1}{#3}{#4}
313     }
314     \__tag_prop_gput:cnn {g__tag_role_NS_#1_prop} {#2}{#3}{#4}
315 }
316 }
317 }

```

(End of definition for __tag_role_read_namespace_line:nw.)

__tag_role_read_namespace:nn

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

```

318 \cs_new_protected:Npn \__tag_role_read_namespace:nn #1 #2 %name of namespace #2 name of file
319 {
320     \prop_if_exist:cF {g__tag_role_NS_#1_prop}
321     { \msg_warning:nnn {tag}{namespace-unknown}{#1} }
322     \file_if_exist:nTF { tagpdf-ns-#2.def }
323     {
324     \ior_open:Nn \g_tmpa_ior {tagpdf-ns-#2.def}
325     \msg_info:nnn {tag}{read-namespace}{#2}
326     \ior_map_inline:Nn \g_tmpa_ior
327     {
328     \__tag_role_read_namespace_line:nw {#1} ##1,,, \q_stop
329     }
330     \ior_close:N \g_tmpa_ior
331     }
332     {
333     \msg_info:nnn {tag}{namespace-missing}{#2}
334     }
335 }
336

```

(End of definition for __tag_role_read_namespace:nn.)

__tag_role_read_namespace:n

This command reads the default namespace file.

```

337 \cs_new_protected:Npn \__tag_role_read_namespace:n #1 %name of namespace
338 {
339     \__tag_role_read_namespace:nn {#1}{#1}
340 }

```

(End of definition for __tag_role_read_namespace:n.)

1.5 Reading the default data

The order is important as we want pdf2 and latex as default: if two namespace define the same tag, the last one defines which one is used if the namespace is not explicitly given.

```

341 \__tag_role_read_namespace:n {pdf}
342 \__tag_role_read_namespace:n {pdf2}
343 \__tag_role_read_namespace:n {mathml}

```

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

```

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

```

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

```

351 \pdf_version_compare:NnTF < {2.0}
352 {
353   \hook_gput_code:nnn {begindocument}{tagpdf}
354   {
355     \bool_lazy_and:nnT
356     {
357       \cs_if_exist_p:N \chapter
358     }
359     {
360       \cs_if_exist_p:N \c@chapter
361     }
362     {
363       \prop_map_inline:cn{g__tag_role_NS_latex-book_prop}
364       {
365         \__tag_role_add_tag:ne {#1}{\use_i:nn #2\c_empty_tl\c_empty_tl}
366       }
367     }
368   }
369 }
370 {
371   \hook_gput_code:nnn {begindocument}{tagpdf}
372   {
373     \bool_lazy_and:nnT
374     {
375       \cs_if_exist_p:N \chapter
376     }
377     {
378       \cs_if_exist_p:N \c@chapter
379     }
380     {
381       \prop_map_inline:cn{g__tag_role_NS_latex-book_prop}
382       {
383         \prop_gput:Nnn \g__tag_role_tags_NS_prop { #1 }{ latex-book }
384         \__tag_prop_gput:Nne
385         \g__tag_role_rolemap_prop {#1}{\use_i:nn #2\c_empty_tl\c_empty_tl}
386       }
387     }
388   }
389 }

```



```

388     }
389 }

```

1.6 Parent-child rules

PDF define various rules about which tag can be a child of another tag. The following code implements the matrix to allow to use it in tests.

`\g_tag_role_parent_child_intarray` This intarray will store the rule as a number. For parent nm and child ij (n,m,i,j digits) the rule is at position nmij. As we have around 56 tags, we need roughly a size 6000.

```

390 \intarray_new:Nn \g_tag_role_parent_child_intarray {6000}

```

(End of definition for `\g_tag_role_parent_child_intarray`.)

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

(End of definition for `\c_tag_role_rules_prop` and `\c_tag_role_rules_num_prop`.)

`_tag_store_parent_child_rule:nnn` The helper command is used to store the rule. It assumes that parent and child are given as 2-digit number!

```

391 \sys_if_engine_luatex:TF
392 {
393   \cs_new_protected:Npn \_tag_store_parent_child_rule:nnn #1 #2 #3 % num parent, num child,
394     {
395       \prop_get:NnTF \c_tag_role_rules_prop{#3} \l_tag_tmp_unused_tl
396       {
397         \intarray_gset:Nnn \g_tag_role_parent_child_intarray
398         { #1#2 }{0\l_tag_tmp_unused_tl}
399         \lua_now:e
400         {
401           ltx.__tag.role.matrix[#1] = ltx.__tag.role.matrix[#1] or {}
402           ltx.__tag.role.matrix[#1][#2] = 0\l_tag_tmp_unused_tl
403         }
404       }
405     {
406       \intarray_gset:Nnn \g_tag_role_parent_child_intarray
407       { #1#2 }{0}
408       \lua_now:e
409       {
410         ltx.__tag.role.matrix[#1] = ltx.__tag.role.matrix[#1] or {}
411         ltx.__tag.role.matrix[#1][#2] = 0
412       }
413     }
414   }
415 }
416 {
417   \cs_new_protected:Npn \_tag_store_parent_child_rule:nnn #1 #2 #3 % num parent, num child,
418     {
419       \prop_get:NnTF \c_tag_role_rules_prop{#3} \l_tag_tmp_unused_tl
420       {
421         \intarray_gset:Nnn \g_tag_role_parent_child_intarray
422         { #1#2 }{0\l_tag_tmp_unused_tl}
423       }

```

```

424     {
425         \intarray_gset:Nnn \g__tag_role_parent_child_intarray
426         { #1#2 }{0}
427     }
428 }
429 }

```

(End of definition for __tag_store_parent_child_rule:nnn.)

1.6.1 Reading in the csv-files

This counter will be used to identify the first (non-comment) line

```

430 \int_zero:N \l__tag_tmpa_int

```

Open the file depending on the PDF version

```

431 \pdf_version_compare:NnTF < {2.0}
432 {
433     \ior_open:Nn \g_tmpa_ior {tagpdf-parent-child.csv}
434 }
435 {
436     \ior_open:Nn \g_tmpa_ior {tagpdf-parent-child-2.csv}
437 }

```

Now the main loop over the file

```

438 \ior_map_inline:Nn \g_tmpa_ior
439 {

```

ignore lines containing only comments

```

440     \tl_if_empty:nF{#1}
441     {

```

count the lines ...

```

442         \int_incr:N\l__tag_tmpa_int

```

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

```

443         \seq_set_from_clist:Nn\l__tag_tmpa_seq { #1 }
444         \int_compare:nNnTF {\l__tag_tmpa_int}=1

```

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

```

445     {
446         \seq_map_indexed_inline:Nn \l__tag_tmpa_seq
447         {
448             \prop_gput:Nne\g__tag_role_index_prop
449             {##2}
450             {\int_compare:nNnT{##1}<{10}{0}##1}
451         }
452     }

```

now the data lines.

```

453     {
454         \seq_set_from_clist:Nn\l__tag_tmpa_seq { #1 }

```

get the name of the child tag from the first column

```

455         \seq_pop_left:NN\l__tag_tmpa_seq\l__tag_tmpa_tl

```

get the number of the child, and store it in \l__tag_tmpp_tl

```

456         \prop_get:NoN \g__tag_role_index_prop { \l__tag_tmpa_tl } \l__tag_tmpp_tl

```

remove column 2+3

```
457 \seq_pop_left:NN\l__tag_tmpa_seq\l__tag_tmpa_tl
458 \seq_pop_left:NN\l__tag_tmpa_seq\l__tag_tmpa_tl
```

Now map over the rest. The index ##1 gives us the number of the parent, ##2 is the data.

```
459 \seq_map_indexed_inline:Nn \l__tag_tmpa_seq
460 {
461   \exp_args:Nne
462   \__tag_store_parent_child_rule:nnn {##1}{\l__tag_tmpb_tl}{ ##2 }
463 }
464 }
465 }
466 }
```

close the read handle.

```
467 \ior_close:N\g_tmpa_ior
```

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

```
468 \prop_get:NnN\g__tag_role_index_prop{StructTreeRoot}\l__tag_tmpa_tl
469 \prop_gput:Nne\g__tag_role_index_prop{Root}{\l__tag_tmpa_tl}
470 \prop_get:NnN\g__tag_role_index_prop{Hn}\l__tag_tmpa_tl
471 \pdf_version_compare:NnTF < {2.0}
472 {
473   \int_step_inline:nn{6}
474   {
475     \prop_gput:Nne\g__tag_role_index_prop{H#1}{\l__tag_tmpa_tl}
476   }
477 }
478 {
479   \int_step_inline:nn{10}
480   {
481     \prop_gput:Nne\g__tag_role_index_prop{H#1}{\l__tag_tmpa_tl}
482   }
483 }
```

all mathml tags are currently handled identically with the exception of math and mtext

```
483 \prop_get:NnN\g__tag_role_index_prop {mathml}\l__tag_tmpa_tl
484 \prop_get:NnN\g__tag_role_index_prop {math}\l__tag_tmpb_tl
485 \prop_get:NnN\g__tag_role_index_prop {mtext}\l__tag_tmpc_tl
486 \prop_map_inline:Nn \g__tag_role_NS_mathml_prop
487 {
488   \prop_gput:Nno\g__tag_role_index_prop {#1} {\l__tag_tmpa_tl}
489 }
490 \prop_gput:Nno\g__tag_role_index_prop{math}{\l__tag_tmpb_tl}
491 \prop_gput:Nno\g__tag_role_index_prop{mtext}{\l__tag_tmpc_tl}
492 }
493 \sys_if_engine luatex:T
494 {
495   \prop_map_inline:Nn\g__tag_role_index_prop
496   {
497     \lua_now:e { ltx.__tag.role.index['#1']=#2 }
498   }
499 }
```

1.6.2 Retrieving the parent-child rule

This command retrieves the rule (as a number) and stores it in the `tl`-var. It assumes that the tags in `#1` and `#2` are standard tags after role mapping for which a rule exist. If the parent is one of `Part`, `Div`, `NonStruct` the result can be state 7, which means that a check must be repeated for the “real parent”.

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

```

500 \tl_new:N \l__tag_parent_child_check_tl
501 \sys_if_engine luatex:TF
502 {
503   \cs_new_protected:Npn \__tag_role_get_parent_child_rule:nnN #1 #2 #3
504     % #1 parent (string, standard tag after rolemapping!)
505     % #2 child (string, standard tag after rolemapping!)
506     % #3 tl for state
507     {
508       \tl_set:Nn #3
509       {
510         \lua_now:etex.print{ltx.__tag.func.role_get_parent_child_rule('#1','#2')}
511       }

```

Debugging messages, this can perhaps go into debug mode.

```

512   \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
513   {
514     \prop_get:NoNF\c__tag_role_rules_num_prop {#3} \l__tag_tmpa_tl
515     {
516       \tl_set:Nn \l__tag_tmpa_tl {unknown}
517     }
518     \tl_set:Nn \l__tag_tmpb_tl {#1}
519     \msg_note:nneee
520     { tag }
521     { role-parent-child-result }
522     { #1 }
523     { #2 }
524     {
525       #3~(=\l__tag_tmpa_tl')
526     }
527   }
528   \int_compare:nNnT {#3} = { 0 }
529   {
530     \msg_warning:nneee
531     { tag }
532     {role-parent-child-result}
533     { #1 }
534     { #2 }
535     { unknown! }
536   }
537 }
538 }
539 }
540 {
541   \cs_new_protected:Npn \__tag_role_get_parent_child_rule:nnN #1 #2 #3
542     % #1 parent (string, standard tag after rolemapping)
543     % #2 child (string, standard tag after rolemapping)
544     % #3 tl for state

```

```

545     {
546         \prop_get:NnN \g__tag_role_index_prop{#1}\l__tag_tmpa_tl
547         \prop_get:NnN \g__tag_role_index_prop{#2}\l__tag_tmpb_tl
548         \bool_lazy_and:nnTF
549         { ! \quark_if_no_value_p:N \l__tag_tmpa_tl }
550         { ! \quark_if_no_value_p:N \l__tag_tmpb_tl }
551     }

```

Get the rule from the intarray

```

552         \tl_set:Nn#3
553         {
554             \intarray_item:Nn
555             \g__tag_role_parent_child_intarray
556             {\l__tag_tmpa_tl\l__tag_tmpb_tl}
557         }
558     }
559     {
560         \tl_set:Nn#3 {0}
561     }

```

Debugging messages, this can perhaps go into debug mode.

```

562         \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
563         {
564             \prop_get:NoNF\c__tag_role_rules_num_prop {#3} \l__tag_tmpa_tl
565             {
566                 \tl_set:Nn \l__tag_tmpa_tl {unknown}
567             }
568             \tl_set:Nn \l__tag_tmpb_tl {#1}
569             \msg_note:nneee
570             { tag }
571             { role-parent-child-result }
572             { #1 }
573             { #2 }
574             {
575                 #3~(=\l__tag_tmpa_tl')
576             }
577         }
578         \int_compare:nNnT {#3} = { 0 }
579         {
580             \msg_warning:nneee
581             { tag }
582             {role-parent-child-result}
583             { #1 }
584             { #2 }
585             { unknown! }
586         }
587     }
588 }
589 \cs_generate_variant:Nn\__tag_role_get_parent_child_rule:nnN {ooN}

```

(End of definition for __tag_role_get_parent_child_rule:nnN.)

__tag_role_check_parent_child:nnnnN

This command rolemaps its arguments and then calls __tag_role_get_parent-child_rule:nnN to retrieve the parent-child rule between both. It does not try to resolve inheritance rules of Part, Div and NonStruct but instead gives back the state 7. It is

then the task of the caller command to find the real parent and run the check again. In pdf 2.0 the name spaces of the tags are relevant, so we have arguments for them, but in pdf <2.0 they are ignored and can be left empty.

```

590 \pdf_version_compare:NnTF < {2.0}
591 {
592   \cs_new_protected:Npn \__tag_role_check_parent_child:nnnnN #1 #2 #3 #4 #5
593     % #1 parent tag,% not necessarily rolemapped, but often the case
594     % #2 NS (empty in pdf 1.x)
595     % #3 child tag, % not necessarily rolemapped, but often the case
596     % #4 NS (empty in pdf 1.x)
597     % #5 tl var: to give the result back.
598   {

```

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

```

599     \prop_get:NnNTF \g__tag_role_index_prop {#1}\l__tag_tmpa_tl
600     {
601       \tl_set:Nn \l__tag_tmpa_tl {#1}
602     }
603     {
604       \prop_get:NnNF \g__tag_role_rolemap_prop {#1}\l__tag_tmpa_tl
605       {
606         \tl_set:Nn \l__tag_tmpa_tl {\q_no_value}
607       }
608     }

```

now the child

```

609     \prop_get:NnNTF \g__tag_role_index_prop {#3}\l__tag_tmpp_tl
610     {
611       \tl_set:Nn \l__tag_tmpp_tl {#3}
612     }
613     {
614       \prop_get:NnNF \g__tag_role_rolemap_prop {#3}\l__tag_tmpp_tl
615       {
616         \tl_set:Nn \l__tag_tmpp_tl {\q_no_value}
617       }
618     }

```

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

```

619     \bool_lazy_and:nnTF
620     { ! \quark_if_no_value_p:N \l__tag_tmpa_tl }
621     { ! \quark_if_no_value_p:N \l__tag_tmpp_tl }
622     {
623       \__tag_role_get_parent_child_rule:ooN
624       { \l__tag_tmpa_tl }
625       { \l__tag_tmpp_tl }
626       #5
627     }
628     {
629       \tl_set:Nn #5 {0}
630       \msg_warning:nneee
631       { tag }
632       {role-parent-child-result}
633       { #1 }
634       { #3 }
635       { unknown! }

```

```

636     }
637   }
638 }

```

and now the pdf 2.0 version

```

639 {
640   \cs_new_protected:Npn \__tag_role_check_parent_child:nnnnN #1 #2 #3 #4 #5 %tag,NS,tag,NS,
641   {
642

```

If the namespace is empty, we assume a standard tag, otherwise we retrieve the rolemapping from the namespace

```

643     \tl_if_empty:nTF {#2}
644     {
645       \tl_set:Nn \l__tag_tmpa_tl {#1}
646     }
647     {
648       \prop_if_exist:cTF { g__tag_role_NS_#2_prop }
649       {
650         \prop_get:cnNTF
651         { g__tag_role_NS_#2_prop }
652         {#1}
653         \l__tag_tmpa_tl
654         {
655           \tl_set:Ne \l__tag_tmpa_tl {\tl_head:N\l__tag_tmpa_tl}
656           \tl_if_empty:NT\l__tag_tmpa_tl
657           {
658             \tl_set:Nn \l__tag_tmpa_tl {#1}
659           }
660         }
661         {
662           \tl_set:Nn \l__tag_tmpa_tl {\q_no_value}
663         }
664       }
665       {
666         \msg_warning:nnn { tag } {role-unknown-NS} { #2}
667         \tl_set:Nn \l__tag_tmpa_tl {\q_no_value}
668       }
669     }

```

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

```

670     \tl_if_empty:nTF {#4}
671     {
672       \tl_set:Nn \l__tag_tmpb_tl {#3}
673     }
674     {
675       \prop_if_exist:cTF { g__tag_role_NS_#4_prop }
676       {
677         \prop_get:cnNTF
678         { g__tag_role_NS_#4_prop }
679         {#3}
680         \l__tag_tmpb_tl
681         {
682           \tl_set:Ne \l__tag_tmpb_tl { \tl_head:N\l__tag_tmpb_tl }

```

```

683         \tl_if_empty:NT\l__tag_tmpb_tl
684         {
685             \tl_set:Nn \l__tag_tmpb_tl {#3}
686         }
687     }
688     {
689         \tl_set:Nn \l__tag_tmpb_tl {\q_no_value}
690     }
691 }
692 {
693     \msg_warning:nnn { tag } {role-unknown-NS} { #4}
694     \tl_set:Nn \l__tag_tmpb_tl {\q_no_value}
695 }
696 }

```

and now get the relation

```

697     \bool_lazy_and:nnTF
698     { ! \quark_if_no_value_p:N \l__tag_tmpa_tl }
699     { ! \quark_if_no_value_p:N \l__tag_tmpb_tl }
700     {
701         \__tag_role_get_parent_child_rule:ooN
702         { \l__tag_tmpa_tl }
703         { \l__tag_tmpb_tl }
704         #5
705     }
706     {
707         \tl_set:Nn #5 {0}
708         \msg_warning:nneee
709         { tag }
710         {role-parent-child-result}
711         { #2 : #1 }
712         { #4 : #3 }
713         { unknown! }
714     }
715 }
716 }
717 \cs_generate_variant:Nn\__tag_role_check_parent_child:nnnnN {oonnN,ooooN}
718 \endpackage

```

(End of definition for __tag_role_check_parent_child:nnnnN.)

\tag_check_child:nnTF

```

719 \base\prg_new_protected_conditional:Npnn \tag_check_child:nn #1 #2 {T,F,TF}{\prg_return_true.
720 \*package}
721 \prg_set_protected_conditional:Npnn \tag_check_child:nn #1 #2 {T,F,TF} {%#1 tag, #2 NS
722 {
723     \seq_get:NN\g__tag_struct_stack_seq\l__tag_tmpa_tl
724     \__tag_struct_get_role:enNN
725     {\l__tag_tmpa_tl}
726     {rolemap}
727     \l__tag_get_parent_tmpa_tl
728     \l__tag_get_parent_tmpb_tl
729     \__tag_role_check_parent_child:oonnN
730     { \l__tag_get_parent_tmpa_tl }
731     { \l__tag_get_parent_tmpb_tl }

```



```

732     {#1}{#2}
733     \l__tag_parent_child_check_tl
734     \int_compare:nNnT {\l__tag_parent_child_check_tl} = { \c__tag_role_rule_checkparent_tl }
735     {
736         \seq_get:NN\g__tag_struct_stack_seq\l__tag_tmpa_tl
737         \__tag_struct_get_role:enNN
738             {\l__tag_tmpa_tl}
739             {parentrole}
740             \l__tag_get_parent_tmpa_tl
741             \l__tag_get_parent_tmpb_tl
742         \__tag_role_check_parent_child:oonnN
743             { \l__tag_get_parent_tmpa_tl }
744             { \l__tag_get_parent_tmpb_tl }
745             {#1}{#2}
746             \l__tag_parent_child_check_tl
747     }
748     \int_compare:nNnTF { \l__tag_parent_child_check_tl } < {0}
749         {\prg_return_false:}
750         {\prg_return_true:}
751 }

```

(End of definition for `\tag_check_child:nnTF`. This function is documented on page 173.)

1.7 Key-val user interface

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

```

tag (rolemap-key)
tag-namespace (rolemap-key) 752 \keys_define:nn { __tag / tag-role }
role (rolemap-key)          753 {
role-namespace (rolemap-key) 754     ,tag .tl_set:N = \l__tag_role_tag_tmpa_tl
role/new-tag (setup-key)     755     ,tag-namespace .tl_set:N = \l__tag_role_tag_namespace_tmpa_tl
add-new-tag (deprecated)     756     ,role .tl_set:N = \l__tag_role_role_tmpa_tl
                             757     ,role-namespace .tl_set:N = \l__tag_role_role_namespace_tmpa_tl
                             758 }
                             759
                             760 \keys_define:nn { __tag / setup }
                             761 {
                             762     role/mathml-tags .bool_gset:N = \g__tag_role_add_mathml_bool
                             763     ,role/new-tag .code:n =
                             764     {
                             765         \keys_set_known:nnnN
                             766             {__tag/tag-role}
                             767             {
                             768                 tag-namespace=user,
                             769                 role-namespace=, %so that we can test for it.
                             770                 #1
                             771             }{__tag/tag-role}\l__tag_tmpa_tl
                             772         \tl_if_empty:NF \l__tag_tmpa_tl
                             773         {
                             774             \exp_args:NNno \seq_set_split:Nnn \l__tag_tmpa_seq { / } {\l__tag_tmpa_tl/}
                             775             \tl_set:Ne \l__tag_role_tag_tmpa_tl { \seq_item:Nn \l__tag_tmpa_seq {1} }
                             776             \tl_set:Ne \l__tag_role_role_tmpa_tl { \seq_item:Nn \l__tag_tmpa_seq {2} }

```

```

777     }
778     \tl_if_empty:NT \l__tag_role_role_namespace_tmpa_tl
779     {
780         \prop_get:NoNTF
781         \g__tag_role_tags_NS_prop
782         { \l__tag_role_role_tmpa_tl }
783         \l__tag_role_role_namespace_tmpa_tl
784         {
785             \prop_get:NoNF
786             \g__tag_role_NS_prop
787             { \l__tag_role_role_namespace_tmpa_tl }
788             \l__tag_tmp_unused_tl
789             {
790                 \tl_set:Nn \l__tag_role_role_namespace_tmpa_tl {user}
791             }
792         }
793     }
794     \tl_set:Nn \l__tag_role_role_namespace_tmpa_tl {user}
795 }
796 }
797 \pdf_version_compare:NnTF < {2.0}
798 {
799     %TODO add check for emptyness?
800     \__tag_role_add_tag:oo
801     { \l__tag_role_tag_tmpa_tl }
802     { \l__tag_role_role_tmpa_tl }
803 }
804 {
805     \__tag_role_add_tag:oooo
806     { \l__tag_role_tag_tmpa_tl }
807     { \l__tag_role_tag_namespace_tmpa_tl }
808     { \l__tag_role_role_tmpa_tl }
809     { \l__tag_role_role_namespace_tmpa_tl }
810 }
811 }
812 ,role/map-tags .choice:
813 ,role/map-tags/false .code:n = { \socket_assign_plug:nn { tag/struct/tag } {latex-
tags} }
814 ,role/map-tags/pdf .code:n = { \socket_assign_plug:nn { tag/struct/tag } {pdf-
tags} }
815 ,role/user-NS .code:n =
816 {
817     \pdf_version_compare:NnF < {2.0}
818     {
819         \pdf_string_from_unicode:nnN{utf8/string}{https://www.latex-project.org/ns/local/#1}
820         \tl_if_empty:NF \l__tag_tmpa_str
821         {
822             \pdfdict_gput:nne
823             {g__tag_role/Namespace_user_dict}
824             {NS}
825             {\l__tag_tmpa_str}
826         }
827     }
828 }

```

deprecated names

```
829      , mathml-tags .bool_gset:N = \g__tag_role_add_mathml_bool
830      , add-new-tag .meta:n = {role/new-tag={#1}}
831    }
832 \end{package}
```

(End of definition for tag (rolemap-key) and others. These functions are documented on page 173.)

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} {2025-05-16} {0.99q}
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/space
35       activate/spaces .default:n = true,
36     }
37   }
38   {
39     \keys_define:nn { __tag / setup }
40     {
41       activate/spaces .choices:nn = { true, false }
42       { \msg_warning:nnn {tag}{sys-no-interwordspace}{dvi} },
43       activate/spaces .default:n = true,
44     }
45   }
46 }
47
48
49 \sys_if_engine_luatex:T
50 {
51   \keys_define:nn { __tag / setup }
52   {
53     activate/spaces .choice:,
54     activate/spaces/true .code:n =
55     {
56       \bool_gset_true:N \g__tag_active_space_bool
57       \lua_now:e{!tx.__tag.func.markspaceon()}
58     },
59     activate/spaces/false .code:n =
60     {
61       \bool_gset_false:N \g__tag_active_space_bool
62       \lua_now:e{!tx.__tag.func.markspaceoff()}
63     },
64     activate/spaces .default:n = true,
65     debug/show/spaces .code:n =
66     { \lua_now:e{!tx.__tag.trace.showspace=true} },
67     debug/show/spacesOff .code:n =
68     { \lua_now:e{!tx.__tag.trace.showspace=nil} },
69   }
70 }

```

(End of definition for activate/spaces (setup-key), interwordspace (deprecated), and show-spaces (deprecated). These functions are documented on page ??.)

`__tag_fakespace`: For luatex we need a command for the fake space as equivalent of the pdftex primitive.

```

71 \sys_if_engine_luatex:T
72 {
73   \cs_new_protected:Nn \__tag_fakespace:
74   {
75     \group_begin:
76     \lua_now:e{\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 \end{package}

```

(End of definition for `__tag_fakespace:`, `\tag_spacechar_on:`, and `\tag_spacechar_off:`. These functions are documented on page ??.)

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

Symbols

\\ 10, 23, 27, 28, 44, 49, 50, 51, 56,
58, 60, 67, 70, 72, 78, 80, 93, 96, 97,
106, 107, 113, 114, 166, 539, 602, 610
_ 428, 439

A

activate_ (setup-key) 40, 284
activate-all (deprecated) (key) 1
activate-mc (deprecated) (key) 1
activate-struct (deprecated) (key) 1
activate-tree (deprecated) (key) 1
activate/all (key) 1, 260
activate/mc (key) 1, 260
activate/socket_ (setup-key) 284
activate/softhyphen (key) 1, 294
activate/space_ (setup-key) 196
activate/spaces (key) 1
activate/spaces_ (setup-key) 6
activate/struct (key) 1, 260
activate/struct-dest (key) 1, 260
activate/tagunmarked (key) 1, 291
activate/tree (key) 1, 260
actualtext (key) 1, 697
actualtext_ (mc-key) 78, 232, 378
add-new-tag_ (deprecated) 752
add-new-tag_ (setup-key) 173
\\AddToHook 13,
16, 33, 50, 63, 70, 99, 302, 386,
519, 521, 522, 526, 530, 537, 566, 616
AF (key) 1, 895
AFinline (key) 1, 895
AFinline-o (key) 1, 895
AFref (key) 1, 895
alt (key) 1, 697
alt_ (mc-key) 78, 232, 378
artifact_ (mc-key) 78, 232, 378
artifact-bool internal commands:
 __artifact-bool 181
artifact-type internal commands:
 __artifact-type 181
\\AssignSocketPlug 596, 597, 635, 642
attr-unknown 21, 84
attribute (key) 1, 1488
attribute-class (key) 1, 1454

B

benchmark commands:
 \\benchmark_tic: 619, 621

 \\benchmark_toc: 622
bool commands:
 \\bool_gset_eq:NN 654, 669, 681, 699
 \\bool_gset_false:N
 61, 215, 366, 655, 682
 \\bool_gset_true:N
 42, 56, 129, 175, 313, 807
 \\bool_if:NTF 9, 13, 18, 31,
 40, 40, 68, 75, 80, 85, 91, 96, 111,
 135, 146, 190, 197, 208, 228, 248,
 264, 265, 313, 314, 317, 317, 323,
 333, 351, 389, 425, 436, 450, 452,
 466, 469, 477, 502, 509, 568, 594,
 649, 664, 676, 694, 1126, 1158, 1189
 \\bool_if:nTF 6, 489
 \\bool_lazy_all:nTF 240
 \\bool_lazy_and:nnTF 274, 284, 304,
 309, 355, 373, 548, 619, 697, 710, 811
 \\bool_lazy_and_p:nn 8
 \\bool_new:N 7, 16, 20, 21, 41, 67, 73,
 124, 125, 126, 127, 128, 130, 132,
 134, 135, 136, 255, 322, 323, 645, 806
 \\bool_set_false:N
 17, 176, 205, 206, 207, 216,
 229, 230, 231, 344, 397, 605, 648, 675
 \\bool_set_true:N
 16, 131, 133, 215, 216,
 217, 240, 241, 242, 256, 346, 396, 604
box commands:
 \\box_dp:N 180, 184
 \\box_ht:N 170
 \\box_new:N 119, 120
 \\box_set_dp:Nn 178, 180
 \\box_set_eq:NN 193
 \\box_set_ht:Nn 177, 179
 \\box_use_drop:N 182, 186
\\boxmaxdepth 91, 181

C

c@g internal commands:
 \\c@g_tag_MCID_abs_int 11,
 15, 28, 37, 50, 57, 60, 68, 74, 92,
 138, 174, 178, 242, 245, 288, 295, 340
 \\c@g_tag_parenttree_obj_int 155, 500
 \\c@g_tag_struct_abs_int
 6, 18, 40, 58,
 91, 93, 114, 115, 118, 149, 165, 166,
 168, 258, 384, 525, 702, 715, 760,

772, 786, 802, 817, 825, 879, 890, 909, 912, 917, 953, 955, 960, 972, 974, 979, 1052, 1063, 1064, 1065, 1066, 1067, 1068, 1070, 1072, 1078, 1083, 1090, 1093, 1103, 1111, 1114, 1129, 1142, 1152, 1165, 1168, 1183, 1184, 1186, 1197, 1481, 1484, 1532	264, 266, 277, 280, 281, 285, 286, 292, 292, 295, 302, 305, 309, 315, 318, 318, 322, 322, 332, 337, 340, 344, 348, 349, 351, 352, 356, 370, 375, 378, 385, 393, 395, 397, 411, 416, 417, 422, 428, 433, 435, 462, 503, 509, 516, 523, 529, 530, 541, 550, 553, 558, 565, 573, 580, 592, 609, 612, 613, 614, 615, 615, 616, 617, 640, 646, 660, 673, 689, 830, 838, 851, 864, 897, 925, 1052, 1053, 1054, 1249, 1290, 1342, 1355, 1375, 1379, 1383, 1387, 1393, 1412, 1436
cctab commands:	
\c_document_cctab 75	
\chapter 184, 357, 375	
check commands:	
check_parent_child_rules 967	
check_update_stashed 967	
clist commands:	
\clist_const:Nn 121, 122	
\clist_if_empty:NTF 1493	
\clist_map_inline:nn ... 147, 419, 876	
\clist_new:N 117	
\clist_set:Nn 1458, 1492	
color commands:	
\color_select:n 428, 439	
cs commands:	
\cs:w 1364, 1368	
\cs_end: 1364, 1368	
\cs_generate_variant:Nn 44, 79, 100, 107, 107, 126, 134, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 153, 156, 157, 158, 163, 164, 168, 180, 191, 192, 193, 194, 194, 195, 196, 197, 214, 226, 247, 253, 263, 265, 276, 279, 304, 314, 317, 589, 608, 656, 717, 896, 924, 945, 1340, 1352, 1392, 1421, 1442	
\cs_gset_eq:NN 414	
\cs_if_exist:NTF .. 231, 570, 618, 619	
\cs_if_exist_p:N 9, 357, 360, 375, 378	
\cs_if_exist_use:NTF 400, 1346	
\cs_if_free:NTF 47	
\cs_new:Nn 83, 86, 109, 131, 136, 293, 409, 410, 411	
\cs_new:Npn 9, 15, 26, 74, 104, 149, 156, 159, 215, 229, 229, 231, 382, 503, 511, 517, 523, 1336, 1422	
\cs_new_protected:Nn 73, 127, 167, 296, 412, 416	
\cs_new_protected:Npn 13, 17, 20, 22, 23, 30, 30, 35, 42, 45, 49, 59, 60, 63, 65, 67, 78, 78, 79, 80, 81, 82, 82, 84, 84, 85, 86, 93, 93, 95, 107, 108, 117, 120, 121, 126, 137, 142, 147, 149, 153, 161, 163, 167, 169, 171, 172, 175, 189, 195, 227, 233, 246, 248, 251, 252, 253, 254, 255, 255, 256, 259, 260, 263,	
\cs_set:Nn 714, 715	
\cs_set:Npn 47, 52, 89, 109	
\cs_set_eq:NN 14, 20, 66, 80, 81, 82, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 222, 223, 224, 225, 234, 235, 236, 237, 238, 248, 257, 258, 377, 378, 379, 380, 621, 622, 707, 708, 709, 710, 716, 717, 721, 722, 723, 724	
\cs_set_protected:Nn 169, 210, 283, 353, 359, 1207, 1208	
\cs_set_protected:Npn .. 9, 15, 16, 22, 29, 35, 37, 40, 54, 55, 58, 58, 62, 65, 65, 70, 72, 73, 78, 82, 83, 86, 94, 94, 98, 101, 115, 119, 143, 201, 210, 220, 222, 224, 228, 235, 240, 249, 265, 283, 299, 305, 358, 362, 366, 370, 1056, 1057, 1243, 1251, 1292, 1344	
\cs_to_str:N 12, 18, 25, 32, 38, 43, 61, 62, 68, 69	
	D
debug/log (key) 1, 278	
debug/show (key) 277	
debug/structures _U (show-key) ... 41, 253	
debug/uncompress (key) 278	
\DebugSocketsOn 44	
\DeclareOption 43, 44	
dim commands:	
\c_max_dim 169, 194	
\c_zero_dim 177, 178, 179	
\documentclass 22	
\DocumentMetadata 21	
	E
E (key) 1, 697, 872	
\endinput 28	
\ERRORusetaggingsocket 106, 121	
exclude-header-footer _U (deprecated) 702	
exp commands:	
\exp_args:Ne 122, 505	

<code>\exp_args:NNe</code>	86, 86, 89, 195, 215	<code>\int_case:nnTF</code>	99, 114, 329
<code>\exp_args:Nne</code> 79, 317, 321, 417, 461, 490		<code>\int_compare:nNnTF</code>	22, 58, 70, 98, 116, 124, 124, 132,
<code>\exp_args:NNne</code>	86		142, 148, 156, 170, 173, 173, 261,
<code>\exp_args:NNno</code>	774		311, 341, 360, 387, 390, 418, 424,
<code>\exp_args:No</code>	291, 320		444, 450, 511, 512, 518, 525, 528,
<code>\exp_last_unbraced:Ne</code> . . .	99, 102, 109		532, 543, 552, 552, 558, 560, 562,
<code>\exp_last_unbraced:No</code> . .	134, 137,		567, 575, 578, 582, 594, 734, 748, 1094
	152, 154, 157, 159, 206, 207, 238,	<code>\int_compare:nTF</code>	
	239, 569, 572, 580, 583, 588, 592, 1232		180, 458, 1474, 1476, 1478, 1502, 1528
<code>\exp_not:n</code>	227, 246	<code>\int_compare_p:nNn</code>	715
F			
file commands:		<code>\int_decr:N</code>	212, 237
<code>\file_if_exist:nTF</code>	322	<code>\int_eval:n</code>	118, 138, 166, 197,
<code>\file_input:n</code>	310		396, 603, 611, 712, 717, 720, 917,
firstkid (key)	1, 697		960, 979, 1064, 1065, 1066, 1067,
flag commands:			1068, 1183, 1184, 1186, 1197, 1484
<code>\flag_clear:n</code>	239	<code>\int_gincr:N</code>	178, 242, 288,
<code>\flag_height:n</code>	178, 251		295, 342, 346, 350, 354, 360, 364,
<code>\flag_new:n</code>	176		368, 372, 500, 903, 1039, 1052, 1063
<code>\flag_raise:n</code>	252	<code>\int_gset:Nn</code>	7, 82, 158
fnote internal commands:		<code>\int_if_zero:nTF</code>	
<code>__fnote_gput_ref:nn</code>	65		212, 213, 237, 238, 599, 607
<code>\fontencoding</code>	6	<code>\int_incr:N</code>	93, 204, 228, 442
<code>\fontfamily</code>	6	<code>\int_new:N</code>	6, 78, 118, 123,
<code>\fontseries</code>	6		137, 155, 200, 325, 326, 327, 328, 895
<code>\fontshape</code>	6	<code>\int_rand:n</code> . .	61, 62, 64, 66, 68, 70, 71
<code>\fontsize</code>	6	<code>\int_set:Nn</code>	279, 282, 285, 286, 287
<code>\footins</code>	573, 592	<code>\int_step_inline:nn</code>	473, 479
G			
g internal commands:		<code>\int_step_inline:nnn</code>	
<code>\g_tag_struct_ref_by_dest:</code>	72		149, 174, 177, 200, 443, 449
group commands:		<code>\int_to_arabic:n</code>	154, 156
<code>\group_begin:</code>	66,	<code>\int_to_Hex:n</code> 61, 62, 64, 66, 68, 70, 71	
	75, 173, 311, 902, 994, 1002, 1037, 1062	<code>\int_use:N</code>	11,
<code>\group_end:</code>	73,		15, 18, 28, 37, 40, 50, 57, 58, 60,
	78, 207, 344, 920, 998, 1008, 1047, 1203		68, 74, 75, 91, 92, 93, 100, 115,
H			
<code>\halign</code>	44		165, 172, 174, 203, 220, 227, 228,
hbox commands:			241, 245, 246, 259, 261, 340, 384,
<code>\hbox_set:Nn</code>	171, 172		428, 439, 525, 548, 549, 557, 558,
hook commands:			702, 760, 772, 786, 802, 817, 825,
<code>\hook_gput_code:nnn</code>			879, 890, 906, 909, 912, 953, 955,
	7, 11, 33, 57, 66, 80,		972, 974, 1043, 1046, 1072, 1078,
	156, 239, 287, 288, 353, 371, 387,		1083, 1090, 1093, 1114, 1129, 1142,
	391, 730, 743, 753, 766, 776, 789, 809		1152, 1165, 1168, 1422, 1481, 1532
<code>\hook_new:n</code>	348	<code>\int_zero:N</code>	90, 105, 430
<code>\hook_use:n</code>	353	intarray commands:	
I			
<code>\ignorespaces</code>	40	<code>\intarray_gset:Nnn</code>	
int commands:			397, 406, 421, 421, 425
<code>\int_abs:n</code>	154	<code>\intarray_item:Nn</code>	423, 426, 554
		<code>\intarray_new:Nn</code>	390, 413
		interwordspace _□ (deprecated)	196, 6
		ior commands:	
		<code>\ior_close:N</code>	330, 467
		<code>\ior_map_inline:Nn</code>	326, 438

`\ior_open:Nn` [324, 433, 436](#)
`\g_tmpa_ior`
 [324, 326, 330, 433, 436, 438, 467](#)
 iow commands:
`\iow_newline:` [205, 303](#)
`\iow_now:Nn` [86](#)
`\iow_term:n` [198, 210, 213, 219, 223,](#)
 [283, 355, 359, 363, 367, 371, 375, 379](#)

K

kernel internal commands:
`__kernel_pdfdict_name:n` [45](#)
 keys commands:
`\keys_define:nn` . [8, 31, 34, 39, 51,](#)
 [131, 143, 181, 195, 202, 232, 245,](#)
 [254, 261, 290, 379, 393, 402, 409,](#)
 [415, 608, 697, 702, 752, 760, 820,](#)
 [872, 946, 1011, 1033, 1443, 1454, 1488](#)
`\keys_set:nn` [10, 18, 18, 19, 128, 185,](#)
 [295, 316, 318, 322, 418, 815, 1044, 1088](#)
`\keys_set_known:nnnN` [765](#)

L

`label (key)` [1, 697](#)
`\label` [12](#)
`label□(mc-key)` [79, 232, 378](#)
`lang (key)` [1, 697](#)
 legacy commands:
`\legacy_if:nTF` [84, 471, 474, 475](#)
`\llap` [428](#)
`log (deprecated) (key)` [278](#)
 ltx. internal commands:
`ltx.__tag.func.alloctag` [310](#)
`ltx.__tag.func.check_parent_-`
 `child_rules` [967](#)
`ltx.__tag.func.fakespace` [489](#)
`ltx.__tag.func.fill_parent_tree_-`
 `line` [859](#)
`ltx.__tag.func.get_num_from` [319](#)
`ltx.__tag.func.get_tag_from` [338](#)
`ltx.__tag.func.mark_page_-`
 `elements` [690](#)
`ltx.__tag.func.mark_shipout` [842](#)
`ltx.__tag.func.markspaceoff` [555](#)
`ltx.__tag.func.markspaceon` [555](#)
`ltx.__tag.func.mc_insert_kids` .. [627](#)
`ltx.__tag.func.mc_num_of_kids` .. [368](#)
`ltx.__tag.func.output_num_from` . [319](#)
`ltx.__tag.func.output_parenttree` [859](#)
`ltx.__tag.func.output_tag_from` . [338](#)
`ltx.__tag.func.role_get_parent_-`
 `child_rule` [960](#)
`ltx.__tag.func.space_chars_-`
 `shipout` [587](#)

`ltx.__tag.func.store_mc_data` ... [353](#)
`ltx.__tag.func.store_mc_in_page` [671](#)
`ltx.__tag.func.store_mc_kid` [362](#)
`ltx.__tag.func.store_mc_label` .. [358](#)
`ltx.__tag.func.store_struct_-`
 `mcabs` [659](#)
`ltx.__tag.func.update_mc_-`
 `attributes` [679](#)
`ltx.__tag.tables.role_tag_-`
 `attribute` [308](#)
`ltx.__tag.trace.log` [222](#)
`ltx.__tag.trace.show_all_mc_data` [279](#)
`ltx.__tag.trace.show_mc_data` ... [264](#)
`ltx.__tag.trace.show_prop` [239](#)
`ltx.__tag.trace.show_seq` [230](#)
`ltx.__tag.trace.show_struct_data` [285](#)
 lua commands:
`\lua_escape:n` [32](#)
`\lua_now:n` [8, 12, 15, 18,](#)
 [25, 26, 32, 35, 38, 42, 43, 49, 50, 54,](#)
 [57, 59, 61, 62, 62, 66, 68, 68, 69, 73,](#)
 [76, 86, 87, 87, 88, 96, 100, 109, 111,](#)
 [120, 133, 137, 138, 152, 158, 160,](#)
 [172, 181, 189, 224, 237, 238, 246,](#)
 [262, 283, 297, 307, 399, 408, 497, 510](#)

M

`\MakeLinkTarget` [142](#)
`mathml (key)` [1, 895](#)
`\maxdimen` [192](#)
`mc-current` [20, 16](#)
`mc-current□(show-key)` [41, 143](#)
`mc-data□(show-key)` [41, 131](#)
`mc-label-unknown` [20, 9](#)
`mc-marks□(show-key)` [41, 202](#)
`mc-nested` [20, 6](#)
`mc-not-open` [20, 13](#)
`mc-popped` [20, 14](#)
`mc-pushed` [20, 14](#)
`mc-tag-missing` [20, 8](#)
`mc-used-twice` [20, 12](#)
`\MessageBreak` [15, 19, 20, 21](#)
 msg commands:
`\msg_error:nn` [299, 320, 473, 1100](#)
`\msg_error:nnn`
 ... [336, 347, 355, 366, 459, 1468, 1508](#)
`\msg_error:nnnnn` [545, 554](#)
`\msg_info:nnn`
 [134, 172, 313, 325, 333, 389, 393](#)
`\msg_info:nnnn` [343, 362, 402](#)
`\msg_line_context:` .. [93, 97, 107,](#)
 [114, 506, 507, 539, 543, 547, 603, 611](#)
`\g_msg_module_name_prop` [30, 34](#)
`\g_msg_module_type_prop` [33](#)

\msg_new:nnn	7, 8, 9, 12, 13, 14, 15, 16, 22, 24, 25, 32, 35, 36, 38, 40, 42, 47, 54, 65, 74, 85, 86, 87, 88, 89, 90, 92, 94, 104, 111, 164, 213, 215, 216, 217, 218, 219, 220, 222, 506, 507, 537, 541, 545, 597, 605
\msg_new:nnnn	225
\msg_note:nn	29, 198
\msg_note:nnn	203, 220, 527, 534, 569, 577
\msg_note:nnnn	127, 226, 245, 513, 520, 554, 562, 562
\msg_note:nnnnn	519, 569
\msg_redirect_name:nnn	541
\msg_show_item_unbraced:n	275
\msg_show_item_unbraced:nn	266
\msg_term:nnnnnn	260, 269
\msg_warning:nn	24, 222
\msg_warning:nnn	12, 14, 42, 44, 53, 242, 248, 306, 321, 329, 374, 382, 407, 431, 666, 693, 848, 861, 1285, 1304, 1330
\msg_warning:nnnn	440, 590, 719
\msg_warning:nnnnn	126, 175, 530, 580, 630, 708
\msg_warning:nnnnnn	146
N	
namespace_(rolemap-key)	173
new-tag	21, 215
newattribute_(deprecated)	109, 1436
\newcommand	601, 602
\newcounter	8
\NewDocumentCommand	6, 23, 29, 34, 40, 46, 51, 56, 126, 315, 606
\newmarks	13
\NewSocketPlug	585, 591, 628, 636
no-struct-dest (deprecated) (key)	1
\nointerlineskip	185
P	
\PackageError	13
\PackageWarning	28, 563
page/exclude-header-footer_(setup-key)	43, 702
page/tabsorder (key)	1, 296
para-flattened_(deprecated)	393
para-hook-count-wrong	21, 225
para/flattened_(tool-key)	393
para/maintag_(setup-key)	393
para/maintag_(tool-key)	393
para/tag_(setup-key)	393
para/tag_(tool-key)	393
para/tagging_(setup-key)	42, 393
para/tagging_(tool-key)	393
\PARALABEL	495
paratag_(deprecated)	393
paratagging_(deprecated)	42, 393
paratagging-show_(deprecated)	42, 393
parent (key)	1, 697
pdf commands:	
\pdf_activate_indexed_structure_-destination:	310
\pdf_bdc:nn	237
\pdf_bdc_shipout:nn	238
\pdf_bmc:n	235
\l_pdf_current_structure_-destination_tl	308
\pdf_emc:	236
\pdf_name_from_unicode_e:n	104, 114, 119, 167, 180, 190, 278, 1006, 1439, 1462, 1498
\pdf_object_if_exist:n	138
\pdf_object_if_exist:nTF	950, 1015
\pdf_object_new:n	108, 34, 36, 154, 262, 310, 321
\pdf_object_new_indexed:nn	31, 1069
\pdf_object_ref:n	108, 56, 131, 135, 139, 141, 192, 318, 335, 953, 1017
\pdf_object_ref_indexed:nn	57, 74, 96, 168, 211, 255, 271, 414, 435, 496, 1338
\pdf_object_ref_last:	108, 104, 118, 124, 294, 1403, 1409, 1517
\pdf_object_unnamed_write:nn	100, 111, 120, 286, 1395, 1512
\pdf_object_write:nnn	257, 281, 311, 330, 337, 342
\pdf_object_write_indexed:nnnn	139, 449
\pdf_pageobject_ref:n	221, 486
\pdf_string_from_unicode:nnN	42, 819
\pdf_uncompress:	288, 290
\pdf_version_compare:NnTF	20, 81, 136, 154, 159, 227, 257, 324, 351, 431, 471, 590, 797, 817
pdfannot commands:	
\pdfannot_dict_put:nnn	140, 737, 760, 783
\pdfannot_link_ref_last:	747, 770, 793
pdfdict commands:	
\pdfdict_gput:nnn	38, 45, 53, 187, 276, 334, 822
\pdfdict_if_empty:nTF	328
\pdfdict_new:n	18, 35, 37
\pdfdict_put:nnn	995, 996, 1003, 1004, 1005, 1038
\pdfdict_use:n	283, 332, 339

\pdffakespace 42, 313
pdf file commands:
 \pdf file_embed_file:nnn ... 148, 1040
 \pdf file_embed_stream:nnN . 896, 904
 \pdf file_embed_stream:nnn 141
\pdf glyph to unicode 30
\pdf interword space off 198, 117
\pdf interword space on 198, 33, 121
pdf management commands:
 \pdf management_add:nnn
 52, 70, 71, 298, 300, 302, 393
 \pdf management_if_active_p: ... 9, 10
 \pdf management_remove:nn 304
phoneme (key) 697
prg commands:
 \prg_do_nothing: 82, 102, 117, 377,
 378, 379, 380, 414, 721, 722, 723, 724
 \prg_generate_conditional_-
 variant:Nnn 138
 \prg_new_conditional:Nnn ... 68, 226
 \prg_new_conditional:Npnn
 233, 257, 272, 282, 481, 487, 498
 \prg_new_eq_conditional:NNn . 82, 233
 \prg_new_protected_conditional:Npnn
 719
 \prg_replicate:nn 153
 \prg_return_false: 78, 230, 234, 252,
 263, 266, 279, 289, 484, 496, 502, 749
 \prg_return_true: .. 79, 229, 249,
 262, 276, 286, 485, 495, 501, 719, 750
 \prg_set_conditional:Npnn 238
 \prg_set_protected_conditional:Npnn
 721
process commands:
 process_softhyphen_pre_ process_-
 softhyphen_post 912
\ProcessOptions 45
prop commands:
 \prop_clear:N 176
 \prop_count:N 203
 \prop_gclear:N 826
 \prop_get:NnN 126, 144, 145,
 177, 198, 214, 268, 299, 456, 468,
 470, 483, 484, 485, 546, 547, 560, 561
 \prop_get:NnNTF 44, 96, 130,
 137, 144, 158, 181, 183, 201, 205,
 236, 294, 295, 304, 324, 339, 358,
 395, 405, 419, 432, 514, 564, 599,
 604, 609, 614, 650, 660, 676, 677,
 727, 780, 785, 840, 853, 1136, 1233,
 1299, 1358, 1396, 1466, 1506, 1510
 \prop_gput:Nnn
 26, 30, 31, 33, 34, 56, 88,
 90, 91, 97, 98, 99, 100, 101, 103,
 110, 112, 113, 119, 121, 122, 142,
 145, 183, 269, 272, 286, 291, 383,
 434, 436, 437, 448, 469, 475, 481,
 488, 490, 491, 701, 827, 829, 1185,
 1196, 1270, 1315, 1438, 1470, 1517
 \prop_gremove:Nn 137, 147, 830
 \prop_gset_eq:NN 146, 1182
 \prop_gset_from_keyval:Nn 799
 \prop_if_exist:NTF 174,
 209, 234, 320, 430, 648, 675, 1255, 1296
 \prop_if_exist_p:N 712
 \prop_if_in:NnTF 76
 \prop_item:Nn
 41, 80, 99, 102, 109, 115,
 144, 187, 244, 508, 1193, 1515, 1522
 \prop_map_function:NN 264
 \prop_map_inline:Nn . 74, 267, 272,
 293, 326, 363, 381, 398, 486, 495, 813
 \prop_map_tokens:Nn 344
 \prop_new:N .. 7, 8, 9, 10, 11, 11, 25,
 33, 114, 144, 180, 798, 1065, 1431, 1434
 \prop_new_linked:N
 17, 66, 71, 73, 181, 1432
 \prop_put:Nnn 143, 188
 \prop_show:N
 .. 67, 95, 189, 1179, 1200, 1484, 1511
property commands:
 \property_new:nnnn
 163, 166, 170, 173, 177
 \property_record:nn 152
 \property_ref:nn 107, 157
 \property_ref:nnn
 41, 156, 161, 181, 190,
 221, 222, 325, 460, 471, 487, 1256, 1260
\providecommand 62, 63, 64, 65, 66, 69, 70, 320
\ProvidesExplFile 3
\ProvidesExplPackage 3, 3,
 3, 3, 3, 3, 3, 3, 3, 7, 7, 26, 37, 1427

Q

quad 232, 233
quark commands:
 \q_no_value 606, 616, 662, 667, 689, 694
 \quark_if_no_value:NTF
 131, 178, 199, 215, 269, 300, 566, 577
 \quark_if_no_value_p:N
 549, 550, 620, 621, 698, 699
 \q_stop 259, 292, 328

R

raw_ (mc-key) 78, 232, 378
ref (key) 1, 697, 872
\RemoveFromHook 34, 524, 525
\renewcommand 604, 605

[\RenewDocumentCommand](#) 8
[\RequirePackage](#)
 20, 46, 316, 319, 325, 328, 564
[\rlap](#) 439
[role_□\(rolemap-key\)](#) 173, 752
role commands:
 [role_get_parent_child_rule](#) 960
[role-MC-child-forbidden](#) 104
[role-missing](#) 21, 86
[role-namespace_□\(rolemap-key\)](#) . 173, 752
[role-parent-child-check](#) 90
[role-parent-child-forbidden](#) 111
[role-parent-child-result](#) 21, 92
[role-parent-child-unresolved](#) 164
[role-remapping](#) 21, 213
[role-struct-parent-child-forbidden](#) . 94
[role-tag](#) 21, 215
[role-unknown](#) 21, 86
[role-unknown-NS](#) 21, 86
[role-unknown-tag](#) 21, 86
[role/new-attribute_□\(setup-key\)](#) 109, 1436
[role/new-tag_□\(setup-key\)](#) 752
[root-AF \(key\)](#) 1, 1011
[root-supplemental-file \(key\)](#) 1033

S

[\selectfont](#) 6
seq commands:
 [\seq_clear:N](#) 319, 448
[\seq_const_from_clist:Nn](#) 21, 34
[\seq_count:N](#) 22, 25, 58,
 331, 444, 1474, 1476, 1478, 1502, 1528
[\seq_get:NN](#) 723, 736
[\seq_get:NNTF](#) 469, 584, 1096, 1222, 1229
[\seq_gpop:NN](#) 1215
[\seq_gpop:NNTF](#) 105, 1216
[\seq_gpop_left:NN](#) 307
[\seq_gpush:Nn](#) 13, 15, 88, 95, 1103, 1109
[\seq_gput_left:Nn](#) .. 42, 185, 273, 311
[\seq_gput_right:Nn](#)
 . 37, 146, 152, 184, 236, 257, 296, 468
[\seq_gset_eq:NN](#) 159, 221, 326
[\seq_if_empty:NNTF](#) 200, 438
[\seq_item:Nn](#)
 . 59, 116, 118, 125, 129, 136, 140,
 186, 348, 355, 368, 491, 493, 500,
 669, 670, 685, 686, 736, 737, 775, 776
[\seq_log:N](#) 175, 199, 248, 394, 555, 570
[\seq_map_function:NN](#) 273
[\seq_map_indexed_inline:Nn](#) 446, 459
[\seq_map_inline:Nn](#) 289, 320, 1464, 1504
[\seq_new:N](#)
 . 12, 14, 14, 15, 16, 17, 18, 19, 21,
 22, 24, 115, 116, 145, 182, 1068, 1435
[\seq_pop_left:NN](#) 455, 457, 458
[\seq_put_right:Nn](#) 321
[\seq_remove_all:Nn](#) 324
[\seq_set_eq:NN](#) 207, 208
[\seq_set_from_clist:NN](#) ... 1459, 1495
[\seq_set_from_clist:Nn](#)
 87, 90, 196, 216, 443, 454
[\seq_set_map_e:NNn](#) 1460, 1496
[\seq_set_split:Nnn](#) 51,
 145, 662, 666, 678, 682, 729, 733, 774
[\seq_show:N](#)
 60, 188, 215, 216, 249, 322,
 323, 325, 478, 1112, 1180, 1201, 1211
[\seq_use:Nn](#)
 50, 110, 111, 205, 232, 233, 383, 1475
Setup keys:
 [activate-all \(deprecated\)](#) 1
 [activate-mc \(deprecated\)](#) 1
 [activate-struct \(deprecated\)](#) 1
 [activate-tree \(deprecated\)](#) 1
 [activate/all](#) 1, 260
 [activate/mc](#) 1, 260
 [activate/softhyphen](#) 1, 294
 [activate/spaces](#) 1
 [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, 1011
 [root-supplemental-file](#) 1033
 [tabsorder \(deprecated\)](#) 1, 296
 [tagunmarked \(deprecated\)](#) 1, 291
 [uncompress \(deprecated\)](#) 278
shipout commands:
 [\g_shipout_readonly_int](#)
 91, 172, 241, 396
[show-kids](#) 21, 64
[show-spaces_□\(deprecated\)](#) 196, 6
[show-struct](#) 21, 64
[\ShowTagging](#) 18, 41, 125
skip commands:
 [\skip_horizontal:n](#) 77
 [\c_zero_skip](#) 77
socket commands:
 [\socket_assign_plug:n](#)
 200, 204, 205,
 209, 210, 517, 518, 534, 696, 813, 814
 [\socket_new:n](#) . 183, 184, 445, 446, 657

<code>\socket_new_plug:nnn</code>	185 , 448 , 467 , 500 , 658 , 674
<code>\socket_use:n</code>	28 , 76 , 519 , 521 , 528 , 532
<code>\socket_use:nn</code>	81 , 199 , 335 , 750 , 1161 , 1277 , 1322
<code>\socket_use:nnn</code>	86
<code>\socket_use:nw</code>	97
<code>\socket_use_expandable:n</code>	92
<code>\socket_use_expandable:nw</code> ..	66 , 112
<code>stash (key)</code>	1 , 697
<code>stash_(mc-key)</code>	79 , 181
str commands:	
<code>\str_case:nnTF</code>	52 , 618 , 1119
<code>\str_const:Nn</code>	59
<code>\str_if_eq:nnTF</code>	117 , 127 , 500 , 586 , 630
<code>\str_if_eq_p:nn</code>	310 , 491 , 493
<code>\str_if_exist:NTF</code>	443 , 583 , 626
<code>\str_new:N</code>	113
<code>\str_set_convert:Nnnn</code>	146 , 255 , 276 , 392 , 405 , 754 , 766 , 780 , 796 , 811 , 884
<code>\str_use:N</code>	67 , 266 , 289
<code>\c_tilde_str</code>	57 , 59
<code>\string</code>	20 , 21 , 22
<code>struct-faulty-nesting</code>	21 , 32
<code>struct-label-unknown</code>	21 , 38
<code>struct-missing-tag</code>	21 , 35
<code>struct-no-objnum</code>	21 , 24
<code>struct-orphan</code>	21 , 25
<code>struct-Ref-unknown</code>	42
<code>struct-show-closing</code>	21 , 40
<code>struct-stack_(show-key)</code>	41 , 245
<code>struct-unknown</code>	21 , 22
<code>struct-used-twice</code>	21 , 36
Structure keys:	
<code>actualtext</code>	1 , 697
<code>AF</code>	1 , 895
<code>AFinline</code>	1 , 895
<code>AFinline-o</code>	1 , 895
<code>AFref</code>	1 , 895
<code>alt</code>	1 , 697
<code>attribute</code>	1 , 1488
<code>attribute-class</code>	1 , 1454
<code>E</code>	1 , 697 , 872
<code>firstkid</code>	1 , 697
<code>label</code>	1 , 697
<code>lang</code>	1 , 697
<code>mathml</code>	1 , 895
<code>parent</code>	1 , 697
<code>phoneme</code>	697
<code>ref</code>	1 , 697 , 872
<code>stash</code>	1 , 697
<code>tag</code>	1 , 697
<code>texsource</code>	1 , 895
<code>title</code>	1 , 697
<code>title-o</code>	1 , 697
<code>\SuspendTagging</code>	44
sys commands:	
<code>\c_sys_backend_str</code>	52
<code>\c_sys_engine_str</code>	12 , 14
<code>\sys_if_engine luatex:TF</code>	42 , 49 , 71 , 83 , 84 , 105 , 187 , 308 , 343 , 391 , 493 , 501
<code>\sys_if_engine pdftex:TF</code> ...	26 , 111
<code>\sys_if_output_pdf:TF</code>	11 , 28 , 113
<code>sys-no-interwordspace</code>	21 , 222
T	
<code>tabsorder (deprecated) (key)</code>	1 , 296
<code>tag (key)</code>	1 , 697
<code>tag_(mc-key)</code>	78 , 232 , 378
<code>tag_(rolemap-key)</code>	173 , 752
tag commands:	
<code>\tag_check_benchmark_on:</code>	617
<code>\tag_check_child:nn</code>	719 , 721
<code>\tag_check_child:nnTF</code>	173 , 719
<code>\tag_get:n</code>	18 , 79 , 106 , 107 , 123 , 124 , 88 , 91 , 229 , 229 , 539
<code>\tag_if_active:</code>	233 , 238
<code>\tag_if_active:TF</code>	18 , 18 , 230 , 231 , 539
<code>\tag_if_active_p:</code>	18 , 230 , 811
<code>\tag_if_box_tagged:N</code>	257
<code>\tag_if_box_tagged:NTF</code>	18 , 256
<code>\tag_if_box_tagged_p:N</code>	18 , 256
<code>\tag_mc_add_missing_to_stream:Nn</code> ..	78 , 66 , 189 , 225 , 573 , 577 , 589 , 592
<code>\tag_mc_artifact_group_begin:n</code> .	77 , 59 , 59 , 62
<code>\tag_mc_artifact_group_end:</code>	77 , 59 , 60 , 70
<code>\tag_mc_begin:n</code>	11 , 77 , 25 , 65 , 113 , 169 , 169 , 295 , 295 , 299 , 305 , 427 , 438 , 464 , 496 , 657 , 685 , 736 , 759 , 782
<code>\tag_mc_begin_pop:n</code>	77 , 75 , 79 , 80 , 101 , 666 , 696 , 750 , 773 , 796
<code>\tag_mc_end:</code>	77 , 31 , 74 , 92 , 210 , 210 , 295 , 296 , 353 , 359 , 429 , 440 , 506 , 663 , 692 , 748 , 771 , 794
<code>\tag_mc_end_push:</code>	77 , 64 , 79 , 79 , 82 , 651 , 678 , 734 , 757 , 780
<code>\tag_mc_if_in:</code>	82 , 233
<code>\tag_mc_if_in:TF</code>	77 , 42 , 68 , 226
<code>\tag_mc_if_in_p:</code>	77 , 68 , 226
<code>\tag_mc_new_stream:n</code>	78 , 17 , 17 , 67 , 67
<code>\tag_mc_reset_box:N</code>	78 , 78 , 78 , 222 , 222
<code>\tag_mc_use:n</code>	77 , 35 , 35 , 36 , 37
<code>\l_tag_para_attr_class_tl</code> ..	388 , 390
<code>\tag_resume:n</code>	7 , 72 , 199 , 235 , 248 , 258 , 662 , 691

\tag_socket_use:n .. 43 , 44 , 62 , 72 , 73	__tag_add_missing_mcs:Nn
\tag_socket_use:nn .. 43 , 44 , 63 , 72 , 78 92 , 167 , 167 , 219
\tag_socket_use:nnn .. 43 , 44 , 64 , 72 , 83	__tag_add_missing_mcs_to_-
\tag_socket_use_expandable:n	stream:Nn .. 65 , 65 , 66 , 189 , 189 , 225
..... 43 , 44 , 65 , 72 , 89	\g__tag_attr_class_used_prop ..
\tag_spacechar_off: ... 81 , 81 , 86 , 115 291 , 293 , 1430 , 1470
\tag_spacechar_on: ... 81 , 82 , 98 , 119	\g__tag_attr_class_used_seq 289 , 1435
\tag_start: 7 , 199 , 210 , 223 , 252	\g__tag_attr_entries_prop
\tag_start:n 7 , 199 , 248 , 256 , 258	295 , 1430 , 1438 , 1466 , 1506 , 1511 , 1515
\tag_stop: ... 7 , 53 , 199 , 201 , 222 , 251	__tag_attr_new_entry:nn
\tag_stop:n 7 , 199 , 234 , 255 , 257	... 672 , 1436 , 1436 , 1442 , 1447 , 1451
\tag_struct_begin:n	\g__tag_attr_objref_prop
.. 106 , 48 , 455 , 462 , 480 , 490 , 684 , 1430 , 1510 , 1517 , 1522
735 , 758 , 781 , 1052 , 1052 , 1056 , 1057	\l__tag_attr_value_tl 1430 ,
\tag_struct_end:	1500 , 1519 , 1524 , 1526 , 1530 , 1534
.. 106 , 26 , 53 , 508 , 512 , 693 , 749 ,	__tag_backend_create_bdc_node .. 434
772 , 795 , 1052 , 1053 , 1207 , 1208 , 1246	__tag_backend_create_bmc_node .. 405
\tag_struct_end:n ... 106 , 1054 , 1243	__tag_backend_create_emc_node .. 376
\tag_struct_gput:nnn 107 ,	__tag_check_add_tag_role:nn
67 , 76 , 878 , 1342 , 1342 , 1344 , 1352 129 , 332 , 332
\tag_struct_gput_ref:nnn 107	__tag_check_add_tag_role:nnn ..
\tag_struct_insert_annot:nn .. 106 , 169 , 351
144 , 747 , 770 , 793 , 1412 , 1412 , 1421	__tag_check_benchmark_tic: . 356 ,
\tag_struct_object_ref:n	360 , 364 , 368 , 372 , 376 , 380 , 615 , 621
.. 106 , 844 , 857 , 868 , 1335 , 1336 , 1340	__tag_check_benchmark_toc: . 358 ,
\tag_struct_parent_int: 106 , 144 ,	362 , 366 , 370 , 374 , 378 , 382 , 616 , 622
740 , 747 , 763 , 770 , 786 , 793 , 1412 , 1422	__tag_check_forbidden_parent_-
\tag_struct_use:n	child:nnnn 120 , 120 , 134 , 170
..... 106 , 107 , 58 , 1249 , 1249 , 1251	__tag_check_if_active_mc: 272
\tag_struct_use_num:n	__tag_check_if_active_mc:TF ..
..... 106 , 1290 , 1290 , 1292 84 , 103 ,
\tag_suspend:n	171 , 191 , 212 , 271 , 301 , 307 , 355 , 361
... 7 , 67 , 199 , 224 , 234 , 257 , 658 , 686	__tag_check_if_active_struct: . 282
\tag_tool:n 40 , 13 , 13 , 14 , 16 , 20	__tag_check_if_active_struct:TF
tag internal commands: 39 , 271 , 1059 , 1060 ,
__tag_activate_mark_space 555	1212 , 1213 , 1245 , 1253 , 1294 , 1415
\g__tag_active_mc_bool	__tag_check_if_mc_in_galley: .. 481
..... 40 , 124 , 243 , 263 , 270 , 274	__tag_check_if_mc_in_galley:TF
\l__tag_active_mc_bool 208 , 229
.... 130 , 206 , 216 , 230 , 241 , 246 , 274	__tag_check_if_mc_tmb_missing: 487
\l__tag_active_socket_bool	__tag_check_if_mc_tmb_missing:TF
..... 75 , 80 , 85 , 112 , 217 , 234 , 487
91 , 96 , 111 , 130 , 207 , 217 , 231 , 242 , 292	__tag_check_if_mc_tmb_missing_-
\g__tag_active_space_bool	p: 487
..... 13 , 56 , 61 , 124	__tag_check_if_mc_tme_missing: 498
\g__tag_active_struct_bool	__tag_check_if_mc_tme_missing:TF
.... 124 , 242 , 265 , 272 , 284 , 306 , 466 155 , 221 , 238 , 498
\l__tag_active_struct_bool	__tag_check_if_mc_tme_missing_-
.... 130 , 205 , 215 , 229 , 240 , 245 , 284	p: 498
\g__tag_active_struct_dest_bool	__tag_check_info_closing_-
..... 124 , 269 , 276 , 305	struct:n 309 , 309 , 317 , 1218
\g__tag_active_tree_bool	__tag_check_init_mc_used:
... 9 , 68 , 124 , 244 , 264 , 271 , 351 , 389 411 , 411 , 414 , 420

<code>__tag_check_mc_if_nested:</code>	<code>__tag_exclude_headfoot_begin:</code> .
. 174 , 312 , 370 , 370 646 , 707 , 708
<code>__tag_check_mc_if_open:</code>	<code>__tag_exclude_headfoot_end:</code>
. 214 , 365 , 370 , 378 660 , 709 , 710
<code>__tag_check_mc_in_galley:TF</code> 481	<code>__tag_exclude_struct_headfoot_-</code>
<code>__tag_check_mc_in_galley_p:</code> 481	begin:n 673 , 714 , 715
<code>__tag_check_mc_pushed_popped:nn</code>	<code>__tag_exclude_struct_headfoot_-</code>
. 89 , 96 , 109 , 112 , 117 , 385 , 385	end: 689 , 716 , 717
<code>__tag_check_mc_tag:N</code>	<code>__tag_fakespace</code> 489
. 187 , 324 , 397 , 397	<code>__tag_fakespace:</code> 71 , 73 , 317
<code>__tag_check_mc_used:n</code>	<code>__tag_finish_structure:</code>
. 145 , 268 , 416 , 416 13 , 16 , 348 , 349
<code>\g__tag_check_mc_used_intarray</code>	<code>\l__tag_get_child_tmpa_tl</code>
. 411 , 421 , 423 , 426 101 , 544 , 549 , 616 , 618 , 628 ,
<code>__tag_check_no_open_struct:</code>	631 , 642 , 1259 , 1263 , 1265 , 1271 , 1281
. 318 , 318 , 1220 , 1227	<code>\l__tag_get_child_tmpb_tl</code>
<code>__tag_check_para_begin_show:nn</code> 101 , 545 , 550 , 617 , 629
. 422 , 463 , 495	<code>\l__tag_get_child_tmppc_tl</code>
<code>__tag_check_para_end_show:nn</code> 101 , 145 , 157 , 159
. 433 , 507	<code>__tag_get_data_mc_counter:</code> 9 , 9
<code>__tag_check_show_MCID_by_page:</code>	<code>__tag_get_data_mc_tag:</code>
. 435 , 435 231 , 231 , 293 , 293
<code>__tag_check_struct_forbidden_-</code>	<code>__tag_get_data_struct_counter:</code>
parent_child:nnn 137 , 163 , 603 522 , 523
<code>__tag_check_struct_used:n</code>	<code>__tag_get_data_struct_id:</code> 511 , 511
. 322 , 322 , 1258	<code>__tag_get_data_struct_num:</code> 516 , 517
<code>__tag_check_structure_has_tag:n</code>	<code>__tag_get_data_struct_tag:</code> 503 , 503
. 292 , 292 , 1093	<code>__tag_get_mathsubtype</code> 300
<code>__tag_check_structure_tag:N</code>	<code>__tag_get_mc_abs_cnt:</code>
. 302 , 302 , 671 , 694 , 745 14 , 15 , 19 , 20 ,
<code>__tag_check_typeout_v:n</code> 110 ,	102 , 137 , 165 , 176 , 183 , 204 , 240 ,
111 , 114 , 149 , 157 , 164 , 202 , 211 ,	248 , 266 , 287 , 301 , 311 , 374 , 382 , 402
224 , 224 , 283 , 473 , 489 , 505 , 576 , 587	<code>__tag_get_mc_cnt_type_tag</code> 294
<code>__tag_check_unresolved_parent_-</code>	<code>__tag_get_num_from</code> 319
child:nnnn 169 , 169	<code>\l__tag_get_parent_tmpa_tl</code>
<code>\g__tag_css_bool</code> 806 , 807 , 811 , 822 101 , 126 , 131 , 135 , 138 , 148 , 151 ,
<code>\g__tag_css_prop</code>	161 , 164 , 174 , 539 , 547 , 560 , 566 ,
. 798 , 799 , 813 , 826 , 827 , 829 , 830	570 , 573 , 640 , 642 , 727 , 730 , 740 , 743
<code>__tag_debug_mc_begin_ignore:n</code>	<code>\l__tag_get_parent_tmppb_tl</code>
. 348 , 516 101 , 149 ,
<code>__tag_debug_mc_begin_insert:n</code>	152 , 162 , 165 , 174 , 540 , 548 , 561 ,
. 309 , 509	577 , 581 , 584 , 641 , 728 , 731 , 741 , 744
<code>__tag_debug_mc_end_ignore:</code> 373 , 530	<code>\l__tag_get_parent_tmppc_tl</code>
<code>__tag_debug_mc_end_insert:</code> 363 , 523 101 , 144 , 152 , 154
<code>__tag_debug_struct_begin_-</code>	<code>__tag_get_tag_from</code> 338
ignore:n 558 , 1205	<code>\l__tag_get_tmppc_tl</code> 101 ,
<code>__tag_debug_struct_begin_-</code>	181 , 186 , 204 , 206 , 207 , 236 , 238 ,
insert:n 550 , 1202	239 , 1139 , 1145 , 1361 , 1363 , 1367 , 1373
<code>__tag_debug_struct_end_check:n</code>	<code>__tag_gincr_para_begin_int:</code>
. 580 , 1245 340 , 344 , 362 , 378 , 461 , 488
<code>__tag_debug_struct_end_ignore:</code>	<code>__tag_gincr_para_end_int:</code>
. 573 , 1240 340 , 352 , 370 , 380 , 504
<code>__tag_debug_struct_end_insert:</code>	<code>__tag_gincr_para_main_begin_-</code>
. 565 , 1238	int: 340 , 340 , 358 , 377 , 454 , 479

```

__tag_gincr_para_main_end_int:
    ..... 340, 348, 366, 379, 511
__tag_hook_kernel_after_foot: .
    ..... 615, 623, 640, 710, 717, 724
__tag_hook_kernel_after_head: .
    ..... 613, 621, 632, 709, 716, 723
__tag_hook_kernel_before_foot:
    ..... 614, 622, 638, 708, 715, 722
__tag_hook_kernel_before_head:
    ..... 612, 620, 630, 707, 714, 721
\g__tag_in_mc_bool .....
    ..... 16, 18, 175, 215, 228,
    313, 366, 654, 655, 669, 681, 682, 699
__tag_insert_bdc_node ..... 434
__tag_insert_bmc_node ..... 405
__tag_insert_emc_node ..... 376
__tag_lastpagelabel: .... 81, 82, 100
__tag_log ..... 222
\l__tag_loglevel_int .....
    .... 123, 124, 132, 170, 173, 279,
    282, 285, 286, 287, 311, 341, 360,
    388, 391, 418, 511, 512, 518, 525,
    532, 552, 558, 560, 562, 567, 575, 582
__tag_mark_spaces ..... 494
__tag_mc_artifact_begin_marks:n
    ..... 23, 45, 81, 321
\l__tag_mc_artifact_bool .....
    ..... 20, 176, 184, 190, 216, 317
\l__tag_mc_artifact_type_tl ....
    ..... 19, 188, 192, 196,
    200, 204, 208, 212, 216, 319, 321, 324
__tag_mc_bdc:nn ..... 234, 237, 283
__tag_mc_bdc_mcid:n ... 123, 239, 255
__tag_mc_bdc_mcid:nn .....
    ..... 239, 240, 257, 262
__tag_mc_bdc_shipout:nn .. 238, 248
__tag_mc_begin_marks:nn .....
    ..... 23, 23, 44, 80, 328
__tag_mc_bmc:n ..... 234, 235, 279
__tag_mc_bmc_artifact: 277, 277, 290
__tag_mc_bmc_artifact:n 277, 281, 291
\l__tag_mc_botmarks_seq .....
    ..... 92, 21, 90, 111,
    161, 208, 216, 216, 221, 233, 483, 500
__tag_mc_check_parent_child:n .
    ..... 121, 121, 180, 201, 337
__tag_mc_disable_marks: .... 78, 78
__tag_mc_emc: .... 158, 234, 236, 368
__tag_mc_end_marks: .. 23, 63, 82, 369
\l__tag_mc_firstmarks_seq .....
    ..... 92, 21, 87, 110, 196, 199,
    200, 207, 208, 215, 232, 483, 491, 493
\g__tag_mc_footnote_marks_seq ... 14
__tag_mc_get_marks: . 84, 84, 207, 228
__tag_mc_handle_artifact:N ....
    ..... 119, 277, 285, 319
__tag_mc_handle_mc_label:n ....
    ..... 26, 26, 194, 331
__tag_mc_handle_mcid:nn .....
    ..... 239, 260, 265, 325
__tag_mc_handle_stash:n 49, 140,
    142, 143, 168, 204, 266, 266, 276, 340
__tag_mc_if_in: .... 68, 82, 226, 233
__tag_mc_if_in:TF 68, 86, 226, 372, 380
__tag_mc_if_in_p: ..... 68, 226
__tag_mc_insert_extra_tmb:n ...
    ..... 108, 108, 171
__tag_mc_insert_extra_tme:n ...
    ..... 108, 153, 172
__tag_mc_insert_mcid_kids:n ...
    ..... 131, 131, 150, 309
__tag_mc_insert_mcid_single_
kids:n ..... 131, 136, 310
\l__tag_mc_key_label_tl .....
    . 22, 192, 194, 296, 328, 329, 331, 414
\l__tag_mc_key_properties_tl ...
    ..... 22, 177, 245, 260, 261,
    281, 282, 327, 388, 397, 398, 410, 411
\l__tag_mc_key_stash_bool .....
    ..... 20, 31, 40, 183, 197, 333
\g__tag_mc_key_tag_tl .... 19, 22,
    180, 219, 231, 237, 293, 315, 367, 384
\l__tag_mc_key_tag_tl 22, 179, 187,
    189, 218, 236, 314, 324, 326, 328, 383
__tag_mc_lua_set_mc_type_attr:n
    ..... 83, 83, 107, 189
__tag_mc_lua_unset_mc_type_
attr: ..... 83, 109, 217
\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, 184, 272
\l__tag_mc_ref_abspage_tl ..... 11
__tag_mc_set_label_used:n 30, 30, 50
\g__tag_mc_stack_seq .....
    ..... 18, 88, 95, 105, 394
__tag_mc_store:nnn .. 93, 93, 107, 134
\l__tag_mc_tmpa_tl ..... 12
g__tag_MCID_abs_int ..... 7
\g__tag_mode_lua_bool .....
    . 41, 42, 135, 146, 248, 313, 314,
    317, 323, 568, 594, 649, 664, 676, 694
__tag_new_output_prop_handler:n
    ..... 74, 84, 108, 1066
__tag_pairs_prop ..... 239

```

\l__tag_para_attr_class_tl	\c__tag_property_struct_clist
. 321 , 390 , 493 121 , 705
\g__tag_para_begin_int	\l__tag_Ref_tmpa_tl 105
. 321 , 346 , 364 , 428 , 552 , 557	g__tag_role/RoleMap_dict 18
\l__tag_para_bool 321 , 395 , 404 , 411 ,	\g__tag_role_add_mathml_bool
417 , 450 , 469 , 502 , 604 , 605 , 648 , 675 73 , 265 , 762 , 829
\g__tag_para_end_int	__tag_role_add_tag:nn
. 321 , 354 , 372 , 439 , 552 , 558 127 , 127 , 153 , 280 , 365 , 800
\l__tag_para_flattened_bool	__tag_role_add_tag:nnnn
. 321 , 400 , 407 , 420 , 452 , 477 , 509 167 , 167 , 226 , 312 , 805
\l__tag_para_main_attr_class_tl	__tag_role_alloctag:nnn 81 ,
. 321 , 483	85 , 95 , 107 , 117 , 126 , 141 , 184 , 277 , 308
\g__tag_para_main_begin_int	__tag_role_check_parent_-
. 321 , 342 , 360 , 543 , 548	child:nnnnN 150 ,
\g__tag_para_main_end_int	163 , 546 , 590 , 592 , 640 , 717 , 729 , 742
. 321 , 350 , 368 , 543 , 549	\l__tag_role_debug_prop 11
__tag_para_main_store_struct:	__tag_role_get:nnNN 154 ,
. 382 , 382 , 459 , 485	156 , 164 , 227 , 229 , 253 , 687 , 738 , 1104
\g__tag_para_main_struct_tl 321 , 384	__tag_role_get_parent_child_-
\l__tag_para_main_tag_tl	rule:nnN
. 321 , 399 , 406 , 419 , 457 , 482 189 , 500 , 503 , 541 , 589 , 623 , 701
\l__tag_para_show_bool	\g__tag_role_index_prop
. 321 , 396 , 397 , 412 , 425 , 436 174 , 10 , 448 , 456 , 468 ,
\l__tag_para_tag_default_tl 321	469 , 470 , 475 , 481 , 483 , 484 , 485 ,
\l__tag_para_tag_tl	488 , 490 , 491 , 495 , 546 , 547 , 599 , 609
. 321 , 398 , 405 , 413 , 418 , 462 , 492	\g__tag_role_NS_<ns>_class_prop 174
\l__tag_parent_child_check_tl	\g__tag_role_NS_<ns>_prop 174
. 155 , 156 , 168 , 171 , 500 ,	\g__tag_role_NS_mathml_prop 267 , 486
593 , 594 , 601 , 604 , 733 , 734 , 746 , 748	__tag_role_NS_new:nnn
__tag_parenttree_add_objr:nn 176 , 20 , 22 , 30 , 74 , 75 , 76 , 77 , 78 , 80
. 163 , 163 , 491	\g__tag_role_NS_prop
\l__tag_parenttree_content_tl 174 , 9 , 26 , 56 , 181 , 326 , 344 , 786
. 170 , 195 , 207 , 227 , 235 , 256 , 259	\g__tag_role_parent_child_-
\g__tag_parenttree_objr_tl	intarray 390 , 397 , 406 , 421 , 425 , 555
. 162 , 165 , 256	__tag_role_read_namespace:n 337 ,
__tag_pdf_name_e:n 104 , 104	337 , 341 , 342 , 343 , 345 , 347 , 349 , 350
__tag_pdf_object_ref 464	__tag_role_read_namespace:nn
__tag_prop_gput:Nnn 318 , 318 , 339 , 348
. 9 , 29 , 89 , 96 , 98 , 111 ,	__tag_role_read_namespace_-
120 , 121 , 128 , 132 , 146 , 149 , 180 ,	line:nw 255 , 259 , 292 , 328
183 , 191 , 201 , 205 , 217 , 220 , 282 ,	\l__tag_role_role_namespace_-
285 , 314 , 315 , 384 , 1264 , 1400 , 1407	tmpa_tl 12 ,
__tag_prop_item:Nn 9 , 52 , 180 , 187	757 , 778 , 783 , 787 , 790 , 794 , 809
__tag_prop_new:N 9 , 9 ,	\l__tag_role_role_tmpa_tl
11 , 19 , 24 , 32 , 107 , 180 , 180 , 194 , 1064 12 , 756 , 776 , 782 , 802 , 808
__tag_prop_new_linked:N	\g__tag_role_rolemap_prop
. 15 , 17 , 180 , 181 174 , 18 , 144 , 146 , 149 , 158 ,
__tag_prop_show:N 9 , 65 , 180 , 189 , 197	214 , 217 , 220 , 269 , 272 , 385 , 604 , 614
\c__tag_property_mc_clist 121 , 247	\c__tag_role_rule_checkparent_tl
__tag_property_record:nn 156 , 173 , 594 , 734
. 28 , 149 , 149 , 158 , 243 , 477 , 703	\c__tag_role_rules_num_prop
__tag_property_ref_lastpage:nn 391 , 514 , 564
. 83 , 159 , 159 , 160 , 174 , 177 , 439 , 453	\c__tag_role_rules_prop 391 , 395 , 419

\l__tag_role_tag_namespace_tmpa_- tl	12, 755, 807	\l__tag_struct_elem_stash_bool	67, 707, 1126, 1159, 1189
\l__tag_role_tag_namespace_tmpb_- tl	14	__tag_struct_exchange_kid_- command:N	305, 305, 314, 345
\l__tag_role_tag_namespace_tmpb_- tluuuuu%	12	__tag_struct_fill_kid_key:n	136, 315, 315, 447
\l__tag_role_tag_tmpa_tl	12, 754, 775, 801, 806	__tag_struct_format_P:nnN	409
\g__tag_role_tags_class_prop	174, 8, 90, 99, 112, 121, 137, 268	__tag_struct_format_parentnum:nnN	412, 412
\g__tag_role_tags_NS_prop	174, 7, 88, 97, 110, 119, 130, 304, 339, 383, 405, 660, 676, 727, 781, 1233	__tag_struct_format_parentrole:nnN	409, 410
\l__tag_role_tmpa_seq	12	__tag_struct_format_Ref	131
\l__tag_role_update_bool	208, 255, 256, 264, 344, 346	__tag_struct_format_Ref:nnN	416, 416
\c__tag_role_userNS_id_str	175, 59, 80	__tag_struct_format_rolemap:nnN	409, 409
\g__tag_root_default_tl	284	__tag_struct_format_tag:nnN	409, 411
\g__tag_saved_in_mc_bool	645, 654, 669, 681, 699	__tag_struct_get_dict_content:nN	138, 395, 395, 448
__tag_seq_gput_left:Nn	9, 40, 185, 193, 268	__tag_struct_get_id:n	96, 101, 114, 115, 148, 149, 454, 513
__tag_seq_gput_right:Nn	9, 35, 180, 184, 192, 231, 241, 252, 291	__tag_struct_get_role:nnNN	145, 158, 195, 195, 214, 536, 541, 613, 625, 637, 724, 737
__tag_seq_item:Nn ...	9, 47, 180, 186	__tag_struct_gput_data_attribute:nn	1393, 1393
__tag_seq_new:N	9, 9, 22, 109, 180, 182, 195, 1067	__tag_struct_gput_data_ref:nn	1375, 1392
__tag_seq_show:N	9, 58, 180, 188, 196	__tag_struct_gput_data_ref_- aux:nnn	1354, 1355, 1377, 1381, 1385, 1389
__tag_show_spacemark	475	__tag_struct_gput_data_ref_- dest:nn	1383
\l__tag_showspace_bool ...	7, 16, 17	__tag_struct_gput_data_ref_- label:nn	1379
\g__tag_softyphen_bool ...	136, 294	__tag_struct_gput_data_ref_- num:nn	1387
__tag_space_chars_shipout	587	__tag_struct_insert_annot:nn	462, 462, 1417
__tag_start_para_ints:	218, 243, 356, 356	__tag_struct_kid_mc_gput_- right:nn ...	215, 227, 228, 247, 269
__tag_stop_para_ints:	208, 232, 356, 375	__tag_struct_kid_OBJR_gput_- right:nnn ..	280, 280, 283, 304, 478
__tag_store_parent_child_- rule:nnn	391, 393, 417, 462	__tag_struct_kid_struct_gput_- left:nn	264, 264, 265, 279
g__tag_struct_1_prop	106	__tag_struct_kid_struct_gput_- right:nn	248, 248, 249, 263, 1261, 1306
__tag_struct_add_AF:nn	908, 925, 945, 952, 972, 1017	g__tag_struct_kids_1_seq	106
__tag_struct_add_inline_AF:nn	897, 924, 986, 990, 997, 1007	\g__tag_struct_label_num_prop	66, 701, 840
\l__tag_struct_addkid_tl	68, 747, 1176	\l__tag_struct_lang_tl	610, 1050, 1075, 1080
\g__tag_struct_AFobj_int	895, 903, 906	__tag_struct_mcid_dict:n	98, 101, 215, 234
__tag_struct_check_parent_- child:nn	553, 553, 608, 644, 653, 1163		
__tag_struct_check_parent_- child_aux:nnnnN .	528, 529, 588, 596		
\g__tag_struct_cont_mc_prop	11, 95, 96, 98, 101, 244		
\g__tag_struct_dest_num_prop	70, 853		

```

\c__tag_struct_null_tl . . . . . 10, 349
\g__tag_struct_objR_seq . . . . . 8
\__tag_struct_output_prop_aux:nn
    . . . . . 74, 74, 88
\l__tag_struct_parenttag_NS_tl .
    . . . . . 58, 737, 740, 744, 1132
\l__tag_struct_parenttag_tl . . .
    . . . . . 58, 736, 739, 743, 745, 1132
\__tag_struct_prop_gput:nnn . . 92,
    93, 94, 100, 111, 116, 121, 126,
    131, 138, 164, 168, 177, 183, 188,
    351, 364, 378, 759, 771, 785, 801,
    816, 824, 889, 911, 954, 973, 1018,
    1071, 1077, 1082, 1113, 1128, 1141,
    1151, 1167, 1309, 1370, 1480, 1531
\g__tag_struct_ref_by_dest_prop
    . . . . . 73, 74
\__tag_struct_Ref_dest:nN . . 830, 851
\__tag_struct_Ref_label:nN . . 830, 838
\__tag_struct_Ref_num:nN . . . 830, 864
\__tag_struct_Ref_obj:nN . . . 830, 830
\g__tag_struct_roletag_NS_tl . . . 58
\l__tag_struct_roletag_NS_tl . .
    . . . . . 61, 1108, 1117, 1155
\l__tag_struct_roletag_tl . . . .
    . . . . 58, 1107, 1110, 1117, 1119, 1155
\__tag_struct_set_tag_info:nnn .
    . . . . . 159, 161, 175, 194, 1089
\g__tag_struct_stack_current_tl
    . . . . . 16, 29, 38, 69, 75, 103, 148,
    154, 162, 202, 270, 274, 309, 338,
    508, 513, 519, 1111, 1174, 1178,
    1179, 1200, 1218, 1224, 1262, 1268,
    1274, 1280, 1307, 1313, 1319, 1325
\l__tag_struct_stack_parent_-
    tmpa_tl . . 16, 471, 480, 497, 717,
    1087, 1094, 1098, 1137, 1164, 1171,
    1175, 1177, 1180, 1192, 1193, 1201
\g__tag_struct_stack_seq . . . . .
    . . . . . 12, 22, 25, 470, 723,
    736, 1097, 1103, 1112, 1211, 1216, 1222
\c__tag_struct_StructElem_-
    entries_seq . . . . . 21
\c__tag_struct_StructTreeRoot_-
    entries_seq . . . . . 21
\g__tag_struct_tag_NS_tl . . 58, 670,
    686, 689, 693, 1092, 1106, 1199, 1235
\g__tag_struct_tag_stack_seq . .
    . . . . . 14, 50, 248,
    249, 555, 570, 584, 1109, 1215, 1229
\g__tag_struct_tag_tl . . . . . 58,
    179, 180, 183, 314, 315, 401, 402,
    669, 671, 685, 688, 692, 694, 1091,
    1105, 1110, 1231, 1233, 1275, 1320
\__tag_struct_use_check_parent_-
    child:nn . . 609, 609, 656, 1279, 1324
\__tag_struct_write_obj . . . . . 131
\__tag_struct_write_obj:n . . . .
    . . . . . 151, 428, 428
\l__tag_tag_stop_int . . 199, 203, 204,
    212, 213, 220, 227, 228, 237, 238, 246
\g__tag_tagunmarked_bool . . 135, 291, 293
\l__tag_tmp_unused_tl . . 104, 130, 297,
    304, 395, 398, 402, 405, 419, 422,
    660, 663, 676, 679, 727, 730, 788, 1506
\l__tag_tmp_unused_tl_uuuu\l__-
    tag_Ref_tmpa_tl . . . . . 101
\l__tag_tmpa_box . . . . .
    . . . . 101, 171, 177, 178, 182, 193, 194
\l__tag_tmpa_clist . . . . .
    . . . . 101, 1458, 1459, 1492, 1493, 1495
\l__tag_tmpa_int . . . . . 90,
    93, 98, 101, 101, 105, 114, 430, 442, 444
\l__tag_tmpa_prop . . . . .
    . . . . . 101, 176, 189, 203, 205
\l__tag_tmpa_seq . . 51, 58, 59, 101, 319,
    321, 323, 324, 325, 326, 443, 446,
    448, 454, 455, 457, 458, 459, 468,
    478, 662, 666, 669, 670, 678, 682,
    685, 686, 729, 733, 736, 737, 774,
    775, 776, 1460, 1464, 1474, 1475,
    1476, 1478, 1496, 1502, 1504, 1528
\l__tag_tmpa_str . . . . .
    . . . . 42, 43, 48, 101, 256, 261, 266,
    277, 282, 289, 393, 398, 406, 411,
    755, 762, 767, 774, 781, 788, 797,
    804, 812, 819, 819, 820, 825, 885, 892
\l__tag_tmpa_tl . . . . . 41,
    42, 47, 49, 49, 51, 56, 86, 88, 93, 94,
    96, 98, 101, 102, 105, 106, 107, 109,
    112, 113, 116, 118, 119, 137, 138,
    139, 141, 143, 144, 146, 177, 178,
    180, 183, 184, 186, 191, 198, 199,
    205, 205, 206, 209, 211, 214, 215,
    220, 268, 269, 271, 275, 277, 288,
    297, 299, 300, 302, 306, 307, 308,
    308, 308, 311, 314, 327, 339, 347,
    349, 358, 437, 445, 448, 453, 455,
    456, 457, 458, 468, 469, 470, 475,
    481, 483, 488, 514, 516, 525, 546,
    549, 556, 564, 566, 575, 584, 588,
    592, 599, 601, 604, 606, 620, 624,
    645, 653, 655, 656, 658, 662, 667,
    690, 692, 698, 702, 723, 725, 736,
    738, 741, 743, 771, 772, 774, 907,
    910, 1215, 1216, 1222, 1224, 1229,
    1232, 1233, 1235, 1302, 1396, 1398,
    1399, 1403, 1466, 1472, 1483, 1510

```

<code>\l__tag_tmpb_box</code>	<code>\tagmcbegin</code>
..... 101 , 172 , 179 , 180 , 184 , 186	40 , 174 , 22
<code>\l__tag_tmpb_seq</code>	<code>\tagmcend</code>
..... 101 , 1459 , 1460 , 1495 , 1496	40 , 22
<code>\l__tag_tmpb_tl</code>	<code>tagmcid</code>
..... 186 , 89 , 101 , 104 , 118 ,	8 , 163
120 , 295 , 301 , 432 , 456 , 462 , 484 ,	<code>\tagmcifinTF</code>
490 , 518 , 547 , 550 , 556 , 568 , 609 ,	40 , 39
611 , 614 , 616 , 621 , 625 , 672 , 680 ,	<code>\tagmcuse</code>
682 , 683 , 685 , 689 , 691 , 693 , 694 ,	40 , 22
699 , 703 , 742 , 744 , 840 , 844 , 853 , 857	<code>\tagpdfparaOff</code>
<code>\l__tag_tmpc_tl</code>	42 , 601
101 , 485 , 491	<code>\tagpdfparaOn</code>
<code>__tag_tree_fill_parenttree:</code> ...	42 , 601
..... 171 , 172 , 253	<code>\tagpdfsetup</code>
<code>__tag_tree_final_checks:</code> 20 , 20 , 354	40 , 64 , 109 , 173 , 6
<code>\g__tag_tree_id_pad_int</code> .. 78 , 82 , 154	<code>\tagpdfsuppressmarks</code>
<code>__tag_tree_lua_fill_parenttree:</code>	42 , 606
..... 233 , 233 , 250	<code>\tagstart</code>
<code>\g__tag_tree_openaction_struct_-</code>	7 , 223 , 254
<code>tl</code>	<code>\tagstop</code>
32 , 38 , 57	7 , 222 , 253
<code>__tag_tree_parenttree_rerun_-</code>	<code>tagstruct</code>
<code>msg:</code>	8 , 163
171 , 220 , 255	<code>\tagstructbegin</code> 41 , 142 , 173 , 174 , 45 , 287
<code>__tag_tree_update_openaction:</code> ..	<code>\tagstructend</code>
..... 42 , 75	41 , 45 , 288
<code>__tag_tree_write_classmap:</code>	<code>tagstructobj</code>
..... 286 , 286 , 369	8 , 163
<code>__tag_tree_write_idtree:</code> .. 86 , 361	<code>\tagstructuse</code>
<code>__tag_tree_write_namespaces:</code> ..	41 , 45
..... 322 , 322 , 373	<code>\tagtool</code>
<code>__tag_tree_write_parenttree:</code> ..	40 , 13
..... 246 , 246 , 357	<code>tagunmarked (deprecated) (key)</code> ... 1 , 291
<code>__tag_tree_write_rolemap:</code>	<code>test/lang_ (setup-key)</code>
..... 263 , 263 , 365	608
<code>__tag_tree_write_structelements:</code>	TeX and L ^A T _E X 2 _ε commands:
..... 147 , 147 , 377	<code>\@M</code>
<code>__tag_tree_write_structtreeroot:</code>	168
..... 126 , 126 , 381	<code>\@auxout</code>
<code>\g__tag_unique_cnt_int</code>	86
..... 137 , 1039 , 1043 , 1046	<code>\@bsphack</code>
<code>__tag_whatsits:</code>	151
..... 35 , 49 , 54 , 55 , 58 , 295 , 296	<code>\@esphack</code>
<code>tag-namespace_ (rolemap-key)</code>	153
752	<code>\@gobble</code>
<code>tag/check/parent-child</code>	31 , 55
183	<code>\@ifpackageloaded</code>
<code>tag/check/parent-child-end</code>	28 , 562
183	<code>\@kernel@after@foot</code>
<code>tag/struct/1</code> internal commands:	623
<code>__tag/struct/1</code>	<code>\@kernel@after@head</code>
31	621
<code>tag/tree/namespaces</code> internal commands:	<code>\@kernel@before@foot</code>
<code>__tag/tree/namespaces</code>	622
321	<code>\@kernel@before@footins</code> ...
<code>tag/tree/parenttree</code> internal commands:	570 , 572
<code>__tag/tree/parenttree</code>	<code>\@kernel@before@head</code>
154	618 , 620
<code>tag/tree/rolemap</code> internal commands:	<code>\@kernel@tagsupport@@makecol</code> ...
<code>__tag/tree/rolemap</code>	574
262	<code>\@makecol</code>
<code>tagabspage</code>	576 , 587
8 , 163	<code>\@maxdepth</code>
<code>tagmcabs</code>	181
8 , 163	<code>\@outputbox</code>
	577 , 589
	<code>\@secondoftwo</code>
	31 , 55
	<code>\c@chapter</code>
	360 , 378
	<code>\c@page</code>
	576 , 588
	<code>\new@label@record</code>
	88
	<code>\on@line</code>
	474 , 489 , 505
	tex commands:
	<code>\tex_botmarks:D</code>
	91
	<code>\tex_firstmarks:D</code>
	88
	<code>\tex_kern:D</code>
	184
	<code>\tex_marks:D</code> ...
	25 , 34 , 47 , 54 , 65 , 71
	<code>\tex_special:D</code>
	58
	<code>\tex_splitbotmarks:D</code>
	217
	<code>\tex_splitfirstmarks:D</code>
	197
	<code>texsource (key)</code>
	1 , 895
	<code>\the</code>
	576 , 588
	<code>\tiny</code>
	428 , 439
	<code>title (key)</code>
	1 , 697
	<code>title-o (key)</code>
	1 , 697

tl commands:

\c_empty_tl	365, 385	192, 196, 200, 204, 206, 207, 208,
\c_space_tl		212, 216, 218, 235, 236, 238, 239,
..... 55, 56, 58, 60, 80, 100, 104,		243, 244, 249, 250, 271, 275, 296,
116, 167, 191, 197, 198, 216, 218,		302, 306, 308, 332, 334, 336, 347,
220, 222, 259, 299, 388, 405, 425,		383, 390, 401, 437, 508, 516, 518,
453, 576, 588, 834, 844, 857, 868,		552, 560, 566, 568, 601, 606, 611,
935, 1192, 1274, 1319, 1403, 1475, 1521		616, 629, 645, 655, 658, 662, 667,
\tl_clear:N		672, 682, 685, 689, 694, 707, 717,
.. 88, 89, 106, 177, 210, 211, 288, 397		736, 737, 743, 744, 747, 775, 776,
\tl_const:Nn	10	790, 794, 1087, 1259, 1367, 1472, 1500
\tl_count:n	79, 83, 154	\tl_set_eq:NN
\tl_gput_right:Nn	165, 933	179, 314
\tl_gset:Nn	18,	\tl_show:N
33, 38, 103, 219, 237, 285, 297, 330,		1174, 1175, 1524, 1530
367, 384, 384, 669, 670, 685, 686,		\tl_tail:n
692, 693, 940, 1111, 1224, 1231, 1235		506
\tl_gset_eq:NN	180, 315	\tl_to_str:n
\tl_head:N	655, 682 32, 47, 149, 202, 217, 506, 539
\tl_if_empty:NTF		\tl_trim_spaces:n
..... 42, 43, 109, 192, 289, 307,		49
329, 399, 656, 683, 772, 778, 820, 1075		\tl_use:N
\tl_if_empty:nTF 51, 69, 77, 89, 142,		261, 916, 959, 978, 1023
196, 210, 253, 262, 266, 274, 295,		
297, 334, 353, 403, 440, 603, 611,		token commands:
643, 670, 778, 794, 809, 825, 900, 970		\token_to_str:N
\tl_if_empty_p:n	310	88, 576, 587
\tl_if_eq:NNTF	349, 483, 642	tree-mcid-index-wrong
\tl_if_eq:NnTF	107	21, 220
\tl_if_eq:nnTF	212, 274, 278	tree-statistic
\tl_if_exist:NTF ..	259, 337, 388, 928	21, 54
\tl_if_head_eq_charcode:nNTF	49	tree-struct-still-open
\tl_if_in:nnTF	185	21, 47
\tl_new:N 11, 12, 12, 13, 14, 15, 16,		
17, 19, 20, 22, 23, 24, 25, 32, 32, 58,		
59, 60, 61, 62, 64, 68, 101, 102, 103,		
104, 105, 106, 107, 108, 109, 110,		
111, 112, 162, 170, 284, 329, 331,		
333, 335, 338, 339, 500, 938, 1050, 1433		
\tl_put_left:Nn	621, 623	
\tl_put_right:Nn		
..... 94, 104, 118, 195, 207,		
226, 245, 256, 260, 261, 281, 282,		
297, 388, 397, 398, 405, 410, 411,		
414, 418, 423, 572, 574, 620, 622,		
832, 842, 855, 866, 1363, 1519, 1526		
\tl_remove_once:Nn	1398, 1399	
\tl_replace_once:Nnn	308	
\tl_set:Nn	41, 63, 65, 69, 86,	
118, 139, 160, 162, 180, 183, 188,		

U

uncompress (deprecated) (key)	278
unittag _l (deprecated)	393
\unskip	40

use commands:

\use:N	67, 229, 607, 1176
\use:n	41, 348
\use_i:nn	
.. 99, 102, 109, 137, 154, 159, 206,	
238, 365, 385, 572, 583, 588, 592, 1232	
\use_ii:nn	104, 119,
134, 152, 157, 207, 239, 344, 569, 580	
\use_none:n	81, 103, 118, 224
\use_none:nn	80, 1348
\UseExpandableTaggingSocket ..	44, 70, 72
\UseSocket	43, 44
\UseTaggingSocket	43, 44, 69, 72

V

\vbadness	168, 192
-----------------	----------

vbox commands:

\vbox_set_split_to_ht:Nnn	194
\vbox_set_to_ht:Nnn	170
\vbox_unpack_drop:N	183
\vfuzz	169
viewer/startstructure _l (setup-key) ..	34