

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

Ulrike Fischer[†]

Released 2026-08-21

Contents

I	The tagpdf main module	7
1	Initialization and test if pdfmanagement is active.	8
2	base package	9
3	Package options	9
4	Packages	10
	4.1 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	Handling link artifacts	15
12	Keys for tagpdfsetup	16
13	loading of engine/more dependent code	19
II	The tagpdf-checks module	
	Messages and check code	20
1	Commands	20

^{*}This file describes v1.0e, last revised 2026-08-21.

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

2	Description of log messages	21
2.1	\ShowTagging command	21
2.2	Messages in checks and commands	21
2.3	Messages from the ptagging code	22
2.4	Warning messages from the lua-code	22
2.5	Info messages from the lua-code	22
2.6	Debug mode messages and code	23
2.7	Messages	23
3	Messages	24
3.1	Messages related to mc-chunks	24
3.2	Messages related to structures	25
3.3	Attributes	27
3.4	Roles	28
3.5	Miscellaneous	31
4	Retrieving data	32
5	PDF version check	33
6	User conditionals	33
7	Internal checks	34
7.1	checks for active tagging	35
7.2	Checks related to structures	35
7.3	Checks related to roles	36
7.4	Check related to mc-chunks	37
7.5	Checks related to the state of MC on a page or in a split stream	40
7.6	Benchmarks	43
 III The tagpdf-user module		
	Code related to L ^A T _E X 2 _ε user commands and document commands	44
1	Setup commands	44
2	Commands related to mc-chunks	44
3	Commands related to structures	44
4	Debugging	45
5	Extension commands	45
5.1	Fake space	45
5.2	Tagging of paragraphs	46
5.3	Header and footer	46
5.4	Link tagging	47
6	Socket support	47
7	User commands and extensions of document commands	48

8	Setup and preamble commands	48
9	Commands for the mc-chunks	48
10	Commands for the structure	49
11	Tagging Socket support	50
12	Debugging	51
13	Commands to extend document commands	54
13.1	Document structure	55
13.2	Structure destinations	55
13.3	Fake space	55
13.4	Paratagging	56
13.5	Language support	60
13.6	Header and footer	61
13.7	Links	64
13.8	Attaching css-files for derivation	68
IV	The <code>tagpdf-tree</code> module	
	Commands trees and main dictionaries	71
1	Trees, pdfmanagement and finalization code	71
1.1	Check structure	71
1.2	Catalog: MarkInfo and StructTreeRoot and OpenAction	72
1.3	Writing the IDtree	73
1.4	Writing structure elements	74
1.5	ParentTree	75
1.6	Rolemap dictionary	78
1.7	Classmap dictionary	79
1.8	Namespaces	79
1.9	Finishing the structure	80
1.10	StructParents entry for Page	81
V	The <code>tagpdf-mc-shared</code> module	
	Code related to Marked Content (mc-chunks), code shared by all modes	82
1	Public Commands	82
2	Public keys	83
3	Marked content code – shared	84
3.1	Variables and counters	84
3.2	Functions	85
3.3	Keys	89

VI	The <code>tagpdf-mc-generic</code> module	
	Code related to Marked Content (<code>mc-chunks</code>), generic mode	91
1	Marked content code – generic mode	91
1.1	Variables	91
1.2	Functions	92
1.3	Looking at MC marks in boxes	95
1.4	Keys	101
VII	The <code>tagpdf-mc-luacode</code> module	
	Code related to Marked Content (<code>mc-chunks</code>), luamode-specific	104
1	Marked content code – luamode code	104
1.1	Commands	106
1.2	Key definitions	110
VIII	The <code>tagpdf-struct</code> module	
	Commands to create the structure	113
1	Public Commands	113
2	Public keys	115
2.1	Keys for the structure commands	115
2.2	Setup keys	117
3	Variables	117
3.1	Variables used by the keys	120
3.2	Variables used by tagging code of basic elements	120
4	Commands	121
4.1	Initialization of the <code>StructTreeRoot</code>	121
4.2	Adding the <code>/ID</code> key	123
4.3	Filling in the tag info	123
4.4	Handlings kids	124
4.5	Output of the object	128
4.6	Commands for the parent-child checks	134
5	Keys	137
6	User commands	145
7	Attributes and attribute classes	155
7.1	Variables	155
7.2	Commands and keys	156
IX	The <code>tagpdf</code> driver for <code>luatex</code>	159
1	Loading the lua	159

2	User commands to access data	163
3	Logging functions	164
4	Helper functions	166
4.1	Retrieve data functions	166
4.2	Functions to insert the pdf literals	168
5	Function for the real space chars	171
6	Function for the tagging	174
7	Parenttree	179
8	parent-child rules	182
9	Link annotations	184
X	The tagpdf-roles module	
	Tags, roles and namespace code	186
1	Code related to roles and structure names	187
1.1	Variables	187
1.2	Namespaces	189
1.3	Adding a new tag	190
1.3.1	pdf 1.7 and earlier	191
1.3.2	The pdf 2.0 version	193
1.4	Helper command to read the data from files	195
1.5	Reading the default data	197
1.6	Parent-child rules	198
1.6.1	Reading in the csv-files	199
1.6.2	Retrieving the parent-child rule	201
1.7	Key-val user interface	206
XI	The tagpdf-space module	
	Code related to real space chars	209
1	Code for interword spaces	209
	Index	213

Ulrike Fischer
Version 1.0e, released 2026-08-21

Part I

The tagpdf main module

<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 switch 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 commands <code>\SuspendTagging</code> ,
<code>\tagstop</code>	<code>\ResumeTagging</code> , <code>\tag_suspend:n</code> and <code>\tag_resume:n</code> instead.
<code>\tagstart</code>	

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

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

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

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

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

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

`activate-struct-dest` (*deprecated*) (*setup key*)
`debug/log` (*setup key*) The `debug/log` key takes currently the values `none`, `v`, `vv`, `vvv`, `all`. More details are in `tagpdf-checks`.

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

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

- `false` and `off` deactivates the automatic handling.

- **char** Replaces the hyphens by U+00AD if the font contains this glyph. Note that in some newer fonts U+00AD is an invisible character and so this could lead to disappearing hyphens.
- **artifact** (default value) This keeps the hard hyphen U+002D but surrounds it by an Artifact-MC. This only works if the document is tagged.
- **true**, on do the same as **artifact**.

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

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

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

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

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

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

tagstruct
tagstructobj
tagabspage
tagmcabs
tagmcid

These are attributes used by the label/ref system.

1 Initialization and test if pdfmanagement is active.

```

1 <@@=tag>
2 <*package>
3 \ProvidesExplPackage {tagpdf} {2026-08-21} {1.0e}
4   { LaTeX kernel code for PDF tagging }
5
6 \IfPDFManagementActiveF
7   {
```

```

8      \PackageError{tagpdf}
9      {
10         PDF-resource-management~is~not~active!\MessageBreak
11         tagpdf~will~not~work.
12     }
13     {
14         Activate~it~with \MessageBreak
15         \string\DocumentMetadata{<options>}\MessageBreak
16         before~\string\documentclass
17     }
18 }
19 \end{package}

```

<*debug>

```

20 \ProvidesExplPackage {tagpdf-debug} {2026-08-21} {1.0e}
21 { debug code for tagpdf }
22 \@ifpackageloaded{tagpdf}{\PackageWarning{tagpdf-debug}{tagpdf~not~loaded,~quitting}\ending

```

</debug> We map the internal module name “tag” to “tagpdf” in messages.

```

23 <*package>
24 \prop_gput:Nnn \g_msg_module_name_prop { tag }{ tagpdf }
25 </package>

```

Debug mode has its special mapping:

```

26 <*debug>
27 \prop_gput:Nnn \g_msg_module_type_prop { tag / debug }{ }
28 \prop_gput:Nnn \g_msg_module_name_prop { tag / debug }{tagpdf~DEBUG}
29 </debug>

```

2 base package

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

```

30 <*base>
31 \ProvidesExplPackage {tagpdf-base} {2026-08-21} {1.0e}
32 {part of tagpdf - provide base, no-op versions of the user commands }
33 </base>

```

3 Package options

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

```

34 <*package>
35 \bool_new:N\g__tag_mode_lua_bool
36 \sys_if_engine luatex:T {\bool_gset_true:N \g__tag_mode_lua_bool}
37 \DeclareOption {luamode} { }
38 \DeclareOption {genericmode}{ }
39 \ProcessOptions

```

4 Packages

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

```
40 \RequirePackage{tagpdf-base}
41 \end{package}
```

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

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

4.1 a LastPage label

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

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

5 Variables

A few temporary variables

```
\l__tag_tmpa_tl
\l__tag_tmpb_tl
\l__tag_tmpc_tl
\l__tag_tmp_unused_tl \l__tag_get_tmpc_tl
\l__tag_get_parent_tmpa_tl
\l__tag_get_parent_tmpb_tl
\l__tag_get_parent_tmpc_tl
\l__tag_get_child_tmpa_tl
\l__tag_get_child_tmpb_tl
\l__tag_get_child_tmpc_tl
\l__tag_tmpa_str
\l__tag_tmpa_prop
\l__tag_tmpa_seq
\l__tag_tmpb_seq
\l__tag_tmpa_clist
\l__tag_tmpa_int
\l__tag_tmpa_box
\l__tag_tmpb_box
```

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

```

76 \int_new:N \l__tag_tmpa_int
77 \box_new:N \l__tag_tmpa_box
78 \box_new:N \l__tag_tmpb_box

```

(End of definition for \l__tag_tmpa_tl and others.)

Attribute lists for the label command. We have a list for mc-related labels, and one for structures.

```

\c__tag_property_mc_clist
\c__tag_property_struct_clist
79 \clist_const:Nn \c__tag_property_mc_clist {tagabspage,tagmcabs,tagmcid}
80 \clist_const:Nn \c__tag_property_struct_clist {tagstruct,tagstructobj}

```

(End of definition for \c__tag_property_mc_clist and \c__tag_property_struct_clist.)

\l__tag_loglevel_int This integer hold the log-level and so allows to control the messages. TODO: a list which log-level shows what is needed. The current behaviour is quite ad-hoc.

```

81 \int_new:N \l__tag_loglevel_int

```

(End of definition for \l__tag_loglevel_int.)

```

\g__tag_active_space_bool
\g__tag_active_mc_bool
\g__tag_active_tree_bool
\g__tag_active_struct_bool
\g__tag_active_struct_dest_bool

```

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

```

82 \bool_new:N \g__tag_active_space_bool
83 \bool_new:N \g__tag_active_mc_bool
84 \bool_new:N \g__tag_active_tree_bool
85 \bool_new:N \g__tag_active_struct_bool
86 \bool_new:N \g__tag_active_struct_dest_bool
87 \bool_gset_true:N \g__tag_active_struct_dest_bool

```

(End of definition for \g__tag_active_space_bool and others.)

```

\l__tag_active_mc_bool
\l__tag_active_struct_bool
\l__tag_active_socket_bool

```

These booleans should help to control the *local* behaviour of tagpdf. In some cases it could e.g. be necessary to stop tagging completely. As local booleans they respect groups. TODO: check if they are used everywhere as needed and as wanted.

```

88 \bool_new:N \l__tag_active_mc_bool
89 \bool_set_true:N \l__tag_active_mc_bool
90 \bool_new:N \l__tag_active_struct_bool
91 \bool_set_true:N \l__tag_active_struct_bool
92 \bool_new:N \l__tag_active_socket_bool

```

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

```

\g__tag_tagunmarked_bool

```

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

```

93 \bool_new:N \g__tag_tagunmarked_bool

```

(End of definition for `\g__tag_tagunmarked_bool`.)

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

```

94 \bool_new:N \g__tag_softhyphen_bool
95 \bool_gset_true:N \g__tag_softhyphen_bool

```

(End of definition for `\g__tag_softhyphen_bool`.)

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

```

96 \int_new:N \g__tag_unique_cnt_int

```

(End of definition for `\g__tag_unique_cnt_int`.)

6 Variants of l3 commands

```

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

```

7 Label and Reference commands

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

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

```

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

```

And a few variants

```

115 \cs_generate_variant:Nn \property_ref:nnn {enn}
116 \cs_generate_variant:Nn \property_ref:nn {en}
117 \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.

```

118 \cs_new:Npn \_tag\_property\_ref\_lastpage:nn #1 #2
119 {
120   \property\_ref:nnn {@tag@LastPage}{#1}{#2}
121 }

```

(End of definition for `\_tag\_property\_ref\_lastpage:nn`.)

8 Setup label attributes

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

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

```

122 \property\_new:nnnn
123   { tagstruct } { now }
124   {1} { \int\_use:N \c@g\_tag\_struct\_abs\_int }
125 \property\_new:nnnn { tagstructobj } { now } {}
126 {
127   \pdf\_object\_ref\_indexed:nn { \_tag/struct } { \c@g\_tag\_struct\_abs\_int }
128 }
129 \property\_new:nnnn
130   { tagabspage } { shipout }
131   {0} { \int\_use:N \g\_shipout\_readonly\_int }
132 \property\_new:nnnn { tagmcabs } { now }
133   {0} { \int\_use:N \c@g\_tag\_MCID\_abs\_int }
134
135 \flag\_new:n { \_tag/mcid }
136 \property\_new:nnnn { tagmcid } { shipout }
137   {0} { \flag\_height:n { \_tag/mcid } }
138

```

(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.

<code>_tag_prop_new:N</code>		
<code>_tag_prop_new_linked:N</code>	139 <code>\cs_set_eq:NN _tag_prop_new:N</code>	<code>\prop_new:N</code>
<code>_tag_seq_new:N</code>	140 <code>\cs_set_eq:NN _tag_prop_new_linked:N</code>	<code>\prop_new_linked:N</code>
<code>_tag_prop_gput:Nnn</code>	141 <code>\cs_set_eq:NN _tag_seq_new:N</code>	<code>\seq_new:N</code>
<code>_tag_seq_gput_right:Nn</code>	142 <code>\cs_set_eq:NN _tag_prop_gput:Nnn</code>	<code>\prop_gput:Nnn</code>
<code>_tag_seq_item:cn</code>	143 <code>\cs_set_eq:NN _tag_seq_gput_right:Nn</code>	<code>\seq_gput_right:Nn</code>
<code>_tag_prop_item:cn</code>	144 <code>\cs_set_eq:NN _tag_seq_gput_left:Nn</code>	<code>\seq_gput_left:Nn</code>
<code>_tag_seq_show:N</code>	145 <code>\cs_set_eq:NN _tag_seq_item:cn</code>	<code>\seq_item:cn</code>
<code>_tag_prop_show:N</code>	146 <code>\cs_set_eq:NN _tag_prop_item:cn</code>	<code>\prop_item:cn</code>

```

147 \cs_set_eq:NN \__tag_seq_show:N \seq_show:N
148 \cs_set_eq:NN \__tag_prop_show:N \prop_show:N
149 % cnx temporary needed for latex-lab-graphic code
150 \cs_generate_variant:Nn \__tag_prop_gput:Nnn { Nen, Nee, Nne, Nno, cnn, cen, cne, cno }
151 \cs_generate_variant:Nn \__tag_seq_gput_right:Nn { Ne , No, cn, ce }
152 \cs_generate_variant:Nn \__tag_seq_gput_left:Nn { ce }
153 \cs_generate_variant:Nn \__tag_prop_new:N { c }
154 \cs_generate_variant:Nn \__tag_seq_new:N { c }
155 \cs_generate_variant:Nn \__tag_seq_show:N { c }
156 \cs_generate_variant:Nn \__tag_prop_show:N { c }
157 \end{package}

```

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

10 General tagging commands

\tag_suspend:n We need commands to stop tagging in some places. They switch local booleans and also stop the counting of paragraphs. The commands keep track of the nesting with a local counter. Tagging only is only restarted at the outer level, if the current level is 1. The **\tag_resume:n** 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_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.

```

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

```

```

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

```

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

```

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

```

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

11 Handling link artifacts

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

```

219 <*package>
220 \cs_new_protected:Npn \__tag_activate_linkbin:
221 {

```

```

222 \sys_if_engine luatex:T
223 {
224   \tagstructbegin{tag=\UseStructureName{link},stash}
225   \tl_const:Nc \c__tag_struct_link_bin_tl {\int_use:N\c@g__tag_struct_abs_int}%
226   \tagstructend
227   \AddToHook{tagpdf/finish/before}[tagpdf/linkbin]
228   {
229     \par
230     \int_compare:nNnT {\directlua{tex.sprint{ltx.__tag.func.linkbin()}}}>{0}
231     {
232       \tagstructbegin{tag=Artifact,title=Ignored~link~annotations}
233       \tag_struct_use_num:o{\c__tag_struct_link_bin_tl}
234       \tagstructend
235     }
236   }
237 }
238 }
239 \AddToHook{begindocument/end}[tagpdf/linkbin]{\__tag_activate_linkbin:}
240 \end{package}

```

12 Keys for tagpdfsetup

TODO: the log-levels must be sorted

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

`activate/all` (*setup key*) 241 `\keys_define:nn { __tag / setup }`

`activate/struct-dest` (*setup key*) 242 `{`

```

243 {
244   activate/mc      .bool_gset:N = \g__tag_active_mc_bool,
245   activate/tree    .bool_gset:N = \g__tag_active_tree_bool,
246   activate/struct .bool_gset:N = \g__tag_active_struct_bool,
247   activate/all     .meta:n =
248     {activate/mc={#1},activate/tree={#1},activate/struct={#1}},
249   activate/all     .default:n = true,
250   activate/struct-dest .bool_gset:N = \g__tag_active_struct_dest_bool,

```

old, deprecated names

```

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

```

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

```

258   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`.

`log` (deprecated) (*setup key*) 259 `debug/log` `.choice:,`

`uncompress` (deprecated) (*setup key*)

```

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

```

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

```

270 log .meta:n = {debug/log={#1}},
271 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.

```

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

```

deprecated name

```

274 tagunmarked .bool_gset:N = \g__tag_tagunmarked_bool,

```

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

```

275 activate/softhyphen .choice:,
276 activate/softhyphen/false .code:n = {\bool_gset_false:N \g__tag_softhyphen_bool},
277 activate/softhyphen/off .code:n = {\bool_gset_false:N \g__tag_softhyphen_bool},
278 activate/softhyphen/char .code:n =
279   {
280     \bool_gset_true:N \g__tag_softhyphen_bool
281     \sys_if_engine luatex:T
282     {
283       \lua_now:e
284       {
285         tex.setattribute
286         (
287           luatexbase.attributes.l__tag_softhyphen_attr,
288           1
289         )
290       }
291     }
292   },
293 activate/softhyphen/artifact .code:n =
294   {
295     \bool_gset_true:N \g__tag_softhyphen_bool
296     \sys_if_engine luatex:T
297     {
298       \lua_now:e
299       {
300         tex.setattribute
301         (
302           luatexbase.attributes.l__tag_softhyphen_attr,
303           -2147483647
304         )

```

```

305     }
306   }
307 },
308 activate/softhyphen/true .code:n =
309 {
310   \bool_gset_true:N \g__tag_softhyphen_bool
311   \sys_if_engine_luatex:T
312   {
313     \lua_now:e
314     {
315       tex.setattribute
316       (
317         luatexbase.attributes.l__tag_softhyphen_attr,
318         -2147483647
319       )
320     }
321   }
322 },
323 activate/softhyphen/on .code:n =
324 {
325   \bool_gset_true:N \g__tag_softhyphen_bool
326   \sys_if_engine_luatex:T
327   {
328     \lua_now:e
329     {
330       tex.setattribute
331       (
332         luatexbase.attributes.l__tag_softhyphen_attr,
333         -2147483647
334       )
335     }
336   }
337 },
338 activate/softhyphen .default:n = artifact,

```

collect-link-artifacts (*setup key*)

```

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

```

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

```

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

```

deprecated name

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

13 loading of engine/more dependent code

```
354 \sys_if_engine luatex:T  
355 {  
356   \file_input:n {tagpdf-luatex.def}  
357 }  
358 </package>  
  
359 <*mcloding>  
360 \bool_if:NTF \g__tag_mode_lua_bool  
361 {  
362   \RequirePackage {tagpdf-mc-code-lua}  
363 }  
364 {  
365   \RequirePackage {tagpdf-mc-code-generic} %  
366 }  
367 </mcloding>  
368 <*debug>  
369 \bool_if:NTF \g__tag_mode_lua_bool  
370 {  
371   \RequirePackage {tagpdf-debug-lua}  
372 }  
373 {  
374   \RequirePackage {tagpdf-debug-generic} %  
375 }  
376 </debug>
```

Ulrike Fischer
Version 1.0e, released 2026-08-21

Part II

The tagpdf-checks module

Messages and check code

1 Commands

<code>\tag_if_active_p: *</code>	This command tests if tagging is active. It only gives true if all tagging has been activated,
<code>\tag_if_active:TF *</code>	and if tagging hasn't been stopped locally.

<code>\tag_if_struct_exist_p:n *</code>	<code>\tag_if_struct_exist:n {<structure number>}</code>
<code>\tag_if_struct_exist:nTF *</code>	This command tests if a number maps to a structure.

<code>\tag_get:n *</code>	<code>\tag_get:n {<keyword>}</code>
---------------------------	---

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

- `mc_tag` is the tag name of the current mc-chunk.
- `struct_tag` is the tag name of the current structure (without rolemapping). It is retrieved from the property and is a string! This is rather slow and `\tag_get:nN` should be preferred.
- `struct_id` is the id (a string ID.07, where the number is the structure number padded with zeros).
- `struct_num` is the current number as stored in `\g_@@_struct_stack_current_tl`.
- `struct_counter` the state of the absolute structure counter, `\c@g_@@_struct_abs_int`.
- `mc_counter` the state of the absolute mc counter, `\c@g_@@_MCID_abs_int`.
- `current_Sect` (defined in latex-lab-sec) gives back the number of current Sect structure, it is e.g. used by the footnote code.

<code>\tag_get:nN</code>	<code>\tag_get:nN {<keyword>}{<tl var>}</code>
--------------------------	--

This is a generic command to retrieve data for the current structure or mc-chunk. Unlike `\tag_get:n` it is not expandable but stores the data in the `<tl var>`. Often this will be faster. Currently the supported values for the argument `<keyword>` are `struct_tag`, `struct_num` and `C`.

- `struct_tag` is the tag name of the current structure (without rolemapping, retrieved from the tag stack).
- `struct_num` is the current number as stored in number stack.
- `C` is a comma list of the current attribute classes values.

<code>\tag_get:nnN</code>	<code>\tag_get:nnN {<number>} {<keyword>} {<t1 var>}</code>
---------------------------	---

This is a generic command to retrieve data for the structure `{<number>}`. See `\tag_get:nN` for a description of the keywords.

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

This tests if a box contains tagging commands. It relies currently on that the code, that saved the box, correctly sets the command `\l_tag_box_int_use:N #1_t1` to a positive value. The LaTeX commands will do that automatically at some time but it is in the responsibility of the user to ensure that when using low-level code. If the internal command doesn't exist the box is assumed to be untagged.

2 Description of log messages

2.1 \ShowTagging command

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

<code>\tag_if_in_p:n *</code>	<code>\tag_if_in:n {<standard structure type>}</code>
-------------------------------	---

`\tag_if_in:nTF *` This command tests if we are currently “in” a specific structure by checking if the argument is on the stack. The argument must be a standard structure type as no role mapping is applied.

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

2.2 Messages in checks and commands

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

2.3 Messages from the ptagging code

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

2.4 Warning messages from the lua-code

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

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

2.5 Info messages from the lua-code

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

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

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

2.6 Debug mode messages and code

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

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

2.7 Messages

mc-nested
mc-tag-missing
mc-label-unknown
mc-used-twice
mc-not-open
mc-pushed
mc-popped
mc-current

Various messages related to mc-chunks. TODO document their meaning.

struct-unknown
struct-no-objnum
struct-orphan
struct-faulty-nesting
struct-missing-tag
struct-used-twice
struct-label-unknown
struct-show-closing

Various messages related to structure. Check the definition in the code for their meaning and the arguments they take.

tree-struct-still-open

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

tree-statistic

Message issued at the end of the compilation showing the number of objects to write

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

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

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

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

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

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

```

1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-checks-code} {2026-08-21} {1.0e}
4 {part of tagpdf - code related to checks, conditionals, debugging and messages}
5 </header>
```

3 Messages

3.1 Messages related to mc-chunks

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

```

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

(End of definition for `mc-nested`. This function is documented on page 23.)

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 23.)

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-already-been-used.\\  
11   Either~rerun-or-remove-one-of~the~uses. }
```

(End of definition for mc-label-unknown. This function is documented on page 23.)

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-already-been-used }
```

(End of definition for mc-used-twice. This function is documented on page 23.)

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 23.)

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 23.)

mc-current Informational messages about current mc state.

```
16 \msg_new:nnn { tag } {mc-current}  
17 { current~MC:~  
18   \bool_if:NTF\g__tag_in_mc_bool  
19   {abscnt=\\_tag_get_mc_abs_cnt:,~tag=\g__tag_mc_key_tag_tl}  
20   {no~MC~open,~current~abscnt=\\_tag_get_mc_abs_cnt:"}  
21 }
```

(End of definition for mc-current. This function is documented on page 23.)

3.2 Messages related to structures

struct-unknown if for example a parent key value points to structure that doesn't exist (yet)

```
22 \msg_new:nnn { tag } {struct-unknown}  
23 { structure~with~number~#1~doesn't~exist\\ #2 }
```

(End of definition for struct-unknown. This function is documented on page 23.)

struct-no-objnum Should not happen ...

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

(End of definition for struct-no-objnum. This function is documented on page 23.)

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

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

(End of definition for `struct-orphan`. This function is documented on page 23.)

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

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

(End of definition for `struct-faulty-nesting`. This function is documented on page 23.)

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

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

(End of definition for `struct-missing-tag`. This function is documented on page 23.)

struct-used-twice

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

(End of definition for `struct-used-twice`. This function is documented on page 23.)

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

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

(End of definition for `struct-label-unknown`. This function is documented on page 23.)

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

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

(End of definition for `struct-show-closing`. This function is documented on page 23.)

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

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

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

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

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

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

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

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

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

The following messages are only needed in debug mode.

show-struct 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 24.)

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 24.)

3.4 Roles

role-missing	Warning message if either the tag or the role is missing
role-unknown	<pre> 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} </pre>
role-unknown-tag	
role-unknown-NS	
	<i>(End of definition for role-missing and others. These functions are documented on page 24.)</i>
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. <pre> 90 \msg_new:nnn { tag } {role-parent-child-check} 91 { Checking~Parent-Child~'#1'--->~'#2' } </pre> <i>(End of definition for role-parent-child-check. This function is documented on page ??.)</i>
role-parent-child-result	This is info and warning message about the containment rules between child and parent tags. <pre> 92 \msg_new:nnn { tag } {role-parent-child-result} 93 { Parent-Child~'#1'--->~'#2'.\\Relation~is~#3~\msg_line_context:} </pre> <i>(End of definition for role-parent-child-result. This function is documented on page 24.)</i>
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) <pre> 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 } </pre> <i>(End of definition for role-struct-parent-child-forbidden. This function is documented on page ??.)</i>
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, <pre> 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 } </pre> <i>(End of definition for role-MC-child-forbidden. This function is documented on page ??.)</i>

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

```

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

```

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

_tag_check_forbidden_parent_child:nnnn

```

120 \cs_new_protected:Npn \_tag_check_forbidden_parent_child:nnnn #1#2#3#4
121 % #1 check number, #2 number of parent struct
122 % #3 parent info, #4 child info
123 {
124   \int_compare:nNnT {#1} < {0}
125   {
126     \msg_warning:nneee
127     { tag }
128     {role-parent-child-forbidden}
129     { #2 }
130     { #3 }
131     { #4 }
132   }
133 }
134 \cs_generate_variant:Nn \_tag_check_forbidden_parent_child:nnnn {nnee}
135
136 % new with structure numbers:
137 \cs_new_protected:Npn \_tag_check_struct_forbidden_parent_child:nnn #1#2#3
138 % #1 check number,
139 % #2 number of parent struct
140 % #3 number of child struct
141 {
142   \int_compare:nNnT {#1} < {0}
143   {
144     \prop_get:cnN {g__tag_struct_#2_prop}{parentrole}\l__tag_get_parent_tmpc_tl
145     \prop_get:cnN {g__tag_struct_#3_prop}{rolemap}\l__tag_get_child_tmpc_tl
146     \msg_warning:nneeee
147     { tag }
148     {role-struct-parent-child-forbidden}
149     { #2 }
150     { #3 }
151     {
152       \exp_last_unbraced:No \use_ii:nn { \l__tag_get_parent_tmpc_tl }
153       :
154       \exp_last_unbraced:No \use_i:nn { \l__tag_get_parent_tmpc_tl }
155     }
156     {
157       \exp_last_unbraced:No \use_ii:nn { \l__tag_get_child_tmpc_tl }
158       :

```

```

159         \exp_last_unbraced:No \use_i:nn { \l__tag_get_child_tmpc_tl }
160     }
161 }
162 }
163 \cs_generate_variant:Nn\__tag_check_struct_forbidden_parent_child:nnn{onn}
(End of definition for \__tag_check_forbidden_parent_child:nnnn.)

```

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

```

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

```

__tag_check_unresolved_parent_child:nnnn

```

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

```

tag/check/parent-child **tag/check/parent-child-end** Sockets 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 24.)

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 24.)

3.5 Miscellaneous

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

```

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

```

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

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

```

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

```

(End of definition for tree-mcid-index-wrong. This function is documented on page 24.)

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

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

(End of definition for sys-no-interwordspace. This function is documented on page 24.)

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

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

(End of definition for __tag_check_typeout_v:n.)

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

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

(End of definition for para-hook-count-wrong. This function is documented on page 24.)

4 Retrieving data

\tag_get:n This retrieves some data. This is a generic command to retrieve data. Currently the supported values for the argument are mc_tag, struct_tag, struct_id, struct_num, struct_counter, mc_counter, current_Sect.

```
235 <base>\cs_new:Npn \tag_get:n #1 { \use:c {__tag_get_data_#1: } }
```

(End of definition for \tag_get:n. This function is documented on page 20.)

\tag_get:nN This retrieves some data. This is a generic command to retrieve data which are then stored in the given tl_var. Currently the supported values for the argument are struct_tag, struct_num and struct_C.

```
236 <base>\cs_new_protected:Npn \tag_get:nN #1#2
237 <base> { \cs_if_exist_use:cT {__tag_get_data_#1:N } #2 }
```

(End of definition for \tag_get:nN. This function is documented on page 20.)

\tag_get:nnN This retrieves some data. This is a generic command to retrieve data from a specific structure which are then stored in the given tl_var. Currently the supported values for the argument are struct_C.

```
238 <base>\cs_new_protected:Npn \tag_get:nnN #1#2#3
239 <base> { \cs_if_exist_use:cT {__tag_get_data_#2:nN }{{#1} #3}}
```

(End of definition for \tag_get:nnN. This function is documented on page 21.)

5 PDF version check

```

240 <*package>
241 \tl_const:Nc\c__tag_check_pdfversion_tl {\pdf_version:}
242 \AddToHook{begindocument/before}
243 {
244   \tl_if_eq:eeF{\c__tag_check_pdfversion_tl}{\pdf_version:}
245   {
246     \msg_error:nnee {tag}{wrong-pdfversion}{\c__tag_check_pdfversion_tl}{\pdf_version:}
247     \pdf_version_gset:e {\c__tag_check_pdfversion_tl}
248     \pdf_object_unnamed_write:nn{dict}{}
249   }
250 }
251 </package>

```

6 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

```

252 <*base>
253 \cs_if_exist:Nf\tag_if_active:T
254 {
255   \prg_new_conditional:Npnn \tag_if_active: { p , T , TF, F }
256   { \prg_return_false: }
257 }
258 </base>
259 <*package>
260 \prg_set_conditional:Npnn \tag_if_active: { p , T , TF, F }
261 {
262   \bool_lazy_all:nTF
263   {
264     {\g__tag_active_struct_bool}
265     {\g__tag_active_mc_bool}
266     {\g__tag_active_tree_bool}
267     {\l__tag_active_struct_bool}
268     {\l__tag_active_mc_bool}
269   }
270   {
271     \prg_return_true:
272   }
273   {
274     \prg_return_false:
275   }
276 }
277 </package>

```

(End of definition for \tag_if_active:TF. This function is documented on page 20.)

\tag_if_struct_exist_p:n this allows to test if a number belongs to a structure.

\tag_if_struct_exist:n TF

```

278 \prg_set_conditional:Npnn \tag_if_struct_exist:n #1 { p , T , TF, F }
279 {
280   \prop_if_exist:cTF { g__tag_struct_#1_prop }
281   {

```

```

282         \prg_return_true:
283     }
284     {
285         \prg_return_false:
286     }
287 }

```

(End of definition for `\tag_if_struct_exist:nTF`. This function is documented on page 20.)

`\tag_if_box_tagged_p:N`
`\tag_if_box_tagged:NTF`

This tests if a box contains tagging commands. It relies on that the code that saved the box correctly set `\l_tag_box_<box number>_tl` to a positive value. The LaTeX commands will do that automatically at some time but it is in the 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.

```

288 <*base>
289 \prg_new_conditional:Npnn \tag_if_box_tagged:N #1 {p,T,F,TF}
290 {
291     \tl_if_exist:cTF {l_tag_box_\int_use:N #1_tl}
292     {
293         \int_compare:nNnTF {0\tl_use:c{l_tag_box_\int_use:N #1_tl}}>{0}
294         { \prg_return_true: }
295         { \prg_return_false: }
296     }
297     {
298         \prg_return_false:
299         % warning??
300     }
301 }
302 </base>

```

(End of definition for `\tag_if_box_tagged:NTF`. This function is documented on page 21.)

`\tag_if_in:n`

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

```

303 <*package>
304 \prg_new_protected_conditional:Npnn\tag_if_in:n #1 {T,F,TF}
305 {
306     \seq_if_in:NnTF \g__tag_struct_roletag_stack_seq {#1}
307     {\prg_return_true:}
308     {\prg_return_false:}
309 }
310 </package>

```

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

7 Internal checks

These are checks used in various places in the code.

7.1 checks for active tagging

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

```

311 (*package)
312 \prg_new_conditional:Npnn \__tag_check_if_active_mc: {T,F,TF}
313 {
314   \bool_lazy_and:nnTF { \g__tag_active_mc_bool } { \l__tag_active_mc_bool }
315   {
316     \prg_return_true:
317   }
318   {
319     \prg_return_false:
320   }
321 }
322 \prg_new_conditional:Npnn \__tag_check_if_active_struct: {T,F,TF}
323 {
324   \bool_lazy_and:nnTF { \g__tag_active_struct_bool } { \l__tag_active_struct_bool }
325   {
326     \prg_return_true:
327   }
328   {
329     \prg_return_false:
330   }
331 }

```

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

7.2 Checks related to structures

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

```

332 \cs_new_protected:Npn \__tag_check_structure_has_tag:n #1 %#1 struct num
333 {
334   \prop_get:cnNF
335   { g__tag_struct_#1_prop }
336   {S}
337   \l__tag_tmp_unused_tl
338   {
339     \msg_error:nn { tag } {struct-missing-tag}
340   }
341 }

```

(End of definition for `\__tag_check_structure_has_tag:n`.)

`\__tag_check_structure_tag:N` This checks if the name of the tag is known, either because it is a standard type or has been rolemapped. This always used with commands, so the argument is N.

```

342 \cs_new_protected:Npn \__tag_check_structure_tag:N #1
343 {
344   \prop_get:NoNF \g__tag_role_tags_NS_prop {#1}\l__tag_tmp_unused_tl
345   {
346     \msg_warning:nne { tag } {role-unknown-tag} {#1}
347   }
348 }

```

(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.

```

349 \cs_new_protected:Npn \__tag_check_info_closing_struct:n #1 %#1 struct num
350 {
351   \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
352   {
353     \msg_info:nnn { tag } {struct-show-closing} {#1}
354   }
355 }
356
357 \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.

```

358 \cs_new_protected:Npn \__tag_check_no_open_struct:
359 {
360   \msg_error:nn { tag } {struct-faulty-nesting}
361 }

```

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

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

```

362 \cs_new_protected:Npn \__tag_check_struct_used:n #1 %#1 label
363 {
364   \prop_get:cnNT
365     {g__tag_struct_\property_ref:enn{tagpdfstruct-#1}{tagstruct}{unknown}_prop}
366     {parentnum}
367     \l__tag_tmpa_tl
368     {
369       \msg_warning:nnn { tag } {struct-used-twice} {#1}
370     }
371 }

```

(End of definition for `\__tag_check_struct_used:n`.)

7.3 Checks related to roles

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

```

372 \cs_new_protected:Npn \__tag_check_add_tag_role:nn #1 #2 %#1 tag, #2 role
373 {
374   \tl_if_empty:nTF {#2}
375   {
376     \msg_error:nnn { tag } {role-missing} {#1}
377   }
378   {
379     \prop_get:NnNTF \g__tag_role_tags_NS_prop {#2} \l__tag_tmpa_tl
380     {
381       \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
382       {
383         \msg_info:nnnn { tag } {role-tag} {#1} {#2}
384       }
385     }
386   }

```

```

385     }
386     {
387         \msg_error:nnn { tag } {role-unknown} {#2}
388     }
389 }
390 }

```

Similar with a namespace

```

391 \cs_new_protected:Npn \__tag_check_add_tag_role:nnn #1 #2 #3 %#1 tag/NS, #2 role #3 namespace
392 {
393     \tl_if_empty:nTF {#2}
394     {
395         \msg_error:nnn { tag } {role-missing} {#1}
396     }
397     {
398         \prop_get:cnTF { g__tag_role_NS_#3_prop } {#2} \l__tag_tmpa_tl
399         {
400             \int_compare:nNtT {\l__tag_loglevel_int} > { 0 }
401             {
402                 \msg_info:nnnn { tag } {role-tag} {#1} {#2/#3}
403             }
404         }
405         {
406             \msg_error:nnn { tag } {role-unknown} {#2/#3}
407         }
408     }
409 }

```

(End of definition for __tag_check_add_tag_role:nn.)

7.4 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).

```

410 \cs_new_protected:Npn \__tag_check_mc_if_nested:
411 {
412     \__tag_mc_if_in:T
413     {
414         \msg_warning:nne { tag } {mc-nested} { \__tag_get_mc_abs_cnt: }
415     }
416 }
417
418 \cs_new_protected:Npn \__tag_check_mc_if_open:
419 {
420     \__tag_mc_if_in:F
421     {
422         \msg_warning:nne { tag } {mc-not-open} { \__tag_get_mc_abs_cnt: }
423     }
424 }

```

(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.

```

425 \cs_new_protected:Npn \__tag_check_mc_pushed_popped:nn #1 #2
426 {
427   \int_compare:nNnT
428     { \l__tag_loglevel_int } = { 2 }
429     { \msg_info:nne {tag}{mc-#1}{#2} }
430   \int_compare:nNnT
431     { \l__tag_loglevel_int } > { 2 }
432     {
433       \msg_info:nne {tag}{mc-#1}{#2}
434       \seq_log:N \g__tag_mc_stack_seq
435     }
436 }

```

(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.

```

437 \cs_new_protected:Npn \__tag_check_mc_tag:N #1 % #1 is var with a tag name in it
438 {
439   \tl_if_empty:NTF #1
440   {
441     \tl_set:No #1 { \g__tag_struct_tag_tl }
442     \msg_info:nnee { tag } {mc-tag-missing} { \g__tag_struct_tag_tl } { \__tag_get_mc_abs_
443   }
444   {
445     \prop_get:NoNF \g__tag_role_tags_NS_prop {#1}\l__tag_tmp_unused_tl
446     {
447       \msg_warning:nne { tag } {role-unknown-tag} {#1}
448     }
449   }
450 }

```

(End of definition for __tag_check_mc_tag:N.)

`\g__tag_check_mc_used_intarray`
`\__tag_check_init_mc_used:` This variable holds the list of used mc numbers. Every time we store a mc-number we will add one to 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??

```

451 \cs_new_protected:Npn \__tag_check_init_mc_used:
452 {
453   \intarray_new:Nn \g__tag_check_mc_used_intarray { 65536 }
454   \cs_gset_eq:NN \__tag_check_init_mc_used: \prg_do_nothing:
455 }

```

(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.

```

456 \cs_new_protected:Npn \_tag\_check\_mc\_used:n #1 %#1 mcid absent
457 {
458   \int_compare:nNnT {\l__tag_loglevel_int} > { 2 }
459   {
460     \_tag\_check\_init\_mc\_used:
461     \intarray_gset:Nnn \g__tag\_check\_mc\_used\_intarray
462       {#1}
463       { \intarray_item:Nn \g__tag\_check\_mc\_used\_intarray {#1} + 1 }
464     \int_compare:nNnT
465       {
466         \intarray_item:Nn \g__tag\_check\_mc\_used\_intarray {#1}
467       }
468       >
469       { 1 }
470       {
471         \msg_warning:nnn { tag } {mc-used-twice} {#1}
472       }
473   }
474 }

```

(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.

```

475 \cs_new_protected:Npn \_tag\_check\_show\_MCID\_by\_page:
476 {
477   \tl_set:Ne \l__tag_tmpa_tl
478   {
479     \_tag\_property\_ref\_lastpage:nn
480     {abspage}
481     {-1}
482   }
483   \int_step_inline:nnnn {1}{1}
484   {
485     \l__tag_tmpa_tl
486   }
487   {
488     \seq_clear:N \l__tag_tmpa_seq
489     \int_step_inline:nnnn
490       {1}
491       {1}
492       {
493         \_tag\_property\_ref\_lastpage:nn
494         {tagmcabs}
495         {-1}
496       }
497   }
498   \int_compare:nT
499     {
500       \property\_ref:enn
501       {mcid-###1}
502       {tagabspage}
503       {-1}
504     }

```

```

505         ##1
506     }
507     {
508         \seq_gput_right:Ne \l__tag_tmpa_seq
509         {
510             Page##1-####1-
511             \property_ref:enn
512             {mcid-####1}
513             {tagmcid}
514             {-1}
515         }
516     }
517 }
518 \seq_show:N \l__tag_tmpa_seq
519 }
520 }

```

(End of definition for __tag_check_show_MCID_by_page:.)

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

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

__tag_check_mc_in_galley_p: 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.

```

521 \prg_new_conditional:Npnn \__tag_check_if_mc_in_galley: { T,F,TF }
522 {
523     \tl_if_eq:NNTF \l__tag_mc_firstmarks_seq \l__tag_mc_botmarks_seq
524     { \prg_return_false: }
525     { \prg_return_true: }
526 }

```

(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
527 \prg_new_conditional:Npnn \__tag_check_if_mc_tmb_missing: { T,F,TF }
528 {
529     \bool_if:nTF
530     {
531         \str_if_eq_p:ee {\seq_item:Nn \l__tag_mc_firstmarks_seq {1}}{e-}
532         ||
533         \str_if_eq_p:ee {\seq_item:Nn \l__tag_mc_firstmarks_seq {1}}{b+}
534     }
535     { \prg_return_true: }
536     { \prg_return_false: }
537 }

```

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

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

```

538 \prg_new_conditional:Npnn \__tag_check_if_mc_tme_missing: { T,F,TF }
539 {
540   \str_if_eq:eeTF {\seq_item:Nn \l__tag_mc_botmarks_seq {1}}{b+}
541   { \prg_return_true: }
542   { \prg_return_false: }
543 }

```

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

```

544 </package>

```

```

545 <*debug>

```

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

```

546 \msg_new:nnn { tag / debug } {mc-begin} { MC~begin~#1~with~options:~\tl_to_str:n{#2}~[\msg_line_no:n{#3}] }
547 \msg_new:nnn { tag / debug } {mc-end} { MC~end~#1~[\msg_line_context:] }
548
549 \cs_new_protected:Npn \__tag_debug_mc_begin_insert:n #1
550 {
551   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
552   {
553     \msg_note:nnnn { tag / debug } {mc-begin} {inserted} { #1 }
554   }
555 }
556 \cs_new_protected:Npn \__tag_debug_mc_begin_ignore:n #1
557 {
558   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
559   {
560     \msg_note:nnnn { tag / debug } {mc-begin } {ignored} { #1 }
561   }
562 }
563 \cs_new_protected:Npn \__tag_debug_mc_end_insert:
564 {
565   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
566   {
567     \msg_note:nnn { tag / debug } {mc-end} {inserted}
568   }
569 }
570 \cs_new_protected:Npn \__tag_debug_mc_end_ignore:
571 {
572   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
573   {
574     \msg_note:nnn { tag / debug } {mc-end } {ignored}
575   }
576 }

```

And now something for the structures

```

577 \msg_new:nnn { tag / debug } {struct-begin}
578 {

```

```

579     Struct~\tag_get:n{struct_num}~begin~#1~with~options::~~\tl_to_str:n{#2}~\\[\msg_line_context:
580   }
581 \msg_new:nnn { tag / debug } {struct-end}
582 {
583   Struct-end~#1~[\msg_line_context:]
584 }
585 \msg_new:nnn { tag / debug } {struct-end-wrong}
586 {
587   Struct-end~'#1'~doesn't~fit~start~'#2'~[\msg_line_context:]
588 }
589
590 \cs_new_protected:Npn \__tag_debug_struct_begin_insert:n #1
591 {
592   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
593   {
594     \msg_note:nnnn { tag / debug } {struct-begin} {inserted} { #1 }
595     \seq_log:N \g__tag_struct_tag_stack_seq
596   }
597 }
598 \cs_new_protected:Npn \__tag_debug_struct_begin_ignore:n #1
599 {
600   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
601   {
602     \msg_note:nnnn { tag / debug } {struct-begin} {ignored} { #1 }
603   }
604 }
605 \cs_new_protected:Npn \__tag_debug_struct_end_insert:
606 {
607   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
608   {
609     \msg_note:nnn { tag / debug } {struct-end} {inserted}
610     \seq_log:N \g__tag_struct_tag_stack_seq
611   }
612 }
613 \cs_new_protected:Npn \__tag_debug_struct_end_ignore:
614 {
615   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
616   {
617     \msg_note:nnn { tag / debug } {struct-end} {ignored}
618   }
619 }
620 \cs_new_protected:Npn \__tag_debug_struct_end_check:n #1
621 {
622   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
623   {
624     \seq_get:NNT \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
625     {
626       \str_if_eq:eeF
627       {#1}
628       {\exp_last_unbraced:No \use_i:nn { \l__tag_tmpa_tl }}
629       {
630         \msg_warning:nnee { tag/debug } { struct-end-wrong }
631         {#1}
632         {\exp_last_unbraced:No \use_i:nn { \l__tag_tmpa_tl }}

```

```

633     }
634   }
635 }
636 }

```

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.

```

637 \msg_new:nnn { tag / debug } {tag-suspend}
638 {
639   \int_if_zero:nTF
640     {#1}
641     {Tagging~suspended}
642     {Tagging~(not)~suspended~(already~inactive)}\
643     level:~#1~==>~\int_eval:n{#1+1}\tl_if_empty:nF{#2}{,~label:~#2}~[\msg_line_context:]
644 }
645 \msg_new:nnn { tag / debug } {tag-resume}
646 {
647   \int_if_zero:nTF
648     {#1}
649     {Tagging~resumed}
650     {Tagging~(not)~resumed}\
651     level:~\int_eval:n{#1+1}~==>~#1\tl_if_empty:nF{#2}{,~label:~#2}~[\msg_line_context:]
652 }
653 </debug>

```

7.6 Benchmarks

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

```

654 <*package>
655 \cs_new_protected:Npn \__tag_check_benchmark_tic:{}
656 \cs_new_protected:Npn \__tag_check_benchmark_toc:{}
657 \cs_new_protected:Npn \tag_check_benchmark_on:
658 {
659   \cs_if_exist:NT \benchmark_tic:
660   {
661     \cs_set_eq:NN \__tag_check_benchmark_tic: \benchmark_tic:
662     \cs_set_eq:NN \__tag_check_benchmark_toc: \benchmark_toc:
663   }
664 }
665 </package>

```

Ulrike Fischer
Version 1.0e, released 2026-08-21

Part III

The **tagpdf-user** module

Code related to L^AT_EX 2_ε user commands and document commands

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.
-----------------------------------	---

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_latex as lua_latex doesn't mind much if a mc tag is not correctly closed. Unlike the expl3 command it is not expandable.

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

3 Commands related to structures

<code>\tagstructbegin</code>	<code>\tagstructbegin{⟨key-val⟩}</code>
<code>\tagstructend</code>	<code>\tagstructend</code>
<code>\tagstructuse</code>	<code>\tagstructuse{⟨label⟩}</code>

These are direct wrappers around `\tag_struct_begin:n`, `\tag_struct_end:` and `\tag_struct_use:n`. The commands and their argument are documented in the tagpdf-struct module.

4 Debugging

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

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

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

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

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

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

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

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

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

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

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

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

5 Extension commands

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

The commands and keys should be viewed as experimental!

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

5.1 Fake space

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

5.2 Tagging of paragraphs

This makes use of the paragraph hooks in LaTeX to automate the tagging of paragraph. It requires sane paragraph nesting, faulty code, e.g. a missing `\par` at the end of a low-level vbox can highly confuse the tagging. The tags should be carefully checked if this is used.

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

The `para/tagging` key can be used in `\tagpdfsetup` and enable/disable tagging of paragraphs. `debug/show=para` puts small colored numbers at the begin and end of a paragraph. This is meant as a debugging help. The number are boxes and have a (tiny) height, so they can affect typesetting.

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

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

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

5.3 Header and footer

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

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

5.4 Link tagging

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

Links can have an alternative text in the Contents but this disturbs reading, so it is not done by default. Another text can be added by changing the dictionary value:

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

6 Socket support

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

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

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

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} {2026-08-21} {1.0e}
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 44.)

`\tag_tool:n`

This command is deprecated and will be removed.

`\tagtool` (deprecated)

```

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` (deprecated). These functions are documented on page ??.)

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 44.)

`\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 44.)

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 44.)

11 Tagging Socket support

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

61 <*package>
62 \cs_set_protected:Npn \tag_socket_use:n #1
63 {
64   \bool_if:NT \l__tag_active_socket_bool
65   { \socket_use:n {tagsupport/#1} }
66 }

67 \cs_set_protected:Npn \tag_socket_use:nn #1#2
68 {
69   \bool_if:NT \l__tag_active_socket_bool
70   { \socket_use:nn {tagsupport/#1} {#2} }
71 }

72 \cs_set_protected:Npn \tag_socket_use:nnn #1#2#3
73 {
74   \bool_if:NTF \l__tag_active_socket_bool
75   { \socket_use:nnn {tagsupport/#1} {#2} {#3} }
76   { #3 }
77 }

78 \cs_set:Npn \tag_socket_use_expandable:n #1
79 {
80   \bool_if:NT \l__tag_active_socket_bool
81   { \socket_use_expandable:n {tagsupport/#1} }
82 }

83 \cs_set_protected:Npn \UseTaggingSocket #1
84 {
85   \bool_if:NTF \l__tag_active_socket_bool
86   { \socket_use:nw {tagsupport/#1} }
87   {
88     \int_case:nnF
89     { \int_use:c { c__socket_tagsupport/#1_args_int } }
90     {
91       0 \prg_do_nothing:
92       1 \use_none:n
93       2 \use_ii:nn
94     }
95   }
96 }

97 }

98 \cs_set:Npn \UseExpandableTaggingSocket #1
99 {
100   \bool_if:NTF \l__tag_active_socket_bool
101   { \socket_use_expandable:nw {tagsupport/#1} }
102   {
103     \int_case:nnF
104     { \int_use:c { c__socket_tagsupport/#1_args_int } }
```

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

```

105         {
106             0 \prg_do_nothing:
107             1 \use_none:n
108             2 \use_ii:nn

```

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

```

109         }
110         { \ERRORusetaggingsocket }
111     }
112 }
113 \</package>

```

(End of definition for \tag_socket_use:n and others. These functions are documented on page 47.)

12 Debugging

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

```

114 \<package>
115 \NewDocumentCommand\ShowTagging { m }
116 {
117     \keys_set:nn { __tag / show }{ #1}
118 }
119 }

```

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

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.

```

120 \keys_define:nn { __tag / show }
121 {
122     mc-data .code:n =
123     {
124         \bool_if:NT \g__tag_mode_lua_bool
125         {
126             \lua_now:e{!tx.__tag.trace.show_all_mc_data(#1,\__tag_get_mc_abs_cnt:,0)}
127         }
128     }
129     ,mc-data .default:n = 1
130 }
131

```

(End of definition for mc-data (show-key). This function is documented on page 45.)

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

```

132 \keys_define:nn { __tag / show }
133 { mc-current .code:n =
134     {
135         \bool_if:NTF \g__tag_mode_lua_bool
136         {
137             \int_compare:nNnTF

```

```

138 { -2147483647 }
139 =
140 {
141   \lua_now:e
142   {
143     tex.print
144       (\int_use:N\c_document_cctab,
145        tex.getattribute
146          (luatexbase.attributes.g__tag_mc_cnt_attr))
147   }
148 }
149 {
150   \lua_now:e
151   {
152     ltx.__tag.trace.log
153     (
154       "mc-current:~no~MC~open,~current~absent
155       =\__tag_get_mc_abs_cnt:"
156       ,0
157     )
158     texio.write_nl("")
159   }
160 }
161 {
162   \lua_now:e
163   {
164     ltx.__tag.trace.log
165     (
166       "mc-current:~absent=\__tag_get_mc_abs_cnt:=="
167       ..
168       tex.getattribute(luatexbase.attributes.g__tag_mc_cnt_attr)
169       ..
170       "~=>tag="
171       ..
172       tostring
173         (ltx.__tag.func.get_tag_from
174          (tex.getattribute
175            (luatexbase.attributes.g__tag_mc_type_attr)))
176       ..
177       "=="
178       ..
179       tex.getattribute
180         (luatexbase.attributes.g__tag_mc_type_attr)
181       ,0
182     )
183     texio.write_nl("")
184   }
185 }
186 }
187 {
188   \msg_note:nn{ tag }{ mc-current }
189 }
190 }
191 }

```

(End of definition for `mc-current` (`show-key`). This function is documented on page 45.)

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).

```

192 \keys_define:nn { __tag / show }
193 {
194   mc-marks .choice: ,
195   mc-marks / show .code:n =
196   {
197     \__tag_mc_get_marks:
198     \__tag_check_if_mc_in_galley:TF
199     {
200       \iow_term:n {Marks~from~this~page:~}
201     }
202     {
203       \iow_term:n {Marks~from~a~previous~page:~}
204     }
205     \seq_show:N \l__tag_mc_firstmarks_seq
206     \seq_show:N \l__tag_mc_botmarks_seq
207     \__tag_check_if_mc_tmb_missing:T
208     {
209       \iow_term:n {BDC~missing~on~this~page!}
210     }
211     \__tag_check_if_mc_tme_missing:T
212     {
213       \iow_term:n {EMC~missing~on~this~page!}
214     }
215   },
216   mc-marks / use .code:n =
217   {
218     \__tag_mc_get_marks:
219     \__tag_check_if_mc_in_galley:TF
220     { Marks~from~this~page:~}
221     { Marks~from~a~previous~page:~}
222     \seq_use:Nn \l__tag_mc_firstmarks_seq {,~}\quad
223     \seq_use:Nn \l__tag_mc_botmarks_seq {,~}\quad
224     \__tag_check_if_mc_tmb_missing:T
225     {
226       BDC~missing~
227     }
228     \__tag_check_if_mc_tme_missing:T
229     {
230       EMC~missing
231     }
232   },
233   mc-marks .default:n = show
234 }

```

(End of definition for `mc-marks` (`show-key`). This function is documented on page 45.)

struct-stack (`show-key`)

```

235 \keys_define:nn { __tag / show }
236 {
237   struct-stack .choice:

```

```

238 ,struct-stack / log .code:n = \seq_log:N \g__tag_struct_tag_stack_seq
239 ,struct-stack / show .code:n = \seq_show:N \g__tag_struct_tag_stack_seq
240 ,struct-stack .default:n = show
241 }
242 \</package>

```

(End of definition for `struct-stack (show-key)`. This function is documented on page 45.)

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.

```

243 <*debug>
244 \keys_define:nn { __tag / show }
245 {
246 ,debug/structures .code:n =
247 {
248   \int_step_inline:nnn{#1}{\c@g__tag_struct_abs_int}
249   {
250     \msg_term:nneeee
251     { tag/debug } { show-struct }
252     { ##1 }
253     {
254       \prop_map_function:cN
255       {g__tag_struct_debug_##1_prop}
256       \msg_show_item_unbraced:nn
257     }
258     { } { }
259     \msg_term:nneeee
260     { tag/debug } { show-kids }
261     { ##1 }
262     {
263       \seq_map_function:cN
264       {g__tag_struct_debug_kids_##1_seq}
265       \msg_show_item_unbraced:n
266     }
267     { } { }
268   }
269 }
270 ,debug/structures .default:n = 1
271 }
272 \</debug>

```

(End of definition for `debug/structures (show-key)`. This function is documented on page 45.)

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.

```

273 <*package>

```

13.1 Document structure

```

\g__tag_root_default_tl
  activate (setup-key)
activate/socket (setup-key)
274 \tl_new:N\g__tag_root_default_tl
275 \tl_gset:Nn\g__tag_root_default_tl {Document}
276
277 \hook_gput_code:nnn{begindocument}{tagpdf}{\tagstructbegin{tag=\g__tag_root_default_tl}}
278 \hook_gput_code:nnn{tagpdf/finish/before}{tagpdf}{\tagstructend}
279
280 \keys_define:nn { __tag / setup}
281 {
282   activate/socket .bool_set:N = \l__tag_active_socket_bool,
283   activate .code:n =
284   {
285     \keys_set:nn { __tag / setup }
286     { activate/mc,activate/tree,activate/struct,activate/socket }
287     \tl_gset:Nn\g__tag_root_default_tl {#1}
288   },
289   activate .default:n = Document
290 }
291

```

(End of definition for `\g__tag_root_default_tl`, `activate (setup-key)`, and `activate/socket (setup-key)`). These functions are documented on page 44.)

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.

```

292 \AddToHook{begindocument/before}
293 {
294   \bool_lazy_and:nnT
295   { \g__tag_active_struct_dest_bool }
296   { \g__tag_active_struct_bool }
297   {
298     \tl_set:Nn \l_pdf_current_structure_destination_tl
299     { {__tag/struct}{\g__tag_struct_stack_current_tl } }
300     \pdf_activate_indexed_structure_destination:
301   }
302 }

```

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

```

303 \bool_if:NT \g__tag_mode_lua_bool
304 {
305   \NewDocumentCommand\pdffakespace { }
306   {
307     \__tag_fakespace:
308   }

```

```

309   }
310   \providecommand\pdfspacespace{}

```

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

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
\c_tag_para_textblock_tl
\c_tag_para_semantic_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.

```

311 </package>
312 <base>\bool_new:N \l__tag_para_flattened_bool
313 <base>\bool_new:N \l__tag_para_bool
314 <*package>
315 \int_new:N \g__tag_para_begin_int
316 \int_new:N \g__tag_para_end_int
317 \int_new:N \g__tag_para_main_begin_int
318 \int_new:N \g__tag_para_main_end_int

```

this could hold the structure number of the current para/semantic (currently unused)

```

319 \tl_new:N \g__tag_para_main_struct_tl
320 \tl_gset:Nn \g__tag_para_main_struct_tl {1}

```

TODO: check the use in ltagging in para/restore

```

321 \tl_new:N \l__tag_para_tag_default_tl
322 \tl_set:Nn \l__tag_para_tag_default_tl { \UseStructureName {para/textblock} }

```

The para-tag names are changed in some structures, so we store the default values for messages, resets and places where we want to force the default names.

```

323 \AddToHookNext{class/before}
324 {%
325   \tl_const:Ne \c_tag_para_textblock_tl { \UseStructureName {para/textblock} }
326   \tl_const:Ne \c_tag_para_semantic_tl { \UseStructureName {para/semantic} }
327 }
328 \tl_new:N \l__tag_para_tag_tl
329 \tl_set:Nn \l__tag_para_tag_tl { \UseStructureName {para/textblock} }
330 \tl_new:N \l__tag_para_main_tag_tl
331 \tl_set:Nn \l__tag_para_main_tag_tl { \UseStructureName {para/semantic} }
332 %\tl_new:N \l__tag_para_attr_class_tl % defined in ltagging
333 \tl_new:N \l__tag_para_main_attr_class_tl

```

(End of definition for \l__tag_para_bool and others. These functions are documented on page ??.)

```

\__tag_gincr_para_main_begin_int:
\__tag_gincr_para_main_end_int:
\__tag_gincr_para_begin_int:
\__tag_gincr_para_end_int:

```

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

```

334 \cs_new_protected:Npn \__tag_gincr_para_main_begin_int:
335 {
336   \int_gincr:N \g__tag_para_main_begin_int
337 }
338 \cs_new_protected:Npn \__tag_gincr_para_begin_int:
339 {
340   \int_gincr:N \g__tag_para_begin_int
341 }

```

```

342 \cs_new_protected:Npn \__tag_gincr_para_main_end_int:
343 {
344   \int_gincr:N \g__tag_para_main_end_int
345 }
346 \cs_new_protected:Npn \__tag_gincr_para_end_int:
347 {
348   \int_gincr:N \g__tag_para_end_int
349 }

```

(End of definition for __tag_gincr_para_main_begin_int: and others.)

```

\__tag_start_para_ints:
\__tag_stop_para_ints:
350 \cs_new_protected:Npn \__tag_start_para_ints:
351 {
352   \cs_set_protected:Npn \__tag_gincr_para_main_begin_int:
353   {
354     \int_gincr:N \g__tag_para_main_begin_int
355   }
356   \cs_set_protected:Npn \__tag_gincr_para_begin_int:
357   {
358     \int_gincr:N \g__tag_para_begin_int
359   }
360   \cs_set_protected:Npn \__tag_gincr_para_main_end_int:
361   {
362     \int_gincr:N \g__tag_para_main_end_int
363   }
364   \cs_set_protected:Npn \__tag_gincr_para_end_int:
365   {
366     \int_gincr:N \g__tag_para_end_int
367   }
368 }
369 \cs_new_protected:Npn \__tag_stop_para_ints:
370 {
371   \cs_set_eq:NN \__tag_gincr_para_main_begin_int:\prg_do_nothing:
372   \cs_set_eq:NN \__tag_gincr_para_begin_int:\prg_do_nothing:
373   \cs_set_eq:NN \__tag_gincr_para_main_end_int:\prg_do_nothing:
374   \cs_set_eq:NN \__tag_gincr_para_end_int:\prg_do_nothing:
375 }

```

(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 (TODO: remove 2026-11-01)

```

\__tag_para_main_store_struct:
376 \cs_new_protected:Npn \__tag_para_main_store_struct:
377 {
378   \tl_gset:Ne \g__tag_para_main_struct_tl {\int_use:N \c@g__tag_struct_abs_int }
379 }

```

(End of definition for __tag_para_main_store_struct:.)

para/tagging (setup-key)
para/tag (setup-key)
para/maintag (setup-key)
 para/tagging (deprecated tool-key)
 para/tag (deprecated tool-key)
 para/maintag (deprecated tool-key)
 para/flattened (deprecated 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`

```

380 \keys_define:nn { __tag / setup }
381 {
382   para/tagging      .bool_set:N = \l__tag_para_bool,
383   debug/show/para   .code:n      = {\bool_set_true:N \l__tag_para_show_bool},
384   debug/show/paraOff.code:n      = {\bool_set_false:N \l__tag_para_show_bool},
385   para/tag          .code:n      =
386   {
387     \tl_set:Nn \l__tag_para_tag_tl {#1}% needed for compatibility with 2026-
06-01
388     \AssignStructureRole{para/textblock}{#1}
389   },
390   para/maintag      .code:n      =
391   {
392     \tl_set:Nn \l__tag_para_main_tag_tl{#1}% needed for compatibility with 2026-
06-01
393     \AssignStructureRole{para/semantic}{#1}
394   },
395   para/flattened     .bool_set:N = \l__tag_para_flattened_bool
396 }

```

deprecated key definitions:

```

397 \keys_define:nn { tag / tool}
398 {
399   para/tagging      .bool_set:N = \l__tag_para_bool,
400   para/tag          .code:n      =
401   {
402     \tl_set:Nn \l__tag_para_tag_tl {#1}% needed for compatibility with 2026-
06-01
403     \AssignStructureRole{para/textblock}{#1}
404   },
405   para/maintag      .code:n      =
406   {
407     \tl_set:Nn \l__tag_para_main_tag_tl{#1}% needed for compatibility with 2026-
06-01
408     \AssignStructureRole{para/semantic}{#1}
409   },
410   para/flattened     .bool_set:N = \l__tag_para_flattened_bool
411 }

```

the deprecated names

```

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

```

(End of definition for `para/tagging` (setup-key) and others. These functions are documented on page 46.)

Helper command for debugging:

```

418 \cs_new_protected:Npn \__tag_check_para_begin_show:nn #1 #2
419   %#1 color, #2 prefix
420   {
421     \bool_if:NT \l__tag_para_show_bool
422     {
423       \tag_mc_begin:n{artifact}
424       \llap{\color_select:n{#1}\tiny#2\int_use:N\g__tag_para_begin_int\ }
425       \tag_mc_end:
426     }
427   }
428
429 \cs_new_protected:Npn \__tag_check_para_end_show:nn #1 #2
430   %#1 color, #2 prefix
431   {
432     \bool_if:NT \l__tag_para_show_bool
433     {
434       \tag_mc_begin:n{artifact}
435       \rlap{\color_select:n{#1}\tiny\ #2\int_use:N\g__tag_para_end_int}
436       \tag_mc_end:
437     }
438   }
439
440 \AddToHook{begindocument/before}[tagpdf/para]
441 {
442   \AddToHook{para/begin}[tagpdf]{ \tag_socket_use:n { para/begin } }
443   \AddToHook{para/end} [tagpdf] { \tag_socket_use:n { para/end } }
444 }

```

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 code didn't guard everything correctly with tagging sockets.

```

444 \AddToHook{enddocument/info}
445 {
446   \tag_if_active:F
447   {
448     \msg_redirect_name:nnn { tag } { para-hook-count-wrong } { warning }
449   }
450   \int_compare:nNnF {\g__tag_para_main_begin_int}={\g__tag_para_main_end_int}
451   {
452     \msg_error:nneee
453     {tag}
454     {para-hook-count-wrong}
455     {\int_use:N\g__tag_para_main_begin_int}
456     {\int_use:N\g__tag_para_main_end_int}
457     {\c_tag_para_textblock_tl}
458   }
459   \int_compare:nNnF {\g__tag_para_begin_int}={\g__tag_para_end_int}
460   {
461     \msg_error:nneee
462     {tag}
463     {para-hook-count-wrong}
464     {\int_use:N\g__tag_para_begin_int}
465     {\int_use:N\g__tag_para_end_int}

```

```

466         {\c_tag_para_semantic_tl}
467     }
468 }

```

</package>

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

\tagpdfparaOff

```

469 \base\newcommand\tagpdfparaOn {}
470 \base\newcommand\tagpdfparaOff{}
471 \*package)
472 \renewcommand\tagpdfparaOn {\bool_set_true:N \l__tag_para_bool}
473 \renewcommand\tagpdfparaOff{\bool_set_false:N \l__tag_para_bool}

```

(End of definition for **\tagpdfparaOn** and **\tagpdfparaOff**. These functions are documented on page 46.)

\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}%

```

```

474 \NewDocumentCommand\tagpdfsuppressmarks{m}
475   {\use:c{\__tag_mc_disable_marks:} #1}}

```

(End of definition for **\tagpdfsuppressmarks**. This function is documented on page 46.)

13.5 Language support

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

test/lang (setup-key)

```

476 \keys_define:nn { __tag / setup }
477   {
478     text / lang .tl_set:N = \l__tag_struct_lang_tl
479   }

```

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

13.6 Header and footer

Header and footer should normally be tagged as artifacts. The following code requires the output tagging sockets. 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.

```
480 \cs_new_protected:Npn \__tag_hook_kernel_before_head:{}
481 \cs_new_protected:Npn \__tag_hook_kernel_after_head:{}
482 \cs_new_protected:Npn \__tag_hook_kernel_before_foot:{}
483 \cs_new_protected:Npn \__tag_hook_kernel_after_foot:{}

```

We use the page sockets.

```
484 \NewTaggingSocketPlug{build/page/header}{tagpdf}
485 {
486   \__tag_hook_kernel_before_head:
487   #2
488   \__tag_hook_kernel_after_head:
489 }
490
491 \AssignTaggingSocketPlug{build/page/header}{tagpdf}
492 \NewTaggingSocketPlug{build/page/footer}{tagpdf}
493 {
494   \__tag_hook_kernel_before_foot:
495   #2
496   \__tag_hook_kernel_after_foot:
497 }
498 \AssignTaggingSocketPlug{build/page/footer}{tagpdf}
499
500 \bool_new:N \g__tag_saved_in_mc_bool
501 \cs_new_protected:Npn \__tag_exclude_headfoot_begin:
502 {
503   \bool_set_false:N \l__tag_para_bool
504   \bool_if:NTF \g__tag_mode_lua_bool
505   {
506     \tag_mc_end_push:
507   }
508   {
509     \bool_gset_eq:NN \g__tag_saved_in_mc_bool \g__tag_in_mc_bool
510     \bool_gset_false:N \g__tag_in_mc_bool
511   }
512   \tag_mc_begin:n {artifact}
513   \tag_suspend:n{headfoot}
514 }
515 \cs_new_protected:Npn \__tag_exclude_headfoot_end:
516 {
517   \tag_resume:n{headfoot}
518   \tag_mc_end:
519   \bool_if:NTF \g__tag_mode_lua_bool
520   {
521     \tag_mc_begin_pop:n{ }
522   }
523   {
524     \bool_gset_eq:NN \g__tag_in_mc_bool \g__tag_saved_in_mc_bool
525   }
526 }

```

This version allows to use an Artifact structure

```

526 \__tag_attr_new_entry:nn {__tag/attr/pagination}{/0/Artifact/Type/Pagination}
527 \cs_new_protected:Npn \__tag_exclude_struct_headfoot_begin:n #1
528 {
529     \bool_set_false:N \l__tag_para_bool
530     \bool_if:NTF \g__tag_mode_lua_bool
531     {
532         \tag_mc_end_push:
533     }
534     {
535         \bool_gset_eq:NN \g__tag_saved_in_mc_bool \g__tag_in_mc_bool
536         \bool_gset_false:N \g__tag_in_mc_bool
537     }
538     \tag_struct_begin:n{tag=Artifact,attribute-class=__tag/attr/#1}
539     \tag_mc_begin:n {artifact=#1}
540     \tag_suspend:n{headfoot}
541 }
542
543 \cs_new_protected:Npn \__tag_exclude_struct_headfoot_end:
544 {
545     \tag_resume:n{headfoot}
546     \tag_mc_end:
547     \tag_struct_end:
548     \bool_if:NTF \g__tag_mode_lua_bool
549     {
550         \tag_mc_begin_pop:n{ }
551     }
552     {
553         \bool_gset_eq:NN \g__tag_in_mc_bool \g__tag_saved_in_mc_bool
554     }
555 }

```

And now the keys

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

```

556 \keys_define:nn { __tag / setup }
557 {
558     page/exclude-header-footer .choice:,
559     page/exclude-header-footer / true .code:n =
560     {
561         \cs_set_eq:NN \__tag_hook_kernel_before_head: \__tag_exclude_headfoot_begin:
562         \cs_set_eq:NN \__tag_hook_kernel_before_foot: \__tag_exclude_headfoot_begin:
563         \cs_set_eq:NN \__tag_hook_kernel_after_head: \__tag_exclude_headfoot_end:
564         \cs_set_eq:NN \__tag_hook_kernel_after_foot: \__tag_exclude_headfoot_end:
565     },
566     page/exclude-header-footer / pagination .code:n =
567     {
568         \cs_set:Nn \__tag_hook_kernel_before_head: { \__tag_exclude_struct_headfoot_begin:n {p
569         \cs_set:Nn \__tag_hook_kernel_before_foot: { \__tag_exclude_struct_headfoot_begin:n {p
570         \cs_set_eq:NN \__tag_hook_kernel_after_head: \__tag_exclude_struct_headfoot_end:
571         \cs_set_eq:NN \__tag_hook_kernel_after_foot: \__tag_exclude_struct_headfoot_end:
572     },
573     page/exclude-header-footer / false .code:n =
574     {

```

```

575     \cs_set_eq:NN \__tag_hook_kernel_before_head: \prg_do_nothing:
576     \cs_set_eq:NN \__tag_hook_kernel_before_foot: \prg_do_nothing:
577     \cs_set_eq:NN \__tag_hook_kernel_after_head: \prg_do_nothing:
578     \cs_set_eq:NN \__tag_hook_kernel_after_foot: \prg_do_nothing:
579   },
580   page/exclude-header-footer .default:n = true,
581   page/exclude-header-footer .initial:n = true,

```

deprecated name

```

582     exclude-header-footer .meta:n = { page/exclude-header-footer = {#1} }
583   }

```

(End of definition for page/exclude-header-footer (setup-key) and exclude-header-footer (deprecated).
These functions are documented on page 46.)

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

```

584 \AtBeginDocument
585 {
586   \NewTaggingSocketPlug{parbox/before}{tag/footer}
587   {
588     \tag_struct_begin:n{tag=Span}
589     \tag_mc_begin:n{ }
590   }
591
592   \NewTaggingSocketPlug{parbox/after}{tag/footer}
593   {
594     \tag_mc_end:
595     \tag_struct_end:
596   }
597 }
598 \cs_new_protected:Npn \__tag_headfoot_tagged_begin:n #1
599 {
600   \AssignTaggingSocketPlug{parbox/before}{tag/footer}
601   \AssignTaggingSocketPlug{parbox/after}{tag/footer}
602   \bool_set_false:N \l__tag_para_bool
603   \bool_if:NTF \g__tag_mode_lua_bool
604   {
605     \tag_mc_end_push:
606   }
607   {
608     \bool_gset_eq:NN \g__tag_saved_in_mc_bool \g__tag_in_mc_bool
609     \bool_gset_false:N \g__tag_in_mc_bool
610   }
611   \tag_struct_begin:n{tag=Artifact,attribute-class=__tag/attr/#1,parent=\tag_get:n{current_
612 }
613
614 \cs_new_protected:Npn \__tag_headfoot_tagged_end:
615 {
616   \tag_struct_end:
617   \bool_if:NTF \g__tag_mode_lua_bool
618   {
619     \tag_mc_begin_pop:n{ }
620   }

```

```

621     {
622       \bool_gset_eq:NN \g__tag_in_mc_bool\g__tag_saved_in_mc_bool
623     }
624   }
625 \keys_define:nn { __tag / setup }
626 {
627   page/tag-header-footer .choice:,
628   page/tag-header-footer/artifact .code:n =
629   {
630     \cs_set:Nn \__tag_hook_kernel_before_head: { \__tag_headfoot_tagged_begin:n {pagination
631     \cs_set:Nn \__tag_hook_kernel_before_foot: { \__tag_headfoot_tagged_begin:n {pagination
632     \cs_set_eq:NN \__tag_hook_kernel_after_head: \__tag_headfoot_tagged_end:
633     \cs_set_eq:NN \__tag_hook_kernel_after_foot: \__tag_headfoot_tagged_end:
634   },
635   page/tag-header-footer .default:n = artifact,
636 }

```

13.7 Links

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

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

```

637 \AddToHookNext{class/before}
638 {
639   \cs_if_exist:NF \l__tag_name_link_tl
640   {
641     \tl_new:N \l__tag_name_link_tl
642     \tl_set:Nn \l__tag_name_link_tl{Link}
643   }
644 }

```

Tagging sockets for links

```

645 \socket_if_exist:nF {tagsupport/link/before}
646 {
647   \NewTaggingSocket{link/before}{1}
648   \NewTaggingSocket{link/after}{1}
649 }
650 \NewTaggingSocketPlug{link/before}{kernel}
651 {
652   \mode_leave_vertical:
653   \tag_mc_end_push:
654   \tag_struct_begin:n { tag=\UseStructureName{link} }
655   \tag_mc_begin:n {}
656   #1
657 }
658 \AssignTaggingSocketPlug{link/before}{kernel}
659
660 \NewTaggingSocketPlug{link/after}{kernel}
661 {

```

```

662 #1
663 \tag_mc_end:
664 \tag_struct_end:
665 \tag_mc_begin_pop:n{}
666 }
667 \AssignTaggingSocketPlug{link/after}{kernel}
668
669
670 \bool_lazy_and:nnTF
671 { \sys_if_engine luatex_p: }
672 {
673   \tl_if_empty_p:e
674   {
675     \lua_now:e
676     { if~ luatexbase.in_callback('pre_shipout_filter','linksplit')~
677       then~else~tex.print('1')~end
678     }
679   }
680 }
681 {
682   \hook_gput_code:nnn
683   {pdfannot/link/URI/before}
684   {tagpdf}
685   {
686     \UseTaggingSocket{link/before}{}
687   }
688
689   \hook_gput_code:nnn
690   {pdfannot/link/URI/after}
691   {tagpdf}
692   {
693     \UseTaggingSocket{link/after}{}
694   }
695
696   \hook_gput_code:nnn
697   {pdfannot/link/GoTo/before}
698   {tagpdf}
699   {
700     \UseTaggingSocket{link/before}{}
701   }
702
703   \hook_gput_code:nnn
704   {pdfannot/link/GoTo/after}
705   {tagpdf}
706   {
707     \UseTaggingSocket{link/after}{}
708   }
709
710   \hook_gput_code:nnn
711   {pdfannot/link/GoToR/before}
712   {tagpdf}
713   {
714     \UseTaggingSocket{link/before}
715   }

```

```

716
717 \hook_gput_code:nnn
718   {pdfannot/link/GoToR/after}
719   {tagpdf}
720   {
721     \UseTaggingSocket{link/after}{}
722   }
723 \hook_gput_code:nnn
724   {pdfannot/link/Launch/before}
725   {tagpdf}
726   {
727     \UseTaggingSocket{link/before}{}
728   }
729
730 \hook_gput_code:nnn
731   {pdfannot/link/Launch/after}
732   {tagpdf}
733   {
734     \UseTaggingSocket{link/after}{}
735   }
736 \hook_gput_code:nnn
737   {pdfannot/link/Named/before}
738   {tagpdf}
739   {
740     \UseTaggingSocket{link/before}{}
741   }
742
743 \hook_gput_code:nnn
744   {pdfannot/link/Named/after}
745   {tagpdf}
746   {
747     \UseTaggingSocket{link/after}{}
748   }
749 }
750 {
751   \hook_gput_code:nnn
752     {pdfannot/link/URI/before}
753     {tagpdf}
754     {
755       \UseTaggingSocket{link/before}
756       {
757         \pdfannot_dict_put:nne
758           { link/URI }
759           { StructParent }
760           { \tag_struct_parent_int: }
761       }
762     }
763
764   \hook_gput_code:nnn
765     {pdfannot/link/URI/after}
766     {tagpdf}
767     {
768       \UseTaggingSocket{link/after}
769       {

```

```

770         \tag_struct_insert_annot:ee
771         {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
772     }
773 }
774
775 \hook_gput_code:nnn
776 {pdfannot/link/GoTo/before}
777 {tagpdf}
778 {
779     \UseTaggingSocket{link/before}
780     {
781         \pdfannot_dict_put:nne
782         { link/GoTo }
783         { StructParent }
784         { \tag_struct_parent_int: }
785     }
786 }
787
788 \hook_gput_code:nnn
789 {pdfannot/link/GoTo/after}
790 {tagpdf}
791 {
792     \UseTaggingSocket{link/after}
793     {
794         \tag_struct_insert_annot:ee
795         {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
796     }
797 }
798
799 \hook_gput_code:nnn
800 {pdfannot/link/GoToR/before}
801 {tagpdf}
802 {
803     \UseTaggingSocket{link/before}
804     {
805         \pdfannot_dict_put:nne
806         { link/GoToR }
807         { StructParent }
808         { \tag_struct_parent_int: }
809     }
810 }
811
812 \hook_gput_code:nnn
813 {pdfannot/link/GoToR/after}
814 {tagpdf}
815 {
816     \UseTaggingSocket{link/after}
817     {
818         \tag_struct_insert_annot:ee
819         {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
820     }
821 }
822
823 \hook_gput_code:nnn

```

```

824     {pdfannot/link/Named/before}
825     {tagpdf}
826     {
827         \UseTaggingSocket{link/before}
828         {
829             \pdfannot_dict_put:nne
830             { link/Named }
831             { StructParent }
832             { \tag_struct_parent_int: }
833         }
834     }
835
836     \hook_gput_code:nnn
837     {pdfannot/link/Named/after}
838     {tagpdf}
839     {
840         \UseTaggingSocket{link/after}
841         {
842             \tag_struct_insert_annot:ee
843             {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
844         }
845     }
846     \hook_gput_code:nnn
847     {pdfannot/link/Launch/before}
848     {tagpdf}
849     {
850         \UseTaggingSocket{link/before}
851         {
852             \pdfannot_dict_put:nne
853             { link/Launch }
854             { StructParent }
855             { \tag_struct_parent_int: }
856         }
857     }
858
859     \hook_gput_code:nnn
860     {pdfannot/link/Launch/after}
861     {tagpdf}
862     {
863         \UseTaggingSocket{link/after}
864         {
865             \tag_struct_insert_annot:ee
866             {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
867         }
868     }
869 }

```

13.8 Attaching css-files for derivation

Derivation to html (https://pdfa.org/wp-content/uploads/2019/06/Deriving_HTML_from_PDF.pdf, implemented by, e.g., ngpdf) can be improved by attaching CSS style definitions in associated files with relationship supplement to the Catalog¹.

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

Such CSS style definitions can be given in two ways:

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

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

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

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

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

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

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

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

```

870 \prop_new:N \g__tag_css_prop
871 \prop_gset_from_keyval:Nn \g__tag_css_prop
872 {
873   latex-list-css.html=true,
874   latex-align-css.html=true
875 }
876
877
878 \bool_new:N \g__tag_css_bool
879 \bool_gset_true:N \g__tag_css_bool

```

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

```

880 \tl_gput_left:Nn \g__kernel_pdfmanagement_end_run_code_tl
881 {
882   \bool_lazy_and:nnT { \g__tag_css_bool }{ \tag_if_active_p: }
883   {
884     \prop_map_inline:Nn \g__tag_css_prop
885     {
886       \keys_set:nn { __tag / setup }{ catalog-supplemental-file= {#1} }
887     }
888   }
889 }

```

```

890
891 \keys_define:nn { __tag / setup }
892 {
893   attach-css .bool_gset:N = \g__tag_css_bool,
894   css-list .code:n =
895   {
896     \tl_if_empty:nTF{#1}
897     { \prop_gclear:N \g__tag_css_prop }
898     { \prop_gput:Nnn \g__tag_css_prop { #1 }{true}}
899   },
900   css-list-add .code:n = { \prop_gput:Nnn \g__tag_css_prop { #1 }{true} },
901   css-list-remove .code:n = { \prop_gremove:Nn \g__tag_css_prop { #1 } },
902 }
</package>
Ulrike Fischer
Version 1.0e, released 2026-08-21

```

Part IV

The tagpdf-tree module

Commands trees and main dictionaries

```
1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-tree-code} {2026-08-21} {1.0e}
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:nNf {\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{begin:document}{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:nNf {\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 be done if values have been used.

```
\__tag_tree_write_classmap:
```

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

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

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

(End of definition for __tag_tree_write_classmap:.)

1.8 Namespaces

Namespaces are handled 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}

```

Ulrike Fischer
Version 1.0e, released 2026-08-21

Part V

The **tagpdf-mc-shared** module

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

1 Public Commands

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

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

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

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

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

New: 2019-11-20

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

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

New: 2021-04-22

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

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

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

New: 2023-06-11

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

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

New: 2024-11-18

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

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

New: 2024-11-18

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

2 Public keys

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

<code>tag (mc-key)</code>	This key is required, unless artifact is used. The value is a tag like <code>P</code> or <code>H1</code> 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 <code>H4</code> is fine).
---------------------------	--

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

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

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

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

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

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

3 Marked content code – shared

```

1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-mc-code-shared} {2026-08-21} {1.0e}
4 {part of tagpdf - code related to marking chunks -
5   code shared by generic and luamode }
6 </header>

```

3.1 Variables and counters

MC chunks must be counted. I use a latex counter for the absolute count, so that it is added to `\cl@@ckpt` and restored e.g. in tabulars and align. `\int_new:N \c@g_@@_MCID_abs_int` and `\tl_put_right:Nn\cl@@ckpt{\@elt{g_@@_MCID_abs_int}}` would work too, but as the name is not `expl3` then too, why bother? The absolute counter can be used to label and to check if the page counter needs a reset.

`g__tag_MCID_abs_int`

```

7 <*base>
8 \newcounter { g__tag_MCID_abs_int }

```

(End of definition for `g__tag_MCID_abs_int`.)

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

```

9 \cs_new:Npn \__tag_get_data_mc_counter:
10 {
11   \int_use:N \c@g__tag_MCID_abs_int
12 }
13 </base>

```

(End of definition for `\__tag_get_data_mc_counter:.`)

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

```

14 <*shared>
15 \cs_new:Npn \__tag_get_mc_abs_cnt: { \int_use:N \c@g__tag_MCID_abs_int }

```

(End of definition for `\__tag_get_mc_abs_cnt:.`)

`\g__tag_in_mc_bool`

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

```

16 \bool_new:N \g__tag_in_mc_bool

```

(End of definition for `\g__tag_in_mc_bool`.)

`\g__tag_mc_parenttree_prop` For every chunk we need to know the structure it is in, to record this in the parent tree. We store this in a property.
key: absolute number of the mc (tagmcabs)
value: the structure number the mc is in

```

17 \__tag_prop_new_linked:N \g__tag_mc_parenttree_prop

```

(End of definition for `\g__tag_mc_parenttree_prop`.)

`\g__tag_mc_parenttree_prop` Some commands (e.g. links) want to close a previous mc and reopen it after they did their work. For this we create a stack:

```

18 \seq_new:N \g__tag_mc_stack_seq

```

(End of definition for `\g__tag_mc_parenttree_prop`.)

`\l__tag_mc_artifact_type_tl` Artifacts can have various types like Pagination or Layout. This stored in this variable.

```

19 \tl_new:N \l__tag_mc_artifact_type_tl

```

(End of definition for `\l__tag_mc_artifact_type_tl`.)

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

```

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

```

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

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

```

22 \tl_new:N \l__tag_mc_lang_tl

```

(End of definition for `\l__tag_mc_lang_tl`.)

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

```

23 \tl_new:N \l__tag_mc_key_tag_tl
24 \tl_new:N \g__tag_mc_key_tag_tl
25 \tl_new:N \l__tag_mc_key_label_tl
26 \tl_new:N \l__tag_mc_key_properties_tl

```

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

3.2 Functions

`\__tag_mc_handle_mc_label:e` The commands labels a mc-chunk. It is used if the user explicitly labels the mc-chunk with the `label` key. The argument is the value provided by the user. It stores the attributes
tagabspage: the absolute page, `\g_shipout_readonly_int`,
tagmcabs: the absolute mc-counter `\c@g_@@_MCID_abs_int`. The reference command is based on `ltproperty`.

```

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

```

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

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

```

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

```

(End of definition for `\__tag_mc_set_label_used:n`.)

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

TODO: is testing for struct the right test?

```

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

```

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

`\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:`

```

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

```

```

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

```

(End of definition for \tag_mc_artifact_group_begin:n and \tag_mc_artifact_group_end:. These functions are documented on page 82.)

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

```

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

```

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

\tag_mc_end_push:
\tag_mc_begin_pop:n

```

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

```

```

115         }
116     }
117     {
118         \__tag_check_mc_pushed_popped:nn {popped}{empty~stack,~nothing}
119     }
120 }
121 }

```

(End of definition for \tag_mc_end_push: and \tag_mc_begin_pop:n. These functions are documented on page 82.)

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

```

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

```

This records if logging is on

```

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

```

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

```

157     \int_compare:nNnT {\l__tag_parent_child_check_tl} = { \c__tag_role_rule_checkparent_tl }
158     {

```

```

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

```

3.3 Keys

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

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

```

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

```

```

202     },
203     __artifact-type / page .code:n =
204     {
205         \tl_set:Nn \l__tag_mc_artifact_type_tl { Page }
206     },
207     __artifact-type / background .code:n =
208     {
209         \tl_set:Nn \l__tag_mc_artifact_type_tl { Background }
210     },
211     __artifact-type / notype .code:n =
212     {
213         \tl_set:Nn \l__tag_mc_artifact_type_tl {}
214     },
215     __artifact-type / .code:n =
216     {
217         \tl_set:Nn \l__tag_mc_artifact_type_tl {}
218     },
219 }

```

(End of definition for `stash (mc-key)`, `__artifact-bool`, and `__artifact-type`. This function is documented on page 84.)

220 $\langle$ /shared $\rangle$

Ulrike Fischer
Version 1.0e, released 2026-08-21

Part VI

The tagpdf-mc-generic module

Code related to Marked Content (mc-chunks), generic mode

1 Marked content code – generic mode

```
1 <@@=tag>
2 <*generic>
3 \ProvidesExplPackage {tagpdf-mc-code-generic} {2026-08-21} {1.0e}
4 {part of tagpdf - code related to marking chunks - generic mode}
5 </generic>
6 <*debug>
7 \ProvidesExplPackage {tagpdf-debug-generic} {2026-08-21} {1.0e}
8 {part of tagpdf - debugging code related to marking chunks - generic mode}
9 </debug>
```

1.1 Variables

```
10 <*generic>
```

`\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.

```
11 \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. TODO: perhaps an interface for more streams will be needed.

```
\g__tag_mc_footnote_marks_seq
\g__tag_mc_multicol_marks_seq
```

```
12 \seq_new:N \g__tag_mc_main_marks_seq
13 \seq_new:N \g__tag_mc_footnote_marks_seq
14 \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
```

```
15 \cs_new_protected:Npn \tag_mc_new_stream:n #1
16 {
17   \seq_new:c { g__tag_mc_multicol_#1_seq }
18 }
```

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

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

```
\l__tag_mc_botmarks_seq
```

```
19 \seq_new:N \l__tag_mc_firstmarks_seq
20 \seq_new:N \l__tag_mc_botmarks_seq
```

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

1.2 Functions

`\__tag_mc_begin_marks:nn` Generic mode need to set marks for the page break and split stream handling. We always 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).

```

21 \cs_new_protected:Npn \__tag_mc_begin_marks:nn #1 #2 %#1 tag, #2 label
22 {
23   \tex_marks:D \g__tag_mc_marks
24   {
25     b-, %first of begin pair
26     \int_use:N\c@g__tag_MCID_abs_int, %mc-num
27     \g__tag_struct_stack_current_tl, %structure num
28     #1, %tag
29     \bool_if:NT \l__tag_mc_key_stash_bool{stash}, % stash info
30     #2, %label
31   }
32   \tex_marks:D \g__tag_mc_marks
33   {
34     b+, % second of begin pair
35     \int_use:N\c@g__tag_MCID_abs_int, %mc-num
36     \g__tag_struct_stack_current_tl, %structure num
37     #1, %tag
38     \bool_if:NT \l__tag_mc_key_stash_bool{stash}, % stash info
39     #2, %label
40   }
41 }
42 \cs_generate_variant:Nn \__tag_mc_begin_marks:nn {oo}
43 \cs_new_protected:Npn \__tag_mc_artifact_begin_marks:n #1 %#1 type
44 {
45   \tex_marks:D \g__tag_mc_marks
46   {
47     b-, %first of begin pair
48     \int_use:N\c@g__tag_MCID_abs_int, %mc-num
49     -1, %structure num
50     #1 %type
51   }
52   \tex_marks:D \g__tag_mc_marks
53   {
54     b+, %first of begin pair
55     \int_use:N\c@g__tag_MCID_abs_int, %mc-num
56     -1, %structure num
57     #1 %Type
58   }
59 }
60
61 \cs_new_protected:Npn \__tag_mc_end_marks:
62 {
63   \tex_marks:D \g__tag_mc_marks
64   {
65     e-, %first of end pair

```

```

66      \int_use:N\c@g__tag_MCID_abs_int, %mc-num
67      \g__tag_struct_stack_current_tl, %structure num
68    }
69    \tex_marks:D \g__tag_mc_marks
70    {
71      e+, %second of end pair
72      \int_use:N\c@g__tag_MCID_abs_int, %mc-num
73      \g__tag_struct_stack_current_tl, %structure num
74    }
75  }

```

(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.

```

76 \cs_new_protected:Npn \__tag_mc_disable_marks:
77 {
78   \cs_set_eq:NN \__tag_mc_begin_marks:nn \use_none:nn
79   \cs_set_eq:NN \__tag_mc_artifact_begin_marks:n \use_none:n
80   \cs_set_eq:NN \__tag_mc_end_marks: \prg_do_nothing:
81 }

```

(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.

```

82 \cs_new_protected:Npn \__tag_mc_get_marks:
83 {
84   \exp_args:NNe
85   \seq_set_from_clist:Nn \l__tag_mc_firstmarks_seq
86   { \tex_firstmarks:D \g__tag_mc_marks }
87   \exp_args:NNe
88   \seq_set_from_clist:Nn \l__tag_mc_botmarks_seq
89   { \tex_botmarks:D \g__tag_mc_marks }
90 }

```

(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.

```

91 \cs_new_protected:Npn \__tag_mc_store:nnn #1 #2 #3 %#1 mc-prev, #2 mc-num #3 structure-
    num
92 {
93   %\prop_show:N \g__tag_struct_cont_mc_prop
94   \prop_get:NnNTF \g__tag_struct_cont_mc_prop {#1} \l__tag_tmpa_tl
95   {
96     \prop_gput:Nne \g__tag_struct_cont_mc_prop {#1}{ \l__tag_tmpa_tl \__tag_struct_mcid_c
97   }
98   {
99     \prop_gput:Nne \g__tag_struct_cont_mc_prop {#1}{ \__tag_struct_mcid_dict:n {#2}}
100  }
101  \prop_gput:Nee \g__tag_mc_parenttree_prop
102  {#2}

```

```

103     {#3}
104   }
105   \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.

```

106 \cs_new_protected:Npn \__tag_mc_insert_extra_tmb:n #1 % #1 stream: e.g. main or footnote
107 {
108   \__tag_check_typeout_v:n {>~ first~ \seq_use:Nn \l__tag_mc_firstmarks_seq {,~}}
109   \__tag_check_typeout_v:n {>~ bot~ \seq_use:Nn \l__tag_mc_botmarks_seq {,~}}
110   \__tag_check_if_mc_tmb_missing:TF
111   {
112     \__tag_check_typeout_v:n {>~ TMB~ ~ missing~ --- inserted}
113     %test if artifact
114     \int_compare:nNnTF { \seq_item:cn { g__tag_mc_#1_marks_seq } {3} } = {-
115       1}
116     {
117       \tl_set:Nx \l__tag_tmpa_tl { \seq_item:cn { g__tag_mc_#1_marks_seq } {4} }
118       \__tag_mc_handle_artifact:N \l__tag_tmpa_tl
119     }
120     {
121       \exp_args:Nx
122       \__tag_mc_bdc_mcid:n
123       {
124         \seq_item:cn { g__tag_mc_#1_marks_seq } {4}
125       }
126       \str_if_eq:eeTF
127       {
128         \seq_item:cn { g__tag_mc_#1_marks_seq } {5}
129       }
130       {
131         %store
132         \__tag_mc_store:eee
133         {
134           \seq_item:cn { g__tag_mc_#1_marks_seq } {2}
135         }
136         { \int_eval:n{\c@g__tag_MCID_abs_int} }
137         {
138           \seq_item:cn { g__tag_mc_#1_marks_seq } {3}
139         }
140       }
141       {
142         %stashed -> warning!!
143       }
144     }

```

```

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

```

(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 and 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.

```

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

```

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

```

175   \box_set_ht:Nn \l__tag_tmpa_box \c_zero_dim
176   \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.

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

```
179 \boxmaxdepth \@maxdepth
180 \box_use_drop:N \l__tag_tmpa_box
181 \vbox_unpack_drop:N #1
```

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

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

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

```
183 \nointerlineskip
184 \box_use_drop:N \l__tag_tmpb_box
185 }
186 }
```

(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.

```
187 \cs_new_protected:Npn \__tag_add_missing_mcs_to_stream:Nn #1#2
188 {
189 \__tag_check_if_active_mc:T {
```

First set up a temp box for trial splitting.

```
190 \vbadness\maxdimen
191 \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).

```
192 \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`

```
193 \exp_args:NNe
194 \seq_set_from_clist:Nn \l__tag_mc_firstmarks_seq
195 { \tex_splitfirstmarks:D \g__tag_mc_marks }
```

Some debugging info:

```
196 % \iow_term:n { First~ mark~ from~ this~ box: }
197 % \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!

```

198   \seq_if_empty:NTF \l__tag_mc_firstmarks_seq
199   {
200     \__tag_check_typeout_v:n
201     {
202       No~ marks~ so~ use~ saved~ bot~ mark:~
203       \seq_use:cn {g__tag_mc_#2_marks_seq} {,~} \iow_newline:
204     }
205     \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).

```

206     \seq_set_eq:NN \l__tag_mc_botmarks_seq \l__tag_mc_firstmarks_seq
207   }

```

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.

```

208   {
209     \__tag_check_typeout_v:n
210     {
211       Pick~ up~ new~ bot~ mark!
212     }
213     \exp_args:NNe
214     \seq_set_from_clist:Nn \l__tag_mc_botmarks_seq
215     { \tex_splitbotmarks:D   \g__tag_mc_marks }
216   }

```

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

```

217   \__tag_add_missing_mcs:Nn #1 {#2}
218   %%
219   \seq_gset_eq:cN {g__tag_mc_#2_marks_seq} \l__tag_mc_botmarks_seq
220   %%
221   }
222 }
223 \cs_set_eq:NN \tag_mc_add_missing_to_stream:Nn \__tag_add_missing_mcs_to_stream:Nn

```

(End of definition for `\tag_mc_add_missing_to_stream:Nn` and `\__tag_add_missing_mcs_to_stream:Nn`. This function is documented on page 83.)

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

`\__tag_mc_if_in: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.

```

224 \prg_new_conditional:Nnn \__tag_mc_if_in: {p,T,F,TF}

```

```

225 {
226   \bool_if:NTF \g__tag_in_mc_bool
227     { \prg_return_true: }
228     { \prg_return_false: }
229 }
230
231 \prg_new_eq_conditional:NNn \tag_mc_if_in: \__tag_mc_if_in: {p,T,F,TF}

```

(End of definition for __tag_mc_if_in:TF and \tag_mc_if_in:TF. This function is documented on page 82.)

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

```

232 % #1 tag, #2 properties
233 \cs_set_eq:NN \__tag_mc_bmc:n \pdf_bmc:n
234 \cs_set_eq:NN \__tag_mc_emc: \pdf_emc:
235 \cs_set_eq:NN \__tag_mc_bdc:nn \pdf_bdc:nn
236 \cs_set_eq:NN \__tag_mc_bdc_shipout:ee \pdf_bdc_shipout:ee

```

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

__tag_mc_bdc_mcid:nn
 __tag_mc_bdc_mcid:n This create a BDC mark with an /MCID key. Most of the work here is to get the current 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_handle_mcid:nn
 __tag_mc_handle_mcid:oo

```

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

```

```

260     \__tag_mc_bdc_mcid:nn {#1} {#2}
261   }
262
263   \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?

```

264   \cs_new_protected:Npn \__tag_mc_handle_stash:n #1 %1 mcidnum
265   {
266     \__tag_check_mc_used:n {#1}
267     \__tag_struct_kid_mc_gput_right:nn
268     { \g__tag_struct_stack_current_tl }
269     {#1}
270     \prop_gput:Nee \g__tag_mc_parenttree_prop
271     {#1}
272     { \g__tag_struct_stack_current_tl }
273   }
274   \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

```

275   \cs_new_protected:Npn \__tag_mc_bmc_artifact:
276   {
277     \__tag_mc_bmc:n {Artifact}
278   }
279   \cs_new_protected:Npn \__tag_mc_bmc_artifact:n #1
280   {
281     \__tag_mc_bdc:nn {Artifact}{/Type/#1}
282   }
283   \cs_new_protected:Npn \__tag_mc_handle_artifact:N #1
284   % #1 is a var containing the artifact type
285   {
286     \int_gincr:N \c@g__tag_MCID_abs_int
287     \tl_if_empty:NTF #1
288     { \__tag_mc_bmc_artifact: }
289     { \exp_args:No\__tag_mc_bmc_artifact:n {#1} }
290   }

```

(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.

```

291   \cs_new:Nn \__tag_get_data_mc_tag: { \g__tag_mc_key_tag_tl }
292   </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. The tag and the state is passed to the end command through a global var and a global boolean.

```

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

```

set default MC tags to structure:

```

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

```

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

```

337         \bool_if:NF \l__tag_mc_key_stash_bool
338         {

```

```

339         \socket_use:nn{tag/check/parent-child}
340         {
341             \__tag_mc_check_parent_child:o
342             { \g__tag_struct_stack_current_tl }
343         }
344         \__tag_mc_handle_stash:e { \int_use:N \c@g__tag_MCID_abs_int }
345
346     }
347 }
348 \group_end:
349 }
350 <*debug>
351 {
352     \__tag_debug_mc_begin_ignore:n { #1 }
353 }
354 </debug>
355 }
356 <*generic>
357 \cs_set_protected:Nn \tag_mc_end:
358 {
359     \__tag_check_if_active_mc:T
360     {
361 </generic>
362 <*debug>
363 \cs_set_protected:Nn \tag_mc_end:
364 {
365     \__tag_check_if_active_mc:TF
366     {
367         \__tag_debug_mc_end_insert:
368 </debug>
369         \__tag_check_mc_if_open:
370         \bool_gset_false:N \g__tag_in_mc_bool
371         \tl_gset:Nn \g__tag_mc_key_tag_tl { }
372         \__tag_mc_emc:
373         \__tag_mc_end_marks:
374     }
375 <*debug>
376 {
377     \__tag_debug_mc_end_ignore:
378 }
379 </debug>
380 }
381 </generic | debug>

```

(End of definition for \tag_mc_begin:n and \tag_mc_end:. These functions are documented on page 82.)

1.4 Keys

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

```

tag (mc-key)
raw (mc-key) 382 <*generic>
alt (mc-key)
actualtext (mc-key)
label (mc-key)
artifact (mc-key)

```

```

383 \keys_define:nn { __tag / mc }
384 {
385   tag .code:n = % the name (H,P,Span) etc
386   {
387     \tl_set:Ne \l__tag_mc_key_tag_tl { #1 }
388     \tl_gset:Ne \g__tag_mc_key_tag_tl { #1 }
389   },
390   raw .code:n =
391   {
392     \tl_put_right:Ne \l__tag_mc_key_properties_tl { #1 }
393   },
394   alt .code:n = % Alt property
395   {
396     \str_set_convert:Noon
397     \l__tag_tmpa_str
398     { #1 }
399     { default }
400     { utf16/hex }
401     \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
402     \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
403   },
404   alttext .meta:n = {alt=#1},

```

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

```

405   lang .code:n = % Lang property
406   {
407     \tl_put_right:Ne \l__tag_mc_key_properties_tl { /Lang~(#1) }
408   },
409   actualtext .code:n = % ActualText property
410   {
411     \tl_if_empty:oF{#1}
412     {
413       \str_set_convert:Noon
414       \l__tag_tmpa_str
415       { #1 }
416       { default }
417       { utf16/hex }
418       \tl_put_right:Nn \l__tag_mc_key_properties_tl { /ActualText~< }
419       \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
420     }
421   },
422   label .tl_set:N = \l__tag_mc_key_label_tl,
423   artifact .code:n =
424   {
425     \exp_args:Nne
426     \keys_set:nn
427     { __tag / mc }
428     { __artifact-bool, __artifact-type=#1 }
429   },
430   artifact .default:n = {notype}
431 }
432 </generic>

```

(End of definition for tag (mc-key) and others. These functions are documented on page 83.)

Ulrike Fischer
Version 1.0e, released 2026-08-21

Part VII

The tagpdf-mc-luacode module

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

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

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

`tag` : the type (a string)

`raw` : more properties (string)

`label`: a string.

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

`kids`: a array of tables

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

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

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

```
1 <@@=tag>
2 <*luamode>
3 \ProvidesExplPackage {tagpdf-mc-code-lua} {2026-08-21} {1.0e}
4   {tagpdf - mc code only for the luamode }
5 </luamode>
6 <*debug>
7 \ProvidesExplPackage {tagpdf-debug-lua} {2026-08-21} {1.0e}
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 83.)

`\_tag_mc_if_in_p:`

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

`\_tag_mc_if_in:TF`

```
68 \prg_new_conditional:Nnn \_tag_mc_if_in: {p,T,F,TF}
```

`\tag_mc_if_in_p:`

```
69 {
```

`\tag_mc_if_in:TF`

```
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 82.)

`\_tag_mc_lua_set_mc_type_attr:n`

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

`\_tag_mc_lua_set_mc_type_attr:o`

```
83 \cs_new_protected:Nn \_tag_mc_lua_set_mc_type_attr:n % #1 is a tag name
```

`\_tag_mc_lua_unset_mc_type_attr:`

```
84 {
```

```
85 %TODO ltx.__tag.func.get_num_from("#1") seems not to return a suitable number??
```

```
86 \tl_set:Nel__tag_tmpa_tl{\lua_now:e{ltx.__tag.func.output_num_from ("#1")}} }
```

```
87 \lua_now:e
```

```
88 {
```

```
89 tex.setattribute
```

```
90 (
```

```
91 "global",
```

```
92 luatexbase.attributes.g__tag_mc_type_attr,
```

```
93 \__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_mccabs
161     (
162         \g__tag_struct_stack_current_tl,#1
163     )
164 }
165 }
166 </luamode | debug>
167 <*luamode>
168 \cs_generate_variant:Nn \__tag_mc_handle_stash:n { e }

```

(End of definition for __tag_mc_handle_stash:n.)

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

```

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

```

set the default tag to the structure:

```

179         \tl_set_eq:NN \l__tag_mc_key_tag_tl \g__tag_struct_tag_tl
180         \tl_gset_eq:NN \g__tag_mc_key_tag_tl \g__tag_struct_tag_tl
181         \lua_now:e
182         {
183             ltx.__tag.func.store_mc_data(\__tag_get_mc_abs_cnt:,"tag","\g__tag_struct_tag_tl
184         }

```

2025-05-23 allow lang on the MC (not really spec conform, but does work in acrobat).

```

185         \tl_if_empty:NTF \l__tag_mc_lang_tl
186         {
187             \keys_set:nn { __tag / mc } { label={}, #1 }
188         }
189         {
190             \keys_set:nn { __tag / mc } { label={}, lang=\l__tag_mc_lang_tl, #1 }
191         }
192         %check that a tag or artifact has been used

```

```

193     \__tag_check_mc_tag:N \l__tag_mc_key_tag_tl
194     %set the attributes:
195     \__tag_mc_lua_set_mc_type_attr:o { \l__tag_mc_key_tag_tl }
196     \bool_if:NF \l__tag_mc_artifact_bool
197     { % store the absolute num name in a label:
198       \tl_if_empty:NF \l__tag_mc_key_label_tl
199       {
200         \__tag_mc_handle_mc_label:e { \l__tag_mc_key_label_tl }
201       }
202       % if not stashed record the absolute number
203       \bool_if:NF \l__tag_mc_key_stash_bool
204       {
205         \socket_use:nn{tag/check/parent-child}
206         {
207           \__tag_mc_check_parent_child:o
208           { \g__tag_struct_stack_current_tl }
209         }
210         \__tag_mc_handle_stash:e { \__tag_get_mc_abs_cnt: }
211       }
212     }
213     \group_end:
214   }
215 }

```

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

\tag_mc_end:

TODO: check how the use command must be guarded.

```

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

```

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

\tag_mc_reset_box:N

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

```

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

```

(End of definition for `\tag_mc_reset_box:N`. This function is documented on page 82.)

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

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

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

1.2 Key definitions

```

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

```

```

281         \tl_put_right:Ne \l__tag_mc_key_properties_tl { /Lang~(#1) }
282         \lua_now:e
283         {
284             ltx.__tag.func.store_mc_data
285             (
286                 \__tag_get_mc_abs_cnt:,"lang","/Lang~(#1)"
287             )
288         }
289     }
290 },
291 alttext .meta:n = {alt=#1},
292 actualtext .code:n = % Alt property
293 {
294     \tl_if_empty:oF{#1}
295     {
296         \str_set_convert:Noon
297         \l__tag_tmpa_str
298         { #1 }
299         { default }
300         { utf16/hex }
301         \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
302         \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
303         \lua_now:e
304         {
305             ltx.__tag.func.store_mc_data
306             (
307                 \__tag_get_mc_abs_cnt:,
308                 "actualtext",
309                 "/ActualText~<\str_use:N \l__tag_tmpa_str>"
310             )
311         }
312     }
313 },
314 label .code:n =
315 {
316     \tl_set:Nn\l__tag_mc_key_label_tl { #1 }
317     \lua_now:e
318     {
319         ltx.__tag.func.store_mc_data
320         (
321             \__tag_get_mc_abs_cnt:,"label","#1"
322         )
323     }
324 },
325 __artifact-store .code:n =
326 {
327     \lua_now:e
328     {
329         ltx.__tag.func.store_mc_data
330         (
331             \__tag_get_mc_abs_cnt:,"artifact","#1"
332         )
333     }
334 },

```

```

335     artifact .code:n      =
336     {
337         \exp_args:Nne
338         \keys_set:nn
339         { __tag / mc }
340         { __artifact-bool, __artifact-type=#1, tag=Artifact }
341         \exp_args:Nne
342         \keys_set:nn
343         { __tag / mc }
344         { __artifact-store=\l__tag_mc_artifact_type_tl }
345     },
346     artifact .default:n    = { notype }
347 }
348
349 </luamode>

```

(End of definition for tag (mc-key) and others. These functions are documented on page 83.)

Ulrike Fischer
Version 1.0e, released 2026-08-21

Part VIII

The tagpdf-struct module

Commands to create the structure

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 existing keywords are mostly related to the **Ref** key (an array).

- The keyword **tag** updates the tag of a structure (the **S** entry). So e.g. `\tag_struct_gput:nnn{\tag_get:n{struct_num}}{tag}{/Bla}` will replace the tag of the current structure by `/Bla`.
- The keyword **C** *replaces* the attribute-class array. The argument should be a comma list of PDF names `/TH-row,/justify`. In case that the list should be extended it is up to the user to get the original list first and manipulate it.
- 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_dest_parent** expects a destination name set with `\MakeLinkTarget` too. It then will refer to the parent of structure in which this `\MakeLinkTarget` was used—this is useful if the target command is, e.g., in a list and one wants to reference the surrounding list item structure.
- 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`.

<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.

<code>\tag_struct_map_function:N</code>	<code>\tag_struct_map_function:N {<function>}</code>
---	--

This command maps pairwise over the tag and the number stack. The argument should be a function with two arguments, where the first is two brace group (tag and rolemap) and the second a structure number. See the marginpar code for an example of use.

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 (<i>struct key</i>)	These keys handle associated files in the structure element.
AFref (<i>struct key</i>)	
AFinline (<i>struct key</i>)	AF = <i><object name></i>
AFinline-o (<i>struct key</i>)	AFref = <i><object reference></i>
texsource (<i>struct key</i>)	AF-inline = <i><text content></i>
mathml (<i>struct key</i>)	

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}
role/new-attribute* (setup-key)
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. Normally such attributes are added to the PDF only if used in some structure. The starred version will add them always.

```

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

```

```

\tag_attr_new:nn \tag_attr_new:nn {\name}{\Content}

```

As attributes have to be generated also quite often in code there is also a function, which is a bit faster than the key-val and also more control over the expansion behaviour.

`root-AF` (setup key)

```

root-AF = \object name

```

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

```

1 \<@@=tag>
2 \<*header>
3 \ProvidesExplPackage {tagpdf-struct-code} {2026-08-21} {1.0e}
4 {part of tagpdf - code related to storing structure}
5 \</header>

```

3 Variables

`\c@g__tag_struct_abs_int` Every structure will have a unique, absolute number.

```

6 \<base>\int_new:N \c@g__tag_struct_abs_int
7 \<base>\int_gset:Nn \c@g__tag_struct_abs_int { 1 }

```

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

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

```

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

```

(End of definition for `\g__tag_struct_objR_seq`.)

`\c__tag_struct_null_tl` In lua mode we have to test if the kids a null
`10 \tl_const:Nn\c__tag_struct_null_tl {null}`
 (End of definition for `\c__tag_struct_null_tl`.)

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

`\g__tag_struct_stack_seq` A stack sequence for the structure stack. When a sequence is opened it's number is put on the stack.
`12 \seq_new:N \g__tag_struct_stack_seq`
`13 \seq_gpush:Nn \g__tag_struct_stack_seq {1}`
 (End of definition for `\g__tag_struct_stack_seq`.)

`\g__tag_struct_tag_stack_seq` We will perhaps also need the tags. While it is possible to get them from the numbered stack, lets build a tag stack too. We create one which contains also the real tags and one with only the rolemapped tags for faster testing.
`14 \seq_new:N \g__tag_struct_tag_stack_seq`
`15 \seq_new:N \g__tag_struct_roletag_stack_seq`
`16 \seq_gpush:Nn \g__tag_struct_tag_stack_seq {{Root}}{StructTreeRoot}}`
`17 \seq_gpush:Nn \g__tag_struct_roletag_stack_seq {StructTreeRoot}`
 (End of definition for `\g__tag_struct_tag_stack_seq`.)

`\g__tag_struct_stack_current_tl` The global variable will hold the current structure number. It is already defined in `tagpdf-base`. The local temporary variable will hold the parent when we fetch it from the stack.
`\l__tag_struct_stack_parent_tmpa_tl`
`18 </package>`
`19 <base>\tl_new:N \g__tag_struct_stack_current_tl`
`20 <base>\tl_gset:Nn \g__tag_struct_stack_current_tl {\int_use:N\c@g__tag_struct_abs_int}`
`21 <*package>`
`22 \tl_new:N \l__tag_struct_stack_parent_tmpa_tl`
 (End of definition for `\g__tag_struct_stack_current_tl` and `\l__tag_struct_stack_parent_tmpa_tl`.)

In luatex we will store the structure number as attribute.

```

23 \sys_if_engine_luatex:TF
24 {
25   \cs_new:Npn \__tag_struct_set_attribute:
26   {
27     \lua_now:e
28     {
29       tex.setattribute
30       (
31         "global",
32         luatexbase.attributes.g__tag_structnum_attr,
33         \g__tag_struct_stack_current_tl
34       )

```

```

35     }
36   }
37 }
38 {
39   \cs_new_eq:NN \__tag_struct_set_attribute: \prg_do_nothing:
40 }

```

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

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

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

All properties have at least the keys

Type StructTreeRoot or StructElem

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

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

```

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

```

```

71     E,                % abbreviation
72     ActualText,
73     AF,                %pdf 2.0, array of dict, associated files
74     NS,                %pdf 2.0, dict, namespace
75     PhoneticAlphabet,  %pdf 2.0
76     Phoneme            %pdf 2.0
77 }

```

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

3.1 Variables used by the keys

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

```

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

```

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

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

```

86 \prop_new_linked:N \g__tag_struct_label_num_prop

```

(End of definition for `\g__tag_struct_label_num_prop`.)

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

```

87 \bool_new:N \l__tag_struct_elem_stash_bool

```

(End of definition for `\l__tag_struct_elem_stash_bool`.)

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

```

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

```

(End of definition for `\l__tag_struct_addkid_tl`.)

3.2 Variables used by tagging code of basic elements

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

```

90 </package>
91 <base>\prop_new_linked:N \g__tag_struct_dest_num_prop
92 <*package>

```

(End of definition for `\g__tag_struct_dest_num_prop`.)

`\tag_struct_recordtarget:` This records for the current target (as stored in `\@currentHref`) the structure number

```

93 \cs_new_protected:Npn \tag_struct_recordtarget:
94 {
95   \tl_if_blank:VF \@currentHref
96   {
97     \prop_gput:Nee
98       \g__tag_struct_dest_num_prop
99       {\@currentHref}
100     {\g__tag_struct_stack_current_tl}
101   }
102 }

```

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

`\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.

```

103 \prop_new_linked:N \g__tag_struct_ref_by_dest_prop
104 \</package>

```

(End of definition for `\g__tag_struct_ref_by_dest_prop.`)

4 Commands

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

```

105 \<*package|debug>
106 \<package>\cs_new_protected:Npn \__tag_struct_prop_gput:nnn #1 #2 #3
107 \<debug>\cs_set_protected:Npn \__tag_struct_prop_gput:nnn #1 #2 #3
108 {
109   \__tag_struct_prop_gput:cnn
110     { \g__tag_struct_#1_prop }{#2}{#3}
111 \<debug>\prop_gput:cnn { \g__tag_struct_debug_#1_prop } {#2} {#3}
112 }
113 \cs_generate_variant:Nn \__tag_struct_prop_gput:nnn {onn,nne,nee,nnn}
114 \</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.

```

115 \<*package>
116 \tl_gset:Nn \g__tag_struct_stack_current_tl {1}

```

`\__tag_pdf_name_e:n`

```

117 \cs_new:Npn \__tag_pdf_name_e:n #1{\pdf_name_from_unicode_e:n{#1}}
118 \</package>

```

(End of definition for `\__tag_pdf_name_e:n`.)

```

g__tag_struct_1_prop
g__tag_struct_kids_1_seq
119 <*package>
120 \__tag_prop_new:c { g__tag_struct_1_prop }
121 \__tag_seq_new:c { g__tag_struct_kids_1_seq }
122
123 \__tag_struct_prop_gput:nne
124 { 1 }
125 { Type }
126 { \pdf_name_from_unicode_e:n {StructTreeRoot} }
127
128 \__tag_struct_prop_gput:nne
129 { 1 }
130 { S }
131 { \pdf_name_from_unicode_e:n {StructTreeRoot} }
132
133 \__tag_struct_prop_gput:nne
134 { 1 }
135 { tag }
136 { {StructTreeRoot}{pdf} }
137
138 \__tag_struct_prop_gput:nne
139 { 1 }
140 { rolemap }
141 { {StructTreeRoot}{pdf} }
142
143 \__tag_struct_prop_gput:nne
144 { 1 }
145 { parentrole }
146 { {StructTreeRoot}{pdf} }
147

```

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

```

148 \pdf_version_compare:NnF < {2.0}
149 {
150   \__tag_struct_prop_gput:nne
151   { 1 }
152   { Namespaces }
153   { \pdf_object_ref:n { __tag/tree/namespaces } }
154 }
155 </package>

```

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

```

156 <debug>\prop_new:c { g__tag_struct_debug_1_prop }
157 <debug>\seq_new:c { g__tag_struct_debug_kids_1_seq }
158 <debug>\prop_gset_eq:cc { g__tag_struct_debug_1_prop }{ g__tag_struct_1_prop }
159 <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`

```
160 (*package)
161 \cs_new:Npn \_tag_struct_get_id:n #1 %#1=struct num
162 {
163   (
164     ID.
165     \prg_replicate:nn
166     { \int_abs:n{\g__tag_tree_id_pad_int - \tl_count:e { \int_to_arabic:n { #1 } }} }
167     { 0 }
168     \int_to_arabic:n { #1 }
169   )
170 }
```

(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.

```
171 \pdf_version_compare:NnTF < {2.0}
172 {
173   \cs_new_protected:Npn \_tag_struct_set_tag_info:nnn #1 #2 #3
174     %#1 structure number, #2 tag, #3 NS
175   {
176     \_tag_struct_prop_gput:nne
177     { #1 }
178     { S }
179     { \pdf_name_from_unicode_e:n {#2} } %
180     \_tag_struct_prop_gput:nnn
181     { #1 }
182     { tag }
183     { {#2} {} }
184   }
185 }
186 {
187   \cs_new_protected:Npn \_tag_struct_set_tag_info:nnn #1 #2 #3
188   {
189     \_tag_struct_prop_gput:nne
190     { #1 }
191     { S }
192     { \pdf_name_from_unicode_e:n {#2} } %
193     \prop_get:NnNT \g__tag_role_NS_prop {#3} \l__tag_get_tmpc_tl
194     {
195       \_tag_struct_prop_gput:nne
196       { #1 }
197       { NS }
198       { \l__tag_get_tmpc_tl } %
199     }
200     \_tag_struct_prop_gput:nnn
```

```

201         { #1 }
202         { tag }
203         { {#2} {#3} }
204     }
205 }
206 \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.

```

207 \cs_new_protected:Npn \__tag_struct_get_role:nnNN #1 #2 #3 #4
208     % #1 : struct num,
209     % #2 : rolemap or parentrole
210     % #3 : tlvar for tag (rolemapped)
211     % #4 : tlvar for NS (rolemapped, so standard or empty or UNKNOWN)
212     {
213     \prop_get:cnNTF
214     { g__tag_struct_#1_prop }
215     { #2 }
216     \l__tag_get_tmpc_tl
217     {
218         \tl_set:Ne #3{\exp_last_unbraced:No\use_i:nn { \l__tag_get_tmpc_tl }}
219         \tl_set:Ne #4{\exp_last_unbraced:No\use_ii:nn { \l__tag_get_tmpc_tl }}
220     }
221     {
222         \tl_clear:N#3
223         \tl_clear:N#4
224     }
225 }
226 \cs_generate_variant:Nn \__tag_struct_get_role:nnNN {enNN}
(End of definition for \__tag_struct_get_role:nnNN.)

```

4.4 Handlings kids

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

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

```

227 \cs_new:Npn \__tag_struct_mcid_dict:n #1 % #1 MCID absnum
228     {
229     <<
230     /Type \c_space_tl /MCR \c_space_tl
231     /Pg

```

```

232         \c_space_t1
233         \pdf_pageobject_ref:n { \property_ref:enn{mcid-#1}{tagabspage}{1} }
234         /MCID \c_space_t1 \property_ref:enn{mcid-#1}{tagmcid}{1}
235     >>
236 }
237 (/package)

238 (*package| debug)
239 (package)\cs_new_protected:Npn \__tag_struct_kid_mc_gput_right:nn #1 #2
240 (debug)\cs_set_protected:Npn \__tag_struct_kid_mc_gput_right:nn #1 #2
241 %%#1 structure num, #2 MCID absnum%
242 {
243     \__tag_seq_gput_right:ce
244     { g__tag_struct_kids_#1_seq }
245     {
246         \__tag_struct_mcid_dict:n {#2}
247     }
248 (debug) \seq_gput_right:cn
249 (debug) { g__tag_struct_debug_kids_#1_seq }
250 (debug) {
251 (debug) MC~#2
252 (debug) }
253 \__tag_seq_gput_right:cn
254 { g__tag_struct_kids_#1_seq }
255 {
256     \prop_item:Nn \g__tag_struct_cont_mc_prop {#2}
257 }
258 }
259 (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.

```

260 (package)\cs_new_protected:Npn \__tag_struct_kid_struct_gput_right:nn #1 #2
261 (debug)\cs_set_protected:Npn \__tag_struct_kid_struct_gput_right:nn #1 #2
262 %%#1 num of parent struct, #2 kid struct
263 {
264     \__tag_seq_gput_right:ce
265     { g__tag_struct_kids_#1_seq }
266     {
267         \pdf_object_ref_indexed:nn { __tag/struct }{ #2 }
268     }
269 (debug) \seq_gput_right:cn
270 (debug) { g__tag_struct_debug_kids_#1_seq }
271 (debug) {
272 (debug) Struct~#2
273 (debug) }
274 }
275 (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.

```

276 (package)\cs_new_protected:Npn \__tag_struct_kid_struct_gput_left:nn #1 #2

```

```

277 <debug>\cs_set_protected:Npn\__tag_struct_kid_struct_gput_left:nn #1 #2
278 %%#1 num of parent struct, #2 kid struct
279 {
280   \__tag_seq_gput_left:ce
281   { g__tag_struct_kids_#1_seq }
282   {
283     \pdf_object_ref_indexed:nn { __tag/struct }{ #2 }
284   }
285 <debug>   \seq_gput_left:cn
286 <debug>   { g__tag_struct_debug_kids_#1_seq }
287 <debug>   {
288 <debug>     Struct~#2
289 <debug>   }
290 }
291 <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

```

292 <package>\cs_new_protected:Npn\__tag_struct_kid_OBJR_gput_right:nnn #1 #2 #3
293 <package>
294 <package>
295 <debug>\cs_set_protected:Npn\__tag_struct_kid_OBJR_gput_right:nnn #1 #2 #3
296 %%#1 num of parent struct,#2 obj reference,#3 page object reference
297 {
298   \pdf_object_unnamed_write:nn
299   { dict }
300   {
301     /Type/OBJR/Obj~#2/Pg~#3
302   }
303   \__tag_seq_gput_right:ce
304   { g__tag_struct_kids_#1_seq }
305   {
306     \pdf_object_ref_last:
307   }
308 <debug>   \seq_gput_right:ce
309 <debug>   { g__tag_struct_debug_kids_#1_seq }
310 <debug>   {
311 <debug>     OBJR~reference
312 <debug>   }
313 }
314 </package|debug>
315 <*package>
316 \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.

```

317 \cs_new_protected:Npn\__tag_struct_exchange_kid_command:N #1 %%#1 = seq var

```

```

318 {
319   \seq_gpop_left:NN #1 \l__tag_tmpa_tl
320   \tl_replace_once:Nnn \l__tag_tmpa_tl
321     {\__tag_mc_insert_mcid_kids:n}
322     {\__tag_mc_insert_mcid_single_kids:n}
323   \seq_gput_left:No #1 { \l__tag_tmpa_tl }
324 }
325
326 \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.

```

327 \cs_new_protected:Npn \__tag_struct_fill_kid_key:n #1 % #1 is the struct num
328 {
329   \bool_if:NF \g__tag_mode_lua_bool
330   {
331     \seq_clear:N \l__tag_tmpa_seq
332     \seq_map_inline:cn { g__tag_struct_kids_#1_seq }
333       { \seq_put_right:Ne \l__tag_tmpa_seq { ##1 } }
334     %\seq_show:c { g__tag_struct_kids_#1_seq }
335     %\seq_show:N \l__tag_tmpa_seq
336     \seq_remove_all:Nn \l__tag_tmpa_seq {}
337     %\seq_show:N \l__tag_tmpa_seq
338     \seq_gset_eq:cN { g__tag_struct_kids_#1_seq } \l__tag_tmpa_seq
339   }
340
341   \int_case:nnF
342     {
343       \seq_count:c
344       {
345         g__tag_struct_kids_#1_seq
346       }
347     }
348     {
349       { 0 }
350       { } %no kids, do nothing
351       { 1 } % 1 kid, insert
352       {
353         % in this case we need a special command in
354         % luamode to get the array right. See issue #13
355         \sys_if_engine luatex:TF
356         {
357           \__tag_struct_exchange_kid_command:c
358           {g__tag_struct_kids_#1_seq}

```

check if we get null

```

359   \tl_set:N\l__tag_tmpa_tl
360     {\use:e{\seq_item:cn {g__tag_struct_kids_#1_seq} {1}}}
361   \tl_if_eq:NNF\l__tag_tmpa_tl \c__tag_struct_null_tl
362   {
363     \__tag_struct_prop_gput:nne
364     {#1}

```

```

365         {K}
366         {
367             \seq_item:cn
368             {
369                 g__tag_struct_kids_#1_seq
370             }
371             {1}
372         }
373     }
374 }
375 {
376     \__tag_struct_prop_gput:nne
377     {#1}
378     {K}
379     {
380         \seq_item:cn
381         {
382             g__tag_struct_kids_#1_seq
383         }
384         {1}
385     }
386 }
387 } %
388 }
389 { %many kids, use an array
390     \__tag_struct_prop_gput:nne
391     {#1}
392     {K}
393     {
394         [
395             \seq_use:cn
396             {
397                 g__tag_struct_kids_#1_seq
398             }
399             {
400                 \c_space_tl
401             }
402         ]
403     }
404 }
405 }
406

```

(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.

```

407 \cs_new_protected:Npn \__tag_struct_get_dict_content:nN #1 #2 %#1: structure num
408 {
409     \tl_clear:N #2
410     \prop_map_inline:cn { g__tag_struct_#1_prop }
411     {

```

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.

```

412     \cs_if_exist_use:cTF {__tag_struct_format_##1:nnN}
413     {
414         {##1}{##2}#2
415     }
416     {
417         \tl_put_right:Ne #2 { \c_space_tl/##1~##2 }
418     }
419 }
420 }

```

(End of definition for `__tag_struct_get_dict_content: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.

```

421 \cs_new:Nn\__tag_struct_format_rolemap:nnN{}
422 \cs_new:Nn\__tag_struct_format_parentrole:nnN{}
423 \cs_new:Nn\__tag_struct_format_tag:nnN{}

```

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

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

```

424 \cs_new_protected:Nn\__tag_struct_format_parentnum:nnN
425 {
426     \tl_put_right:Ne #3 { ~/P~\pdf_object_ref_indexed:nn { __tag/struct} { #2 } }
427 }

```

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

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

```

428 \cs_new_protected:Nn\__tag_struct_format_Ref:nnN
429 {
430     \tl_put_right:Nn #3 { ~/#1~[ } %]
431     \clist_map_inline:nn{ #2 }
432     {
433         ##1 #3
434     }
435     \tl_put_right:Nn #3
436     { %[
437         \c_space_tl]
438     }
439 }

```

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

This writes out the structure object. This is done in the finish code, in the tree module and guarded by the tree boolean.

```

440 \cs_new_protected:Npn \__tag_struct_write_obj:n #1 % #1 is the struct num
441 {
442     \prop_if_exist:cTF { g__tag_struct_#1_prop }
443     {

```

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.

```

444     \prop_get:cnNF { g__tag_struct_#1_prop } {parentnum}\l__tag_tmpb_tl
445     {
446     %       \prop_gput:cne { g__tag_struct_#1_prop } {P}
447     %       {\pdf_object_ref_indexed:nn { __tag/struct }{1}}
448     \prop_gput:cne { g__tag_struct_#1_prop } {parentnum}{2}
449     \prop_gput:cne { g__tag_struct_#1_prop } {S}{/Artifact}
450     \seq_if_empty:cF {g__tag_struct_kids_#1_seq}
451     {
452         \msg_warning:nnee
453         {tag}
454         {struct-orphan}
455         { #1 }
456         {\seq_count:c{g__tag_struct_kids_#1_seq}}
457     }
458     }
459     \__tag_struct_fill_kid_key:n { #1 }
460     \__tag_struct_get_dict_content:nN { #1 } \l__tag_tmpa_tl
461     \pdf_object_write_indexed:nnne
462     { __tag/struct }{ #1 }
463     {dict}
464     {
465         \l__tag_tmpa_tl\c_space_tl
466         /ID~\__tag_struct_get_id:n{#1}
467     }
468     }
469     {
470     {
471         \msg_error:nnn { tag } { struct-no-objnum } { #1}
472     }
473     }

```

(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:

474 \cs_new_protected:Npn \__tag_struct_insert_annot:nn #1 #2
475   %#1 object reference to the annotation/xform
476   %#2 structparent number
477   {
478     \bool_if:NT \g__tag_active_struct_bool
479     {
480       %get the number of the parent structure:
481       \seq_get:NNF
482         \g__tag_struct_stack_seq
483         \l__tag_struct_stack_parent_tmpa_tl
484         {
485           \msg_error:nn { tag } { struct-faulty-nesting }
486         }
487       %put the obj number of the annot in the kid entry, this also creates
488       %the OBJR object
489       \__tag_property_record:nn {@tag@objr@page@#2 }{ tagabspage }
490       \__tag_struct_kid_OBJR_gput_right:eee
491       {
492         \l__tag_struct_stack_parent_tmpa_tl
493       }
494       {
495         #1 %
496       }
497       {
498         \pdf_pageobject_ref:n
499         { \property_ref:nnn {@tag@objr@page@#2 }{ tagabspage }{1} }
500       }
501       % add the parent obj number to the parent tree:
502       % the command always expands its arguments!
503       \__tag_parenttree_add_objr:nn
504       {
505         #2
506       }
507       {
508         \pdf_object_ref_indexed:nn
509         { __tag/struct }{ \l__tag_struct_stack_parent_tmpa_tl }
510       }
511       % increase the int:
512       \int_gincr:N \c@g__tag_parenttree_obj_int
513     }
514   }

(End of definition for \__tag_struct_insert_annot:nn.)

```

__tag_struct_insert_annot_shipout:nnn

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

```

515 \cs_new_protected:Npn \__tag_struct_insert_annot_shipout:nnn #1#2#3

```

```

516 % #1 structnum, #2 object reference, #3 StructParentNum
517 {
518   \_tag_struct_kid_OBJR_gput_right:eee
519   {
520     #1
521   }
522   {
523     #2
524   }
525   {
526     \pdf_pageobject_ref:n
527     { \int_use:N \g_shipout_readonly_int } %
528   }
529 % add the parent obj number to the parent tree:
530 % the command always expands its arguments!
531 \_tag_parenttree_add_objr:nn
532 {
533   #3
534 }
535 {
536   \pdf_object_ref_indexed:nn
537   { \_tag/struct }{ #1 }
538 }
539 }

```

(End of definition for _tag_struct_insert_annot_shipout:nnn.)

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

```

540 \cs_new:Npn \_tag_get_data_struct_tag:
541 {
542   \exp_args:Ne
543   \tl_tail:n
544   {
545     \prop_item:cn {g__tag_struct_\g__tag_struct_stack_current_tl _prop}{S}
546   }
547 }

```

(End of definition for _tag_get_data_struct_tag:.)

_tag_get_data_struct_tag:N this command allows \tag_get:nN to get the current structure tag with the keyword **struct_tag**.

```

548 \cs_new_protected:Npn \_tag_get_data_struct_tag:N #1
549 {
550   \seq_get:NN\g__tag_struct_tag_stack_seq \l__tag_tmpb_tl
551   \tl_set:Ne #1 {\exp_last_unbraced:NV\use_i:nn\l__tag_tmpb_tl}
552 }

```

(End of definition for _tag_get_data_struct_tag:N.)

_tag_get_data_struct_num:N this command allows \tag_get:nN to get the current structure number with the keyword **struct_num**.

```

553 \cs_new_protected:Npn \_tag_get_data_struct_num:N #1
554 {
555   \seq_get:NN \g__tag_struct_stack_seq #1
556 }

```

(End of definition for _tag_get_data_struct_num:N.)

_tag_get_data_struct_C:nN This command allows \tag_get:nnN to get the C entries of a structure as a comma list with the keyword C. The items are PDF names. TODO: this is too complicated, the C entry should be stored differently to allow easier access, but that can only be done after 2026-11-01, as the table code currently relies on the internal representation.

```

557 \cs_new_protected:Npn \_tag_get_data_struct_C:nN #1 #2
558 {
559   \prop_get:cnNTF {g__tag_struct_ #1 _prop}
560   {C}
561   \l__tag_tmp_unused_tl
562   {
563     \str_remove_once:Nn \l__tag_tmp_unused_tl {}
564     \str_remove_once:Nn \l__tag_tmp_unused_tl {}
565     \seq_set_split:Nne \l__tag_tmpa_seq {/} {\l__tag_tmp_unused_tl}
566     \seq_remove_all:Nn \l__tag_tmpa_seq {}
567     \seq_set_map:NNn \l__tag_tmpa_seq \l__tag_tmpa_seq {/##1}
568     \tl_set:Ne #2 {\seq_use:Nn \l__tag_tmpa_seq {,}}
569   }
570   { \tl_set:Nn #2 {} }
571 }

```

(End of definition for _tag_get_data_struct_C:nN.)

_tag_get_data_struct_id: this command allows \tag_get:n to get the current structure id with the keyword **struct_id**.

```

572 \cs_new:Npn \_tag_get_data_struct_id:
573 {
574   \_tag_struct_get_id:n {\g__tag_struct_stack_current_tl}
575 }
576 \end{package}

```

(End of definition for _tag_get_data_struct_id:.)

_tag_get_data_struct_num: this command allows \tag_get:n to get the current structure number with the keyword **struct_num**. We will need to handle nesting

```

577 \begin{base}
578 \cs_new:Npn \_tag_get_data_struct_num:
579 {
580   \g__tag_struct_stack_current_tl
581 }
582 \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.

```

583 \begin{base}
584 \cs_new:Npn \_tag_get_data_struct_counter:
585 {
586   \int_use:N \c@g__tag_struct_abs_int
587 }
588 \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

```

589 (*package)
590 \cs_new_protected:Npn \_tag_struct_check_parent_child_aux:nnnnN #1#2#3#4#5
591 {
592 % #1 structure number of parent
593 % #2 key to use to retrieve role of parent (either rolemap or parentrole field)
594 % #3 structure number of parent
595 % #4 key to use to retrieve role of child (either rolemap or parentrole field)
596 % #5 t1 for return value

```

get parent rolemap

```

597 \_tag_struct_get_role:nnNN
598 {#1}
599 {#2}
600 \l__tag_get_parent_tmpa_tl
601 \l__tag_get_parent_tmpb_tl

```

get child rolemap

```

602 \_tag_struct_get_role:nnNN
603 {#3}
604 {#4}
605 \l__tag_get_child_tmpa_tl
606 \l__tag_get_child_tmpb_tl

```

check

```

607 \_tag_role_check_parent_child:ooooN
608 { \l__tag_get_parent_tmpa_tl } % rolemapped from above
609 { \l__tag_get_parent_tmpb_tl } % rolemapped from above
610 { \l__tag_get_child_tmpa_tl } %
611 { \l__tag_get_child_tmpb_tl } %
612 #5
613 }

```

(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.

```

614 \cs_new_protected:Npn \_tag_struct_check_parent_child:nn #1 #2
615 % #1 structure number of parent
616 % #2 structure number of child. %
617 % This assumes that the fields rolemap/parentrole has already been filled.
618 {

```

This records if logging is on

```

619 \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
620 {
621 \prop_get:cnN{g__tag_struct_#1_prop}{tag}\l__tag_get_parent_tmpa_tl
622 \prop_get:cnN{g__tag_struct_#2_prop}{tag}\l__tag_get_parent_tmpb_tl
623 \msg_note:nnee
624 { tag }
625 { role-parent-child-check }
626 {
627 \quark_if_no_value:NTF \l__tag_get_parent_tmpa_tl
628 {??}

```

```

629         {
630             \exp_last_unbraced:No\use_ii:nn
631             { \l__tag_get_parent_tmpa_tl }
632             :
633             \exp_last_unbraced:No\use_i:nn
634             { \l__tag_get_parent_tmpa_tl }
635         }
636     }
637     {
638         \quark_if_no_value:NTF \l__tag_get_parent_tmpb_tl
639         {??}
640         {
641             \exp_last_unbraced:No\use_ii:nn
642             { \l__tag_get_parent_tmpb_tl }
643             :
644             \exp_last_unbraced:No\use_i:nn
645             { \l__tag_get_parent_tmpb_tl }
646         }
647     }
648 }
649 \__tag_struct_check_parent_child_aux:nnnnN
650 {#1}
651 {rolemap}
652 {#2}
653 {rolemap}
654 \l__tag_parent_child_check_tl

```

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

```

655 \int_compare:nNnT {\l__tag_parent_child_check_tl} = { \c__tag_role_rule_checkparent_tl }
656 {
657     \__tag_struct_check_parent_child_aux:nnnnN
658     {#1}
659     {parentrole}
660     {#2}
661     {rolemap}
662     \l__tag_parent_child_check_tl
663 }
664 \__tag_check_struct_forbidden_parent_child:onm
665 {\l__tag_parent_child_check_tl}
666 {#1}
667 {#2}
668 }
669 \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 (par-

ent) and if yet then test rolemmap (child) against rolemmap (parent) and if =7 rolemmap(child) against parentrole(parent). that is again the normal check.

```

670 \cs_new_protected:Npn \__tag_struct_use_check_parent_child:nn #1 #2
671 % #1 structure number of parent
672 % #2 structure number of child. %
673 {
674   \__tag_struct_get_role:enNN
675   {#2}
676   {rolemmap}
677   \l__tag_get_child_tmpa_tl
678   \l__tag_get_child_tmpb_tl
679   \str_case:onTF { \l__tag_get_child_tmpa_tl }
680   {
681     {Part} {}
682     {Div} {}
683     {NonStruct} {}
684   }
685   { %child=Part etc
686     \__tag_struct_get_role:enNN
687     {#2}
688     {parentrole}
689     \l__tag_get_child_tmpa_tl
690     \l__tag_get_child_tmpb_tl
691     \str_if_eq:noTF
692     {STASHED}{\l__tag_get_child_tmpa_tl}
693     {
694       % warn about unknown relationship
695     }
696     {
697       % test if
698       \__tag_struct_get_role:enNN
699       {#1}
700       {parentrole}
701       \l__tag_get_parent_tmpa_tl
702       \l__tag_get_parent_tmpb_tl
703       \tl_if_eq:NNTF\l__tag_get_parent_tmpa_tl \l__tag_get_child_tmpa_tl
704       {
705         \__tag_struct_check_parent_child:nn {#1}{#2}
706       }
707       {
708         %warn that parent-tag was misused.
709       }
710     }
711   }
712   {
713     %child not Part etc, normal parent child test.
714     \__tag_struct_check_parent_child:nn {#1}{#2}
715   }
716 }
717 \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.

```

718 \socket_new:nn { tag/struct/tag }{1}
719 \socket_new_plug:nnn { tag/struct/tag }{ latex-tags }
720 {
721   \prop_get:NeNTF \g__tag_role_tags_NS_prop {#1} \l__tag_tmp_unused_tl
722   {
723     \seq_set_split:Nne \l__tag_tmpa_seq { / }
724     {#1/\l__tag_tmp_unused_tl}
725   }
726   {
727     \seq_set_split:Nne \l__tag_tmpa_seq { / }
728     {#1/}
729   }
730   \tl_gset:Ne \g__tag_struct_tag_tl { \seq_item:Nn\l__tag_tmpa_seq {1} }
731   \tl_gset:Ne \g__tag_struct_tag_NS_tl{ \seq_item:Nn\l__tag_tmpa_seq {2} }
732   \__tag_check_structure_tag:N \g__tag_struct_tag_tl
733 }
734
735 \socket_new_plug:nnn { tag/struct/tag }{ pdf-tags }
736 {
737   \prop_get:NeNTF \g__tag_role_tags_NS_prop {#1} \l__tag_tmp_unused_tl
738   {
739     \seq_set_split:Nne \l__tag_tmpa_seq { / }
740     {#1/\l__tag_tmp_unused_tl}
741   }
742   {
743     \seq_set_split:Nne \l__tag_tmpa_seq { / }
744     {#1/}
745   }
746   \tl_gset:Ne \g__tag_struct_tag_tl { \seq_item:Nn\l__tag_tmpa_seq {1} }
747   \tl_gset:Ne \g__tag_struct_tag_NS_tl{ \seq_item:Nn\l__tag_tmpa_seq {2} }
748   \__tag_role_get:ooNN
749   { \g__tag_struct_tag_tl }
750   { \g__tag_struct_tag_NS_tl}
751   \l__tag_tmpa_tl
752   \l__tag_tmpb_tl
753   \tl_gset:Ne \g__tag_struct_tag_tl {\l__tag_tmpa_tl}
754   \tl_gset:Ne \g__tag_struct_tag_NS_tl{\l__tag_tmpb_tl}
755   \__tag_check_structure_tag:N \g__tag_struct_tag_tl
756 }
757 \socket_assign_plug:nn { tag/struct/tag } {latex-tags}

```

```

label (struct key)
stash (struct key) 758 \keys_define:nn { __tag / struct }
parent (struct key) 759 {
firstkid (struct key) 760   label .code:n      =
tag (struct key) 761   {
title (struct key) 762     \prop_gput:Nee\g__tag_struct_label_num_prop
title-o (struct key)
alt (struct key)
actualtext (struct key)
lang (struct key)
ref (struct key)
E (struct key)
phoneme (struct key)

```

```

763         {#1}{\int_use:N \c@g__tag_struct_abs_int}
764     \__tag_property_record:eo
765     {tagpdfstruct-#1}
766     { \c__tag_property_struct_clist }
767 },
768 stash .bool_set:N      = \l__tag_struct_elem_stash_bool,
769 parent .code:n         =
770 {
771     \bool_lazy_and:nnTF
772     {
773         \prop_if_exist_p:c { g__tag_struct\_int_eval:n {#1}_prop }
774     }
775     {
776         \int_compare_p:nNn {#1}<{\c@g__tag_struct_abs_int}
777     }
778     { \tl_set:Ne \l__tag_struct_stack_parent_tmpa_tl { \int_eval:n {#1} } }
779     {
780         \msg_warning:nnee { tag } { struct-unknown }
781         { \int_eval:n {#1} }
782         { parent-key-ignored }
783     }
784 },
785 parent .default:n      = {-1},
786 parent-tag .code:n     =
787 {
788     \prop_get:NeNTF \g__tag_role_tags_NS_prop {#1} \l__tag_tmp_unused_tl
789     {
790         \seq_set_split:Nne \l__tag_tmpa_seq { / }
791         {#1/\l__tag_tmp_unused_tl}
792     }
793     {
794         \seq_set_split:Nne \l__tag_tmpa_seq { / }
795         {#1/}
796     }
797     \tl_set:Ne \l__tag_struct_parenttag_tl { \seq_item:Nn\l__tag_tmpa_seq {1} }
798     \tl_set:Ne \l__tag_struct_parenttag_NS_tl{ \seq_item:Nn\l__tag_tmpa_seq {2} }
799     \__tag_role_get:ooNN
800     { \l__tag_struct_parenttag_tl }
801     { \l__tag_struct_parenttag_NS_tl}
802     \l__tag_tmpa_tl
803     \l__tag_tmpb_tl
804     \tl_set:No \l__tag_struct_parenttag_tl { \l__tag_tmpa_tl }
805     \tl_set:No \l__tag_struct_parenttag_NS_tl{ \l__tag_tmpb_tl }
806     \__tag_check_structure_tag:N \l__tag_struct_parenttag_tl
807 },
808 firstkid .code:n = { \tl_set:Nn \l__tag_struct_addkid_tl {left} },
809 tag .code:n      = % S property
810 {
811     \socket_use:nn { tag/struct/tag }{#1}
812 },
813 title .code:n    = % T property
814 {
815     \str_set_convert:Nnnn
816     \l__tag_tmpa_str

```

```

817         { #1 }
818         { default }
819         { utf16/hex }
820     \_tag_struct_prop_gput:nne
821     { \int_use:N \c@g__tag_struct_abs_int }
822     { T }
823     { <\l__tag_tmpa_str> }
824 },
825 title-o .code:n          = % T property
826 {
827     \str_set_convert:Nonn
828     \l__tag_tmpa_str
829     { #1 }
830     { default }
831     { utf16/hex }
832     \_tag_struct_prop_gput:nne
833     { \int_use:N \c@g__tag_struct_abs_int }
834     { T }
835     { <\l__tag_tmpa_str> }
836 },
837 alt .code:n              = % Alt property
838 {
839     \tl_if_empty:oF{#1}
840     {
841         \str_set_convert:Noon
842         \l__tag_tmpa_str
843         { #1 }
844         { default }
845         { utf16/hex }
846         \_tag_struct_prop_gput:nne
847         { \int_use:N \c@g__tag_struct_abs_int }
848         { Alt }
849         { <\l__tag_tmpa_str> }
850     }
851 },
852 alttext .meta:n = {alt=#1},
853 actualtext .code:n = % ActualText property
854 {
855     \tl_if_empty:oF{#1}
856     {
857         \str_set_convert:Noon
858         \l__tag_tmpa_str
859         { #1 }
860         { default }
861         { utf16/hex }
862         \_tag_struct_prop_gput:nne
863         { \int_use:N \c@g__tag_struct_abs_int }
864         { ActualText }
865         { <\l__tag_tmpa_str>}
866     }
867 },
868 phoneme .code:n = % Phoneme property
869 {
870     \tl_if_empty:oF{#1}

```

```

871         {
872             \str_set_convert:Noon
873             \l__tag_tmpa_str
874             { #1 }
875             { default }
876             { utf16/hex }
877             \__tag_struct_prop_gput:nne
878             { \int_use:N \c@g__tag_struct_abs_int }
879             { Phoneme }
880             { <\l__tag_tmpa_str>}
881         }
882     },
883     lang .code:n          = % Lang property
884     {
885         \__tag_struct_prop_gput:nne
886         { \int_use:N \c@g__tag_struct_abs_int }
887         { Lang }
888         { (#1) }
889     },
890 }

```

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

```

\__tag_struct_Ref_obj:nN
\__tag_struct_Ref_label:nN
\__tag_struct_Ref_dest:nN
\__tag_struct_Ref_dest_parent:nN
\__tag_struct_Ref_num:nN

```

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

```

891 \cs_new_protected:Npn \__tag_struct_Ref_obj:nN #1 #2 %#1 a object reference
892 {
893     \tl_put_right:Nx#2
894     {
895         \c_space_tl#1
896     }
897 }
898
899 \cs_new_protected:Npn \__tag_struct_Ref_label:nN #1 #2 %#1 a label
900 {
901     \prop_get:NnNTF \g__tag_struct_label_num_prop {#1} \l__tag_tmpb_tl
902     {
903         \tl_put_right:Nx#2
904         {
905             \c_space_tl\tag_struct_object_ref:e{ \l__tag_tmpb_tl }
906         }
907     }
908     {
909         \msg_warning:nnn {tag}{struct-Ref-unknown}{Label~'#1'}
910     }
911 }
912 \cs_new_protected:Npn \__tag_struct_Ref_dest:nN #1 #2 %#1 a dest name
913 {
914     \prop_get:NnNTF \g__tag_struct_dest_num_prop {#1} \l__tag_tmpb_tl
915     {

```

```

916         \tl_put_right:Ne#2
917         {
918             \c_space_tl\tag_struct_object_ref:e{ \l__tag_tmpb_tl }
919         }
920     }
921     {
922         \msg_warning:nnn {tag}{struct-Ref-unknown}{Destination~'#1'}
923     }
924 }
925 \cs_new_protected:Npn \__tag_struct_Ref_dest_parent:nN #1 #2 %#1 a dest name
926 {
927     \prop_get:NnNTF \g__tag_struct_dest_num_prop {#1} \l__tag_tmpb_tl
928     { %do not overwrite \l__tag_tmpa_tl!
929         \prop_get:cnN
930         { g__tag_struct_\l__tag_tmpb_tl _prop }{parentnum}\l__tag_tmpb_tl
931         \tl_put_right:Ne#2
932         {
933             \c_space_tl\tag_struct_object_ref:e{ \l__tag_tmpb_tl }
934         }
935     }
936     {
937         \msg_warning:nnn {tag}{struct-Ref-unknown}{Destination~'#1'}
938     }
939 }
940 \cs_new_protected:Npn \__tag_struct_Ref_num:nN #1 #2 %#1 a structure number
941 {
942     \tl_put_right:Ne#2
943     {
944         \c_space_tl\tag_struct_object_ref:e{ #1 }
945     }
946 }
947

```

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

ref (struct key)

E (struct key)

```

948 \keys_define:nn { __tag / struct }
949 {
950     ref .code:n          = % ref property
951     {
952         \clist_map_inline:on {#1}
953         {
954             \tag_struct_gput:nne
955             {\int_use:N \c@g__tag_struct_abs_int}{ref_label}{ ##1 }
956         }
957     },
958     E .code:n           = % E property
959     {
960         \str_set_convert:Nnon
961         \l__tag_tmpa_str
962         { #1 }
963         { default }
964         { utf16/hex }
965         \__tag_struct_prop_gput:nne
966         { \int_use:N \c@g__tag_struct_abs_int }

```

```

967         { E }
968         { <\l__tag_tmpa_str> }
969     },
970 }

```

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

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

```

971 \int_new:N\g__tag_struct_AFobj_int

```

(End of definition for \g__tag_struct_AFobj_int.)

```

972 \cs_generate_variant:Nn \pdffile_embed_stream:nnN {neN}
973 \cs_new_protected:Npn \__tag_struct_add_inline_AF:nn #1 #2
974 % #1 content, #2 extension
975 {
976     \tl_if_empty:nF{#1}
977     {
978         \group_begin:
979         \int_gincr:N \g__tag_struct_AFobj_int
980         \pdffile_embed_stream:neN
981         {#1}
982         {tag-AFfile\int_use:N\g__tag_struct_AFobj_int.#2}
983         \l__tag_tmpa_tl
984         \__tag_struct_add_AF:ee
985         { \int_use:N \c@g__tag_struct_abs_int }
986         { \l__tag_tmpa_tl }
987         \__tag_struct_prop_gput:nne
988         { \int_use:N \c@g__tag_struct_abs_int }
989         { AF }
990         {
991             [
992                 \tl_use:c
993                 { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
994             ]
995         }
996     } \group_end:
997 }
998 }
999
1000 \cs_generate_variant:Nn \__tag_struct_add_inline_AF:nn {on}
1001 \cs_new_protected:Npn \__tag_struct_add_AF:nn #1 #2
1002 % #1 struct num #2 object reference
1003 {
1004     \tl_if_exist:cTF
1005     {
1006         g__tag_struct_#1_AF_tl
1007     }
1008     {

```

```

1009         \tl_gput_right:ce
1010         { g__tag_struct_#1_AF_tl }
1011         { \c_space_tl #2 }
1012     }
1013     {
1014         \tl_new:c
1015         { g__tag_struct_#1_AF_tl }
1016         \tl_gset:ce
1017         { g__tag_struct_#1_AF_tl }
1018         { #2 }
1019     }
1020 }
1021 \cs_generate_variant:Nn \__tag_struct_add_AF:nn {en,ee}
1022 \keys_define:nn { __tag / struct }
1023 {
1024     AF .code:n          = % AF property
1025     {
1026         \pdf_object_if_exist:eTF {#1}
1027         {
1028             \__tag_struct_add_AF:ee
1029             { \int_use:N \c@g__tag_struct_abs_int }{\pdf_object_ref:e {#1}}
1030             \__tag_struct_prop_gput:nne
1031             { \int_use:N \c@g__tag_struct_abs_int }
1032             { AF }
1033             {
1034                 [
1035                     \tl_use:c
1036                     { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
1037                 ]
1038             }
1039         }
1040         {
1041             % message?
1042         }
1043     },
1044     AFref .code:n       = % AF property
1045     {
1046         \tl_if_empty:eF {#1}
1047         {
1048             \__tag_struct_add_AF:ee { \int_use:N \c@g__tag_struct_abs_int }{#1}
1049             \__tag_struct_prop_gput:nne
1050             { \int_use:N \c@g__tag_struct_abs_int }
1051             { AF }
1052             {
1053                 [
1054                     \tl_use:c
1055                     { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
1056                 ]
1057             }
1058         }
1059     },
1060     ,AFinline .code:n =
1061     {
1062         \__tag_struct_add_inline_AF:nn {#1}{txt}

```

```

1063     }
1064     ,AFinline-o .code:n =
1065     {
1066         \__tag_struct_add_inline_AF:on {#1}{txt}
1067     }
1068     ,texsource .code:n =
1069     {
1070         \group_begin:
1071         \pdfdict_put:nnn { l_pdffile/Filespec } {Desc}{(TeX~source)}
1072         \pdfdict_put:nnn { l_pdffile/Filespec }{AFRelationship} { /Source }
1073         \__tag_struct_add_inline_AF:on {#1}{tex}
1074         \group_end:
1075     }
1076     ,mathml .code:n =
1077     {
1078         \group_begin:
1079         \pdfdict_put:nnn { l_pdffile/Filespec } {Desc}{(mathml~representation)}
1080         \pdfdict_put:nnn { l_pdffile/Filespec }{AFRelationship} { /Supplement }
1081         \pdfdict_put:nne { l_pdffile }{Subtype}
1082         { \pdf_name_from_unicode_e:n{application/mathml+xml} }
1083         \__tag_struct_add_inline_AF:on {#1}{xml}
1084         \group_end:
1085     }
1086 }

```

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!

```

1087 \keys_define:nn { __tag / setup }
1088 {
1089     root-AF .code:n =
1090     {
1091         \pdf_object_if_exist:nTF {#1}
1092         {
1093             \__tag_struct_add_AF:ee { 1 }{\pdf_object_ref:n {#1}}
1094             \__tag_struct_prop_gput:nne
1095             { 1 }
1096             { AF }
1097             {
1098                 [
1099                     \tl_use:c
1100                     { g__tag_struct_1_AF_tl }
1101                 ]
1102             }
1103         }
1104         {
1105             }
1106         }
1107     },
1108 }

```

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

```

1109 \keys_define:nn { __tag / setup }
1110 {

```

```

1111 root-supplemental-file .code:n =
1112 {
1113   \group_begin:
1114   \pdfdict_put:nnn {l_pdffile/Filespec} {AFRelationship}{/Supplement}
1115   \int_gincr:N \g__tag_unique_cnt_int
1116   \pdffile_embed_file:eee
1117   {#1}
1118   {#1}
1119   {__tag_latex_css_\int_use:N\g__tag_unique_cnt_int}
1120   \keys_set:nn
1121   {__tag / setup}
1122   {root-AF={__tag_latex_css_\int_use:N\g__tag_unique_cnt_int}}
1123   \group_end:
1124 }
1125 }

```

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

```

1126 \keys_define:nn { __tag / setup }
1127 {
1128   catalog-supplemental-file .code:n =
1129   {
1130     \group_begin:
1131     \pdfdict_put:nnn {l_pdffile/Filespec} {AFRelationship}{/Supplement}
1132     \int_gincr:N \g__tag_unique_cnt_int
1133     \pdffile_embed_file:eee
1134     {#1}
1135     {#1}
1136     {__tag_latex_css_\int_use:N\g__tag_unique_cnt_int}
1137     \pdfmanagement_add:nne
1138     {Catalog}
1139     {AF}
1140     {\pdf_object_ref:e{__tag_latex_css_\int_use:N\g__tag_unique_cnt_int }}
1141     \group_end:
1142   }
1143 }

```

6 User commands

We allow to set a language by default

`\l__tag_struct_lang_tl`

```

1144 \tl_new:N \l__tag_struct_lang_tl
1145 \</package>

```

(End of definition for `\l__tag_struct_lang_tl`.)

`\tag_struct_begin:n`

`\tag_struct_end:`

```

1146 <base>\cs_new_protected:Npn \tag_struct_begin:n #1 {\int_gincr:N \c@g__tag_struct_abs_int}
1147 <base>\cs_new_protected:Npn \tag_struct_end: {}
1148 <base>\cs_new_protected:Npn \tag_struct_end:n {}
1149 <*package | debug>
1150 <package>\cs_set_protected:Npn \tag_struct_begin:n #1 {%#1 key-val

```

```

1151 <debug>\cs_set_protected:Npn \tag_struct_begin:n #1 %#1 key-val
1152 {
1153 <package>\__tag_check_if_active_struct:T
1154 <debug>\__tag_check_if_active_struct:TF
1155 {
1156   \group_begin:
1157   \int_gincr:N \c@g__tag_struct_abs_int
1158   \__tag_prop_new:c { g__tag_struct\_int_eval:n { \c@g__tag_struct_abs_int }_prop }
1159 <debug>   \prop_new:c { g__tag_struct_debug\_int_eval:n { \c@g__tag_struct_abs_int }_prop }
1160   \__tag_seq_new:c { g__tag_struct_kids\_int_eval:n { \c@g__tag_struct_abs_int }_seq }
1161 <debug>   \seq_new:c { g__tag_struct_debug_kids\_int_eval:n { \c@g__tag_struct_abs_int }_seq }
1162   \pdf_object_new_indexed:nn { __tag/struct }
1163   { \c@g__tag_struct_abs_int }
1164   \__tag_struct_prop_gput:nnn
1165   { \int_use:N \c@g__tag_struct_abs_int }
1166   { Type }
1167   { /StructElem }
1168   \tl_if_empty:NF \l__tag_struct_lang_tl
1169   {
1170     \__tag_struct_prop_gput:nne
1171     { \int_use:N \c@g__tag_struct_abs_int }
1172     { Lang }
1173     { (\l__tag_struct_lang_tl) }
1174   }
1175   \__tag_struct_prop_gput:nnn
1176   { \int_use:N \c@g__tag_struct_abs_int }
1177   { Type }
1178   { /StructElem }
1179
1180   \tl_set:Nn \l__tag_struct_stack_parent_tmpa_tl {-1}
1181   \keys_set:nn { __tag / struct } { #1 }
1182
1183   \__tag_struct_set_tag_info:eoo
1184   { \int_use:N \c@g__tag_struct_abs_int }
1185   { \g__tag_struct_tag_tl }
1186   { \g__tag_struct_tag_NS_tl }
1187   \__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.

```

1187   \int_compare:nNnT { \l__tag_struct_stack_parent_tmpa_tl } = { -1 }
1188   {
1189     \seq_get:NnF
1190     \g__tag_struct_stack_seq
1191     \l__tag_struct_stack_parent_tmpa_tl
1192     {
1193       \msg_error:nn { tag } { struct-faulty-nesting }
1194     }
1195   }
1196   \seq_gpush:NV \g__tag_struct_stack_seq \c@g__tag_struct_abs_int
1197   \__tag_role_get:ooNN
1198   { \g__tag_struct_tag_tl }
1199   { \g__tag_struct_tag_NS_tl }
1200   \l__tag_struct_roletag_tl
1201   \l__tag_struct_roletag_NS_tl

```

We push the role tag on the stack:

```

1202      \seq_gpush:Ne \g__tag_struct_tag_stack_seq
1203      {{\g__tag_struct_tag_t1}{\l__tag_struct_roletag_t1}}
1204      \seq_gpush:Ne \g__tag_struct_roletag_stack_seq
1205      {\l__tag_struct_roletag_t1}
1206      \tl_gset:NV \g__tag_struct_stack_current_t1 \c@g__tag_struct_abs_int
1207      \__tag_struct_set_attribute:
1208      %\seq_show:N \g__tag_struct_stack_seq

```

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

```

1209      \__tag_struct_prop_gput:nne
1210      { \int_use:N \c@g__tag_struct_abs_int }
1211      { rolemap }
1212      {
1213        {\l__tag_struct_roletag_t1}{\l__tag_struct_roletag_NS_t1}
1214      }

```

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.

```

1215      \str_case:onTF { \l__tag_struct_roletag_t1 }
1216      {
1217        {Part} {}
1218        {Div} {}
1219        {NonStruct} {}
1220      }
1221      {
1222        \bool_if:NTF \l__tag_struct_elem_stash_bool
1223        {
1224          \__tag_struct_prop_gput:nne
1225          { \int_use:N \c@g__tag_struct_abs_int }
1226          { parentrole }
1227          {
1228            {\l__tag_struct_parenttag_t1}{\l__tag_struct_parenttag_NS_t1}
1229          }
1230        }
1231        {
1232          \prop_get:cnNT
1233          { g__tag_struct_ \l__tag_struct_stack_parent_tmpa_t1 _prop }
1234          { parentrole }
1235          \l__tag_get_tmpc_t1
1236          {
1237            \__tag_struct_prop_gput:nno
1238            { \int_use:N \c@g__tag_struct_abs_int }
1239            { parentrole }
1240            {
1241              \l__tag_get_tmpc_t1
1242            }
1243          }
1244        }

```

```

1245     }
1246     {
1247         \__tag_struct_prop_gput:nne
1248         { \int_use:N \c@g__tag_struct_abs_int }
1249         { parentrole }
1250         {
1251             {\l__tag_struct_roletag_tl}{\l__tag_struct_roletag_NS_tl}
1252         }
1253     }
1254     \bool_if:NF
1255     \l__tag_struct_elem_stash_bool
1256     {

```

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.

```

1257         \socket_use:nn{tag/check/parent-child}
1258         {
1259             \__tag_struct_check_parent_child:oo
1260             { \l__tag_struct_stack_parent_tmpa_tl }
1261             { \int_use:N \c@g__tag_struct_abs_int }
1262         }

```

Set the Parent structure number.

```

1263         \__tag_struct_prop_gput:nne
1264         { \int_use:N \c@g__tag_struct_abs_int }
1265         { parentnum }
1266         {
1267             \l__tag_struct_stack_parent_tmpa_tl
1268         }
1269
1270         %record this structure as kid:
1271         %\tl_show:N \g__tag_struct_stack_current_tl
1272         %\tl_show:N \l__tag_struct_stack_parent_tmpa_tl
1273         \use:c { __tag_struct_kid_struct_gput_ \l__tag_struct_addkid_tl :ee }
1274         { \l__tag_struct_stack_parent_tmpa_tl }
1275         { \g__tag_struct_stack_current_tl }
1276         %\prop_show:c { g__tag_struct_\g__tag_struct_stack_current_tl _prop }
1277         %\seq_show:c {g__tag_struct_kids_\l__tag_struct_stack_parent_tmpa_tl _seq}

```

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.

```

1278 <debug>         \prop_gset_eq:cc
1279 <debug>         { g__tag_struct_debug_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
1280 <debug>         { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
1281 <debug>         \prop_gput:cne
1282 <debug>         { g__tag_struct_debug_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
1283 <debug>         { parentnum }
1284 <debug>         {
1285 <debug>             \bool_if:NTF \l__tag_struct_elem_stash_bool
1286 <debug>             {no~parent::~stashed}
1287 <debug>             {
1288 <debug>                 \l__tag_struct_stack_parent_tmpa_tl\c_space_tl =~

```

```

1289 <debug>                                \prop_item:cn{ g__tag_struct_\l__tag_struct_stack_parent_tmpa_tl _p
1290 <debug>                                }
1291 <debug>                                }
1292 <debug>                                \prop_gput:cne
1293 <debug>                                { g__tag_struct_debug\int_eval:n {\c@g__tag_struct_abs_int}_prop }
1294 <debug>                                { NS }
1295 <debug>                                { \g__tag_struct_tag_NS_tl }

1296                                %\prop_show:c { g__tag_struct_\g__tag_struct_stack_current_tl _prop }
1297                                %\seq_show:c {g__tag_struct_kids_\l__tag_struct_stack_parent_tmpa_tl _seq}
1298 <debug> \__tag_debug_struct_begin_insert:n { #1 }
1299 \group_end:
1300 }
1301 <debug>{ \__tag_debug_struct_begin_ignore:n { #1 }}
1302 }
1303 <package>\cs_set_protected:Nn \tag_struct_end:
1304 <debug>\cs_set_protected:Nn \tag_struct_end:
1305 { %take the current structure num from the stack:
1306   %the objects are written later, lua mode hasn't all needed info yet
1307   %\seq_show:N \g__tag_struct_stack_seq
1308 <package>\__tag_check_if_active_struct:T
1309 <debug>\__tag_check_if_active_struct:TF
1310 {
1311   \seq_gpop:NN \g__tag_struct_roletag_stack_seq \l__tag_tmpa_tl
1312   \seq_gpop:NN \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
1313   \seq_gpop:NNTF \g__tag_struct_stack_seq \l__tag_tmpa_tl
1314   {
1315     \__tag_check_info_closing_struct:o { \g__tag_struct_stack_current_tl }
1316   }
1317   { \__tag_check_no_open_struct: }
1318   % get the previous one, shouldn't be empty as the root should be there
1319   \seq_get:NNTF \g__tag_struct_stack_seq \l__tag_tmpa_tl
1320   {
1321     \tl_gset:No \g__tag_struct_stack_current_tl { \l__tag_tmpa_tl }
1322     \__tag_struct_set_attribute:
1323   }
1324   {
1325     \__tag_check_no_open_struct:
1326   }
1327   \seq_get:NNT \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
1328   {
1329     \tl_gset:Ne \g__tag_struct_tag_tl
1330     { \exp_last_unbraced:No\use_i:nn { \l__tag_tmpa_tl } }
1331     \prop_get:NoNT\g__tag_role_tags_NS_prop { \g__tag_struct_tag_tl} \l__tag_tmpa_tl
1332     {
1333       \tl_gset:Ne \g__tag_struct_tag_NS_tl { \l__tag_tmpa_tl }
1334     }
1335   }
1336 <debug>\__tag_debug_struct_end_insert:
1337 }
1338 <debug>{ \__tag_debug_struct_end_ignore:}
1339 }
1340
1341 \cs_set_protected:Npn \tag_struct_end:n #1
1342 {

```

```

1343 <debug>    \__tag_check_if_active_struct:T{\__tag_debug_struct_end_check:n{#1}}
1344    \tag_struct_end:
1345  }
1346 </package | debug>

```

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

\tag_struct_use:n This command allows to use a stashed structure in another place. TODO: decide how it should be guarded. Probably by the struct-check.

```

1347 <base>\cs_new_protected:Npn \tag_struct_use:n #1 {}
1348 <*package | debug>
1349 \cs_set_protected:Npn \tag_struct_use:n #1 %#1 is the label
1350 {
1351   \__tag_check_if_active_struct:T
1352   {
1353     \prop_if_exist:cTF
1354     { g__tag_struct_\property_ref:enn{tagpdfstruct-#1}{tagstruct}{unknown}_prop } %
1355     {
1356       \__tag_check_struct_used:n {#1}
1357       \tl_set:Nx \l__tag_get_child_tmpa_tl
1358       { \property_ref:enn{tagpdfstruct-#1}{tagstruct}{1} }

```

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

```

1359   \__tag_struct_kid_struct_gput_right:ee
1360   { \g__tag_struct_stack_current_tl }
1361   { \l__tag_get_child_tmpa_tl }

```

add the current structure to the labeled one as parents

```

1362   \__tag_prop_gput:cne
1363   { g__tag_struct_\l__tag_get_child_tmpa_tl _prop }
1364   { parentnum }
1365   {
1366     \g__tag_struct_stack_current_tl
1367   }

```

debug code

```

1368 <debug>    \prop_gput:cne
1369 <debug>    { g__tag_struct_debug_\l__tag_get_child_tmpa_tl _prop }
1370 <debug>    { parentnum }
1371 <debug>    {
1372 <debug>        \g__tag_struct_stack_current_tl\c_space_tl=~
1373 <debug>        \g__tag_struct_tag_tl
1374 <debug>    }

```

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

```

1375   \socket_use:nn{tag/check/parent-child}
1376   {
1377     \__tag_struct_use_check_parent_child:oo
1378     { \g__tag_struct_stack_current_tl }

```

```

1379         { \l__tag_get_child_tmpa_tl }
1380     }
1381 }
1382 {
1383     \msg_warning:nnn{ tag }{struct-label-unknown}{#1}
1384 }
1385 }
1386 }
1387 </package|debug>

```

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

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

```

1388 <base>\cs_new_protected:Npn \tag_struct_use_num:n #1 {}
1389 <*package|debug>
1390 \cs_set_protected:Npn \tag_struct_use_num:n #1 {%#1 is structure number
1391 {
1392     \__tag_check_if_active_struct:T
1393     {
1394         \prop_if_exist:cTF
1395         { g__tag_struct_#1_prop } %
1396         {
1397             \prop_get:cnNT
1398             {g__tag_struct_#1_prop}
1399             {parentnum}
1400             \l__tag_tmpa_tl
1401             {
1402                 \msg_warning:nnn { tag } {struct-used-twice} {#1}
1403             }

```

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

```

1404         \__tag_struct_kid_struct_gput_right:ee
1405         { \g__tag_struct_stack_current_tl }
1406         { #1 }

```

add the current structure to #1 as parent

```

1407         \__tag_struct_prop_gput:nne
1408         { #1 }
1409         { parentnum }
1410         {
1411             \g__tag_struct_stack_current_tl
1412         }
1413 <debug>         \prop_gput:cne
1414 <debug>         { g__tag_struct_debug_#1_prop }
1415 <debug>         { parentnum }
1416 <debug>         {
1417 <debug>         \g__tag_struct_stack_current_tl\c_space_tl=~
1418 <debug>         \g__tag_struct_tag_tl
1419 <debug>         }

```

check if the tag is allowed as child.

```

1420         \socket_use:nn{tag/check/parent-child}

```

```

1421         {
1422             \_tag_struct_use_check_parent_child:oo
1423             {\g\_tag_struct_stack_current_tl}
1424             {#1}
1425         }
1426     }
1427     {
1428         \msg_warning:nnn{ tag }{struct-label-unknown}{#1}
1429     }
1430 }
1431 }
1432 \cs_generate_variant:Nn\tag_struct_use_num:n {o}
1433 \</package\ debug>

```

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

`\tag_struct_object_ref:n` This is a command that allows to reference a structure. The argument is the number which can be get for the current structure with `\tag_get:n{struct_num}` or `\tag_get:nN{struct_num}` TODO check if it should be in base too.

```

1434 \<package>
1435 \cs_new:Npn \tag_struct_object_ref:n #1
1436 {
1437     \pdf_object_ref_indexed:nn {\_tag/struct}{ #1 }
1438 }
1439 \cs_generate_variant:Nn \tag_struct_object_ref:n {e}
1440 \</package>

```

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

`\tag_struct_gput:nnn` This is a command that allows to update the data of a structure. This often can't 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 **tag** updates the tag of a structure (the **S** entry).
- The keyword **C** *replaces* the attribute-class array. The argument should be a comma list of PDF names **/TH-row,/justify**. In case that the list should be extended it is up to the user to get the original list first and manipulate it.
- 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_dest_parent** expects a destination name set with `\MakeLinkTarget` too. It then will refer to the parent of structure in which this `\MakeLinkTarget` was used—this is useful if the target command is, e.g., in a list and one wants to reference the surrounding list item structure.
- 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.

```

1441 <base>\cs_new_protected:Npn \tag_struct_gput:nnn #1 #2 #3{
1442 <*package>
1443 \cs_set_protected:Npn \tag_struct_gput:nnn #1 #2 #3
1444 {
1445   \cs_if_exist_use:cF {__tag_struct_gput_data_#2:nn}
1446   { %warning??
1447     \use_none:nn
1448   }
1449   {#1}{#3}
1450 }
1451 \cs_generate_variant:Nn \tag_struct_gput:nnn {ene,nne}

```

This maps over the tag stack (which contains the tag and the rolemap in two brace groups) and the number stack. The argument is a function that can handle this two arguments.

\tag_struct_map_function:N

```

1452 \cs_new_protected:Npn \tag_struct_map_function:N #1
1453 {
1454   \seq_map_pairwise_function:NNN
1455   \g__tag_struct_tag_stack_seq
1456   \g__tag_struct_stack_seq
1457   #1
1458 }
1459 </package>

```

(End of definition for \tag_struct_gput:nnn and \tag_struct_map_function:N. These functions are documented on page 114.)

__tag_struct_gput_data_ref_aux:nnn

```

1460 <*package>
1461 \cs_new_protected:Npn \__tag_struct_gput_data_ref_aux:nnn #1 #2 #3
1462   % #1 receiving struct num, #2 key word #3 value
1463   {
1464     \prop_get:cnNTF
1465     { g__tag_struct_#1_prop }
1466     {Ref}
1467     \l__tag_get_tmpc_tl
1468     {
1469       \tl_put_right:No \l__tag_get_tmpc_tl
1470       {\cs:w __tag_struct_Ref_#2:nN \cs_end: {#3},}
1471     }
1472     {
1473       \tl_set:No \l__tag_get_tmpc_tl
1474       {\cs:w __tag_struct_Ref_#2:nN \cs_end: {#3},}
1475     }
1476     \__tag_struct_prop_gput:nno
1477     { #1 }
1478     { Ref }
1479     { \l__tag_get_tmpc_tl }
1480   }

```

```

1481 \cs_new_protected:Npn \__tag_struct_gput_data_ref:nn #1 #2
1482 {
1483   \__tag_struct_gput_data_ref_aux:nnn {#1}{obj}{#2}
1484 }
1485 \cs_new_protected:Npn \__tag_struct_gput_data_ref_label:nn #1 #2
1486 {
1487   \__tag_struct_gput_data_ref_aux:nnn {#1}{label}{#2}
1488 }
1489 \cs_new_protected:Npn \__tag_struct_gput_data_ref_dest:nn #1 #2
1490 {
1491   \__tag_struct_gput_data_ref_aux:nnn {#1}{dest}{#2}
1492 }
1493 \cs_new_protected:Npn \__tag_struct_gput_data_ref_dest_parent:nn #1 #2
1494 {
1495   \__tag_struct_gput_data_ref_aux:nnn {#1}{dest_parent}{#2}
1496 }
1497 \cs_new_protected:Npn \__tag_struct_gput_data_ref_num:nn #1 #2
1498 {
1499   \__tag_struct_gput_data_ref_aux:nnn {#1}{num}{#2}
1500 }
1501
1502 \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

```

1503 \cs_new_protected:Npn \__tag_struct_gput_data_attribute:nn #1 #2
1504 {
1505   \pdf_object_unnamed_write:nn {dict} {#2}
1506   \prop_get:cnNTF { g__tag_struct_#1_prop }{A} \l__tag_tmpa_tl
1507   {
1508     \tl_remove_once:Nn\l__tag_tmpa_tl{[]}
1509     \tl_remove_once:Nn\l__tag_tmpa_tl{]}
1510     \__tag_prop_gput:cne { g__tag_struct_#1_prop }
1511     { A }
1512     {
1513       [ \l__tag_tmpa_tl \c_space_tl \pdf_object_ref_last: ]
1514     }
1515   }
1516   {
1517     \__tag_prop_gput:cne { g__tag_struct_#1_prop }
1518     { A }
1519     { \pdf_object_ref_last: }
1520   }
1521 }

```

(End of definition for __tag_struct_gput_data_attribute:nn.)

__tag_struct_gput_data_tag:nn

This is used by \tag_struct_gput:nnn to update the tag of a structure.

```

1522 \cs_new_protected:Npn \__tag_struct_gput_data_tag:nn #1 #2
1523 {
1524   \__tag_struct_prop_gput:nne{#1}{S}{#2}
1525 }

```

(End of definition for __tag_struct_gput_data_tag:nn.)

`\__tag_struct_gput_data_C:nn` This is used by `\tag_struct_gput:nnn` to update the C entry of a structure.

```

1526 \cs_new_protected:Npn \__tag_struct_gput_data_C:nn #1 #2
1527 {
1528   \int_case:nnF {\clist_count:n{#2}}
1529   {
1530     {0}{ }
1531     {1}{\__tag_struct_prop_gput:nne{#1}{C}{#2}}
1532   }
1533   {
1534     \__tag_struct_prop_gput:nne{#1}{C}{[\clist_use:nn{#2}{ }]}
1535   }
1536 }

```

(End of definition for `\__tag_struct_gput_data_C:nn`.)

`\tag_struct_insert_annot:nn` This are the user command to insert annotations. They must be used together to get the numbers right. They use a counter to the `StructParent` and `\tag_struct_insert_annot:nn` increases the counter given back by `\tag_struct_parent_int:.`

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

```

1537 \cs_new_protected:Npn \tag_struct_insert_annot:nn #1 #2 % #1 should be an object reference
1538                                     % #2 struct parent num
1539 {
1540   \__tag_check_if_active_struct:T
1541   {
1542     \__tag_struct_insert_annot:nn {#1}{#2}
1543   }
1544 }
1545
1546 \cs_generate_variant:Nn \tag_struct_insert_annot:nn {xx,ee}
1547 \cs_new:Npn \tag_struct_parent_int: {\int_use:c { c@g__tag_parenttree_obj_int }}
1548
1549 </package>
1550

```

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

7 Attributes and attribute classes

```

1551 <*header>
1552 \ProvidesExplPackage {tagpdf-attr-code} {2026-08-21} {1.0e}
1553   {part of tagpdf - code related to attributes and attribute classes}
1554 </header>

```

7.1 Variables

`\g__tag_attr_entries_prop` `\g_@@_attr_entries_prop` will store attribute names and their dictionary content.
`\g__tag_attr_class_used_prop` `\g_@@_attr_class_used_prop` will hold the attributes which have been used as class name. `\l_@@_attr_value_tl` is used to build the attribute array or key. Every time an attribute is used for the first time, and object is created with its content, the name-object reference relation is stored in `\g_@@_attr_objref_prop`

```

1555 <*package>

```

```

1556 \prop_new:N \g__tag_attr_entries_prop
1557 \prop_new_linked:N \g__tag_attr_class_used_prop
1558 \tl_new:N \l__tag_attr_value_tl
1559 \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.

```

1560 \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. `\__tag_attr_new_entry:nn` `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)

```

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

1561 \cs_new_protected:Npn \__tag_attr_new_entry:nn #1 #2 %#1:name, #2: content
1562 {
1563   \prop_gput:Nen \g__tag_attr_entries_prop
1564     {\pdf_name_from_unicode_e:n{#1}}{#2}
1565 }
1566 \cs_new_protected:Npn \__tag_attr_new_used_entry:nn #1 #2 %#1:name, #2: content
1567 {
1568   \prop_gput:Nen \g__tag_attr_entries_prop
1569     {\pdf_name_from_unicode_e:n{#1}}{#2}
1570   \seq_gput_left:Ne\g__tag_attr_class_used_seq
1571     {\pdf_name_from_unicode_e:n{#1}}
1572 }
1573
1574 \cs_set_eq:NN \tag_attr_new:nn \__tag_attr_new_entry:nn
1575 \cs_generate_variant:Nn \__tag_attr_new_entry:nn {ee}
1576 \cs_generate_variant:Nn \tag_attr_new:nn {ee}
1577
1578 \keys_define:nn { __tag / setup }
1579 {
1580   role/new-attribute .code:n =
1581   {
1582     \__tag_attr_new_entry:nn #1
1583   },
1584   role/new-attribute* .code:n =
1585   {
1586     \__tag_attr_new_used_entry:nn #1
1587   }

```

deprecated name

```
1588     ,newattribute .code:n =
1589     {
1590         \__tag_attr_new_entry:nn #1
1591     },
1592 }
```

(End of definition for \tag_attr_new_entry:nn and others. These functions are documented on page ??.)

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

```
1593 \keys_define:nn { __tag / struct }
1594 {
1595     attribute-class .code:n =
1596     {
1597         \clist_set:Nc \l__tag_tmpa_clist { #1 }
1598         \seq_set_from_clist:NN \l__tag_tmpb_seq \l__tag_tmpa_clist
```

we convert the names into pdf names with slash

```
1599         \seq_set_map_e:NNn \l__tag_tmpa_seq \l__tag_tmpb_seq
1600         {
1601             \pdf_name_from_unicode_e:n {##1}
1602         }
1603         \seq_map_inline:Nn \l__tag_tmpa_seq
1604         {
1605             \prop_get:NnNF \g__tag_attr_entries_prop {##1}\l__tag_tmpa_tl
1606             {
1607                 \msg_error:nnn { tag } { attr-unknown } { ##1 }
1608             }
1609             \prop_gput:Nnn\g__tag_attr_class_used_prop { ##1} {}
1610         }
1611         \tl_set:Nc \l__tag_tmpa_tl
1612         {
1613             \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}
1614             \seq_use:Nn \l__tag_tmpa_seq { \c_space_tl }
1615             \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}
1616         }
1617         \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 0 }
1618         {
1619             \__tag_struct_prop_gput:nne
1620             { \int_use:N \c@g__tag_struct_abs_int }
1621             { C }
1622             { \l__tag_tmpa_tl }
1623             %\prop_show:c { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
1624         }
1625     }
1626 }
```

attribute (*struct key*)

```
1627 \keys_define:nn { __tag / struct }
1628 {
1629     attribute .code:n = % A property (attribute, value currently a dictionary)
1630     {
```

```

1631 \clist_set:Ne \l__tag_tmpa_clist { #1 }
1632 \clist_if_empty:NF \l__tag_tmpa_clist
1633 {
1634 \seq_set_from_clist:NN \l__tag_tmpb_seq \l__tag_tmpa_clist

we convert the names into pdf names with slash
1635 \seq_set_map_e:NNn \l__tag_tmpa_seq \l__tag_tmpb_seq
1636 {
1637 \pdf_name_from_unicode_e:n {##1}
1638 }
1639 \tl_set:Ne \l__tag_attr_value_tl
1640 {
1641 \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}%
1642 }
1643 \seq_map_inline:Nn \l__tag_tmpa_seq
1644 {
1645 \prop_get:NnNF \g__tag_attr_entries_prop {##1}\l__tag_tmp_unused_tl
1646 {
1647 \msg_error:nnn { tag } { attr-unknown } { ##1 }
1648 }
1649 \prop_get:NnNF \g__tag_attr_objref_prop {##1}\l__tag_tmpa_tl
1650 {%\prop_show:N \g__tag_attr_entries_prop
1651 \pdf_object_unnamed_write:ne
1652 { dict }
1653 {
1654 \prop_item:Nn\g__tag_attr_entries_prop {##1}
1655 }
1656 \prop_gput:Nne \g__tag_attr_objref_prop {##1} {\pdf_object_ref_last:}
1657 }
1658 \tl_put_right:Ne \l__tag_attr_value_tl
1659 {
1660 \c_space_tl
1661 \prop_item:Nn \g__tag_attr_objref_prop {##1}
1662 }
1663 % \tl_show:N \l__tag_attr_value_tl
1664 }
1665 \tl_put_right:Ne \l__tag_attr_value_tl
1666 { %[
1667 \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}%
1668 }
1669 % \tl_show:N \l__tag_attr_value_tl
1670 \__tag_struct_prop_gput:nne
1671 { \int_use:N \c@g__tag_struct_abs_int }
1672 { A }
1673 { \l__tag_attr_value_tl }
1674 }
1675 },
1676 }
1677 </package>

```

Ulrike Fischer
Version 1.0e, released 2026-08-21

Part IX

The tagpdf driver for luatex

```
1 <@@=tag>
2 <*luatex>
3 \ProvidesExplFile {tagpdf-luatex.def} {2026-08-21} {1.0e}
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.sprint(\int_use:N\c_document_cctab,ltx.__tag.tables['#1']['#2']) }
50 }
51
52 \cs_set:Npn \__tag_prop_item:cn #1 #2
53 {
54   \lua_now:e { tex.sprint(\int_use:N\c_document_cctab,ltx.__tag.tables['#1']['#2']) }
55 }
56
57 %for debugging commands that show both the seq/prop and the lua tables
58 \cs_set_protected:Npn \__tag_seq_show:N #1
59 {
60   \seq_show:N #1
61   \lua_now:e { ltx.__tag.trace.log ("lua~sequence~array~\cs_to_str:N#1",1) }
62   \lua_now:e { ltx.__tag.trace.show_seq (ltx.__tag.tables['\cs_to_str:N#1']) }
63 }
64
65 \cs_set_protected:Npn \__tag_prop_show:N #1
66 {
67   \prop_show:N #1
68   \lua_now:e {ltx.__tag.trace.log ("lua~property~table~\cs_to_str:N#1",1) }
69   \lua_now:e {ltx.__tag.trace.show_prop (ltx.__tag.tables['\cs_to_str:N#1']) }
70 }

```

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

```

71 \</luatex>

```

The module declaration

```

72 \<lua>
73 -- tagpdf.lua
74 -- Ulrike Fischer
75
76 local ProvidesLuaModule = {
77   name      = "tagpdf",
78   version   = "1.0e",      --TAGVERSION
79   date      = "2026-08-21", --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 EM
125 ltx.__tag.func.fill_parent_tree_line (page): outputs the entries of the parenttree for this
126 ltx.__tag.func.output_parenttree(): outputs the content of the parenttree
127 ltx.__tag.func.pdf_object_ref(name,index): outputs the object reference for the object name
128 ltx.__tag.func.markspaceon(), ltx.__tag.func.markspaceoff(): (de)activates the marking of po
129 ltx.__tag.func.linkbin() returns the number of OBJR stored in the linkbin structure
130 ltx.__tag.trace.show_mc_data (num,loglevel): shows ltx.__tag.mc[num] is the current log leve
131 ltx.__tag.trace.show_all_mc_data (max,loglevel): shows a maximum about mc's if the current l

```

```

132 ltx.__tag.trace.show_seq: shows a sequence (array)
133 ltx.__tag.trace.show_struct_data (num): shows data of structure num
134 ltx.__tag.trace.show_prop: shows a prop
135 ltx.__tag.trace.log
136 ltx.__tag.trace.showspace : boolean
137
138 ltx.tag.get_structnum: number, shows the current structure number
139 ltx.tag.get_structnum_next: number, shows the next structure number
140 --]]
141

```

This set-ups the main attribute registers. The `mc_type` attribute stores the type (P, Span etc) encoded as a num, The `mc_cnt` attribute stores the absolute number and allows so to see if a node belongs to the same mc-chunk. The `structnum` attribute stores the structure number. The `interwordspace` attr is set by the function `@@_mark_spaces`, and marks the place where spaces should be inserted. The `interwordfont` attr is set by the function `@@_mark_spaces` too and stores the font, so that we can decide which font to use for the real space char. The `interwordspaceOff` attr allows to locally suppress the insertion of real space chars, e.g. when they are inserted by other means (e.g. with `\char`). The `softhyphenattribute` allows to decide how to handle a soft hyphen: if unset it is surrounded by an Artifact mc, if set to 1 the char is changed. Other values will perhaps be used later.

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

2 User commands to access data

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

\

```

205 local __tag_get_struct_num =
206 function()
207     local a = token.get_macro("g__tag_struct_stack_current_tl")
208     return a

```

```

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

```

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

3 Logging functions

`__tag_log` This rather simple log function takes as argument a message (string) and a number and will output the message to the log/terminal if the current loglevel is greater or equal than num.

```

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

```

(End of definition for `__tag_log` and `ltx.__tag.trace.log`.)

`ltx.__tag.trace.show_seq` This shows the content of a seq as stored in the tables table. It is used by the `\@@_seq_show:N` function. It is not used in user commands, only for debugging, and so requires log level >0.

```

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

```

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

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

```

243 local __tag_pairs_prop =

```

```

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

```

(End of definition for __tag_pairs_prop and ltx.__tag.trace.show_prop.)

ltx.__tag.trace.show_mc_data This shows some data for a mc given by num. If something is shown depends on the log level. The function is used by the following function and then in \ShowTagging

```

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

```

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

ltx.__tag.trace.show_all_mc_data This shows data for the mc's between min and max (numbers). It is used by the \ShowTagging function.

```

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

```

(End of definition for `ltx.__tag.trace.show_all_mc_data`.)

`ltx.__tag.trace.show_struct_data` This function shows some struct data. Unused but kept for debugging.

```

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

```

(End of definition for `ltx.__tag.trace.show_struct_data`.)

4 Helper functions

4.1 Retrieve data functions

`__tag_get_mc_cnt_type_tag` This takes a node as argument and returns the mc-cnt, the mc-type and and the tag (calculated from the mc-cnt).

```

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

```

(End of definition for `__tag_get_mc_cnt_type_tag`.)

`__tag_get_mathsubtype` This function allows to detect if we are at the begin or the end of math. It takes as argument a mathnode.

```

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

```

(End of definition for `__tag_get_mathsubtype`.)

`ltx.__tag.tables.role_tag_attribute` The first is a table with key a tag and value a number (the attribute) The second is an array with the attribute value as key.

```

312 ltx.__tag.tables.role_tag_attribute = {}
313 ltx.__tag.tables.role_attribute_tag = {}

```

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

`ltx.__tag.func.alloctag`

```

314 local __tag_alloctag =
315   function (tag)
316     if not ltx.__tag.tables.role_tag_attribute[tag] then

```

```

317     table.insert(ltx.__tag.tables.role_attribute_tag,tag)
318     ltx.__tag.tables.role_tag_attribute[tag]=#ltx.__tag.tables.role_attribute_tag
319     __tag_log ("Add "..tag.." "..ltx.__tag.tables.role_tag_attribute[tag],3)
320 end
321 end
322 ltx.__tag.func.alloctag = __tag_alloctag
(End of definition for ltx.__tag.func.alloctag.)

```

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

```

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

```

```

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

```

(End of definition for `__tag_get_num_from`, `ltx.__tag.func.get_num_from`, and `ltx.__tag.func.output_num_from`.)

These functions are the opposites to the previous function: they take as argument a number (the attribute value) and return the string `tag`. The first function outputs the string for lua, while the `output` function outputs to tex.

```

__tag_get_tag_from
ltx.__tag.func.get_tag_from
ltx.__tag.func.output_tag_from

```

```

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

```

(End of definition for `__tag_get_tag_from`, `ltx.__tag.func.get_tag_from`, and `ltx.__tag.func.output_tag_from`.)

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

```

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

```

(End of definition for `ltx.__tag.func.store_mc_data`.)

ltx.__tag.func.store_mc_label This function stores the `label=num` relationship in the `labels` subtable. TODO: this is probably unused and can go.

```

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

```

(End of definition for `ltx.__tag.func.store_mc_label`.)

ltx.__tag.func.store_mc_kid This function is used in the traversing code. It stores a sub-chunk of a mc `mcnum` into the `kids` table.

```

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

```

(End of definition for `ltx.__tag.func.store_mc_kid`.)

ltx.__tag.func.mc_num_of_kids This function returns the number of kids a mc `mcnum` has. We need to account for the case that a mc can have no kids.

```

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

```

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

4.2 Functions to insert the pdf literals

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

```

380 local __tag_backend_create_emc_node
381 if tex.outputmode == 0 then
382   if token.get_macro("c_sys_backend_str") == "dvipdfmx" then
383     function __tag_backend_create_emc_node ()
384       local emcnode = nodenew("whatsit","special")
385       emcnode.data = "pdf:code EMC"
386       return emcnode
387     end

```

```

388 else -- assume a dvips variant
389   function __tag_backend_create_emc_node ()
390     local emcnode = nodenew("whatsit","special")
391     emcnode.data = "ps:SDict begin mark /EMC pdfmark end"
392     return emcnode
393   end
394 end
395 else -- pdf mode
396   function __tag_backend_create_emc_node ()
397     local emcnode = nodenew("whatsit","pdf_literal")
398     emcnode.data = "EMC"
399     emcnode.mode=1
400     return emcnode
401   end
402 end
403
404 local function __tag_insert_emc_node (head,current)
405   local emcnode= __tag_backend_create_emc_node()
406   head = node.insert_before(head,current,emcnode)
407   return head
408 end

```

(End of definition for __tag_backend_create_emc_node and __tag_insert_emc_node.)

__tag_backend_create_bmc_node
__tag_insert_bmc_node

This inserts a simple bmc node

```

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

```

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

```

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

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

```

(End of definition for `__tag_backend_create_bdc_node` and `__tag_insert_bdc_node`.)

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

```

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

```

(End of definition for `__tag_pdf_object_ref`.)

5 Function for the real space chars

`--tag_show_spacemark` A debugging function, it is used to inserts red color markers in the places where space chars can go, it can have side effects so not always reliable, but ok.

```

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

```

(End of definition for `--tag_show_spacemark`.)

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

```

ltx.__tag.func.fakespace
493 local function __tag_fakespace()
494   tex.setattribute(iwspaceattributeid,1)
495   tex.setattribute(iwfontattributeid,font.current())
496 end
497 ltx.__tag.func.fakespace = __tag_fakespace

```

(End of definition for `--tag_fakespace` and `ltx.__tag.func.fakespace`.)

`--tag_mark_spaces` a function to mark up places where real space chars should be inserted. It only sets attributes, these are then be used in a later traversing which inserts the actual spaces. When space handling is activated this function is inserted in some callbacks.

```

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

```

```

518 elseif glyph.next and (glyph.next.id==KERN) and not inside_math then
519   local kern = glyph.next
520   if kern.next and (kern.next.id== GLUE) and (kern.next.width >0)
521     -- the attribute is also set on the kern in case the kern+glue is
522     -- discarded at a line break tagging issue #1102
523     -- TODO iterate back through all discardable nodes.
524   then
525     nodesetattribute(kern,iwspaceattributeid,1)
526     nodesetattribute(kern,iwfontattributeid,glyph.font)
527     nodesetattribute(kern.next,iwspaceattributeid,1)
528     nodesetattribute(kern.next,iwfontattributeid,glyph.font)
529   end
530 end
531 -- look also back
532 if glyph.prev and (glyph.prev.id == GLUE)
533   and not inside_math
534   and (glyph.prev.width >0)
535   and not nodehasattribute(glyph.prev,iwspaceattributeid)
536 then
537   nodesetattribute(glyph.prev,iwspaceattributeid,1)
538   nodesetattribute(glyph.prev,iwfontattributeid,glyph.font)
539 -- for debugging
540 if ltx.__tag.trace.showspace then
541   __tag_show_spacemark (head,glyph)
542 end
543 end
544 elseif id == PENALTY then
545   local glyph = n
546   -- __tag_log ("PENALTY ".. n.subtype.."VALUE"..n.penalty,3)
547   if glyph.next and (glyph.next.id == GLUE)
548     and not inside_math and (glyph.next.width >0) and n.subtype==0
549   then
550     nodesetattribute(glyph.next,iwspaceattributeid,1)
551     -- changed 2024-01-18, issue #72
552     nodesetattribute(glyph.next,iwfontattributeid,default_currfontid)
553 -- for debugging
554 if ltx.__tag.trace.showspace then
555   __tag_show_spacemark (head,glyph)
556 end
557 end
558 elseif id == MATH then
559   inside_math = (n.subtype == 0)
560 end
561 end
562 return head
563 end

```

(End of definition for __tag_mark_spaces.)

```

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

```

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

```

564 local function __tag_activate_mark_space ()
565   if not luatexbase.in_callback ("pre_linebreak_filter","markspaces") then
566     luatexbase.add_to_callback("pre_linebreak_filter",__tag_mark_spaces,"markspaces")

```

```

567   luatexbase.add_to_callback("hpack_filter",__tag_mark_spaces,"markspaces")
568   end
569 end
570
571 ltx.__tag.func.markspaceon=__tag_activate_mark_space
572
573 local function __tag_deactivate_mark_space ()
574   if luatexbase.in_callback ("pre_linebreak_filter","markspaces") then
575     luatexbase.remove_from_callback("pre_linebreak_filter","markspaces")
576     luatexbase.remove_from_callback("hpack_filter","markspaces")
577   end
578 end
579
580 ltx.__tag.func.markspaceoff=__tag_deactivate_mark_space
(End of definition for __tag_activate_mark_space, ltx.__tag.func.markspaceon, and ltx.__tag.func.markspaceoff.)

```

We need two local variable to setup a default space char.

```

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

```

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

```

586 local function __tag_font_has_space (fontid)
587   t= fonts.hashes.identifiers[fontid]

```

test if t is nil, see <https://github.com/latex3/tagging-project/issues/1494>

```

588   if not t then return false end
589   if luaotfload.aux.slot_of_name(fontid,"space")
590     or t and t.characters and t.characters[32] and t.characters[32]["unicode"]==32
591   then
592     return true
593   else
594     return false
595   end
596 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.

```

597 local function __tag_space_chars_shipout (box)
598   local head = box.head
599   if head then
600     for n in node.traverse(head) do
601       local spaceattr = -1
602       if not nodehasattribute(n,iwspaceOffattributeid) then
603         spaceattr = nodegetattribute(n,iwspaceattributeid) or -1
604       end
605       if n.id == HLIST then -- enter the hlist
606         __tag_space_chars_shipout (n)
607       elseif n.id == VLIST then -- enter the vlist
608         __tag_space_chars_shipout (n)

```

```

609     elseif n.id == GLUE then
610         if ltx.__tag.trace.showspaces and spaceattr==1 then
611             __tag_show_spacemark (head,n,"0 1 0")
612         end
613         if spaceattr==1 then
614             local space
615             local space_char = node.copy(default_space_char)
616             local curfont = node.getattribute(n,iwfontattributeid)
617             __tag_log ("INFO SPACE-FUNCTION-FONT: ".. tostring(curfont),3)
618             if curfont and
619                 -- luaotfload.aux.slot_of_name(curfont,"space")
620                 __tag_font_has_space (curfont)
621             then
622                 space_char.font=curfont
623             end
624             head, space = node.insert_before(head, n, space_char) --
625             n.width = n.width - space.width
626             space.attr = n.attr
627         end
628     end
629 end
630 box.head = head
631 end
632 end
633
634 function ltx.__tag.func.space_chars_shipout (box)
635     __tag_space_chars_shipout (box)
636 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.

```

637 function ltx.__tag.func.mc_insert_kids (mcnum,single)
638     if ltx.__tag.mc[mcnum] then
639         __tag_log("INFO TEX-MC-INSERT-KID-TEST: " .. mcnum,4)
640         if ltx.__tag.mc[mcnum]["kids"] then
641             if #ltx.__tag.mc[mcnum]["kids"] > 1 and single==1 then
642                 tex.sprint(catlatex,"(")
643             end
644             for i,kidstable in ipairs( ltx.__tag.mc[mcnum]["kids"] ) do
645                 local kidnum = kidstable["kid"]
646                 local kidpage = kidstable["page"]
647                 local kidpageobjnum = pdfpageref(kidpage)
648                 __tag_log("INFO TEX-MC-INSERT-KID: " .. mcnum ..
649                     " insert KID " .. i..
650                     " with num " .. kidnum ..
651                     " on page " .. kidpage.."/"..kidpageobjnum,3)
652                 tex.sprint(catlatex,"<</Type /MCR /Pg "..kidpageobjnum .. " 0 R /MCID "..kidnum.. ">> "

```

```

653     end
654     if #ltx.__tag.mc[mcnum]["kids"] > 1 and single==1 then
655         tex.sprint(catlatex,"")
656     end
657 else
658     -- this is typically not a problem, e.g. empty hbox in footer/header can
659     -- trigger this warning.
660     __tag_log("WARN TEX-MC-INSERT-NO-KIDS: "..mcnum.." has no kids",2)
661     if single==1 then
662         tex.sprint(catlatex,"null")
663     end
664 end
665 else
666     __tag_log("WARN TEX-MC-INSERT-MISSING: "..mcnum.." doesn't exist",0)
667 end
668 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.

```

669 function ltx.__tag.func.store_struct_mcabs (structnum,mcnum)
670     ltx.__tag.struct[structnum]=ltx.__tag.struct[structnum] or { }
671     ltx.__tag.struct[structnum]["mc"]=ltx.__tag.struct[structnum]["mc"] or { }
672     -- a structure can contain more than one mc chunk, the content should be ordered
673     tableinsert(ltx.__tag.struct[structnum]["mc"],mcnum)
674     __tag_log("INFO TEX-MC-INTO-STRUCT: "..
675         mcnum.." inserted in struct "..structnum,3)
676     -- but every mc can only be in one structure
677     ltx.__tag.mc[mcnum]= ltx.__tag.mc[mcnum] or { }
678     ltx.__tag.mc[mcnum]["parent"] = structnum
679 end
680

```

(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.

```

681 -- pay attention: lua counts arrays from 1, tex pages from one
682 -- mcid and arrays in pdf count from 0.
683 function ltx.__tag.func.store_mc_in_page (mcnum,mcpagecnt,page)
684     ltx.__tag.page[page] = ltx.__tag.page[page] or {}
685     ltx.__tag.page[page][mcpagecnt] = mcnum
686     __tag_log("INFO TAG-MC-INTO-PAGE: page " .. page ..
687         ": inserting MCID " .. mcpagecnt .. " => " .. mcnum,3)
688 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)

```

689 local function __tag_update_mc_attributes (head,mcnum,type)
690     for n in node.traverse(head) do
691         node.set_attribute(n,mccntattributeid,mcnum)

```

```

692 node.set_attribute(n,mctypeattributeid,type)
693 if n.id == HLIST or n.id == VLIST then
694     __tag_update_mc_attributes (n.list,mcnum,type)
695 end
696 end
697 return head
698 end
699 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.

```

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

```

```

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

```

```

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

```

```

851 __tag_log ("INFO TAG-QUITTING-BOX "..
852           tostring(name)..
853           " TYPE ".. node.type(node.getid(box)),4)
854 return mcopen,mcpagecnt,mccntprev,mctypeprev
855 end
856

```

(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.

```

857 function ltx.__tag.func.mark_shipout (box)
858   mcopen = ltx.__tag.func.mark_page_elements (box,-1,-100,0,"Shipout",-1)
859   if mcopen~=0 then -- there is a chunk open, close it (hope there is only one ...
860     local emcnode = __tag_backend_create_emc_node ()
861     local list = box.list
862     if list then
863       list = node.insert_after (list,node.tail(list),emcnode)
864       mcopen = mcopen - 1
865       __tag_log ("INFO SHIPOUT-INSERT-LAST-EMC: MCOPEN " .. mcopen,3)
866     else
867       __tag_log ("WARN SHIPOUT-UPS: this shouldn't happen",0)
868     end
869     if mcopen ~=0 then
870       __tag_log ("WARN SHIPOUT-MC-OPEN: " .. mcopen,1)
871     end
872   end
873 end

```

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

7 Parenttree

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

```

874 function ltx.__tag.func.fill_parent_tree_line (page)
875   -- we need to get page-> i=kid -> mcnum -> structnum
876   -- pay attention: the kid numbers and the page number in the parent tree start with 0!
877   local numsentry = ""
878   local pdfpage = page-1
879   if ltx.__tag.page[page] and ltx.__tag.page[page][0] then
880     mcchunks=#ltx.__tag.page[page]
881     __tag_log("INFO PARENTTREE-NUM: page "..
882             page.." has "..mcchunks.." +1 Elements ",4)
883     for i=0,mcchunks do
884       -- what does this log??
885       __tag_log("INFO PARENTTREE-CHUNKS: "..
886               ltx.__tag.page[page][i],4)
887     end
888     if mcchunks == 0 then
889       -- only one chunk so no need for an array
890       local mcnum = ltx.__tag.page[page][0]

```

```

891     local structnum = ltx.__tag.mc[mcnum]["parent"]
892     local propname = "g__tag_struct"..structnum.."_prop"
893     --local objref = ltx.__tag.tables[propname]["objref"] or "XXXX"
894     local objref = __tag_pdf_object_ref('__tag/struct',structnum)
895     __tag_log("INFO PARENTTREE-STRUCT-OBJREF: =====>"..
896         tostring(objref),5)
897     numsentry = pdfpage .. " [".. objref .. "]"
898     __tag_log("INFO PARENTTREE-NUMENTRY: page " ..
899         page.. " num entry = ".. numsentry,3)
900     else
901         numsentry = pdfpage .. " ["
902         for i=0,mcchunks do
903             local mcnum = ltx.__tag.page[page][i]
904             local structnum = ltx.__tag.mc[mcnum]["parent"] or 0
905             local propname = "g__tag_struct"..structnum.."_prop"
906             --local objref = ltx.__tag.tables[propname]["objref"] or "XXXX"
907             local objref = __tag_pdf_object_ref('__tag/struct',structnum)
908             numsentry = numsentry .. " [".. objref
909             end
910             numsentry = numsentry .. "]"
911             __tag_log("INFO PARENTTREE-NUMENTRY: page " ..
912                 page.. " num entry = ".. numsentry,3)
913             end
914         else
915             __tag_log ("INFO PARENTTREE-NO-DATA: page "..page,3)
916             numsentry = pdfpage.." ["
917             end
918         return numsentry
919     end
920
921 function ltx.__tag.func.output_parenttree (abspage)
922     for i=1,abspage do
923         line = ltx.__tag.func.fill_parent_tree_line (i) .. "^^J"
924         tex.sprint(catlatex,line)
925     end
926 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.

```

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

```

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

```

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

```

```

938     t[fid] = result
939     return result
940 end})

```

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

```

941 local function process_softhyphen_pre(head, _context, _dir)
942   if softhyphenbool.mode ~= truebool.mode then return true end
943   for disc, sub in node.traverse_id(DISC, head) do
944     if sub == explicit_disc or sub == regular_disc then
945       for n, _ch, _f in node.traverse_char(disc.pre) do
946         local props = properties[n]
947         if not props then
948           props = {}
949           properties[n] = props
950         end
951         props[is_soft_hyphen_prop] = true
952       end
953     end
954   end
955   return true
956 end
957

```

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

```

958 local function process_softhyphen_post(head, _context, _dir)
959   if softhyphenbool.mode ~= truebool.mode then return true end
960   for disc, sub in node.traverse_id(DISC, head) do
961     for n, ch, fid in node.traverse_glyph(disc.pre) do
962       local props = properties[n]
963       if ch == hyphen_char and props and props[is_soft_hyphen_prop] then
964         if nodegetattribute(n,softhyphenattribute) and softhyphen_fonts[fid] then
965           n.char = soft_hyphen_char
966           props.glyph_info = nil
967         else
968           nodesetattribute(n,mctypeattributeid,-2147483647)
969           nodesetattribute(n,mccntattributeid,-2147483647)
970         end
971       end
972     end
973   end
974   return true
975 end
976
977 luatexbase.add_to_callback('pre_shaping_filter', process_softhyphen_pre, 'tagpdf.rewrite-
softhyphen')
978 luatexbase.add_to_callback('post_shaping_filter', process_softhyphen_post, 'tagpdf.rewrite-
softhyphen')
979 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`

```

980 local function role_get_parent_child_rule (parent,child)
981     local state=
982     ltx.__tag.role.matrix[ltx.__tag.role.index[parent]]
983     and ltx.__tag.role.matrix[ltx.__tag.role.index[parent]][ltx.__tag.role.index[child]] or 0
984     return state
985 end
986 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.

```

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

```

```

1019     local r,rNS=
1020         string.match(ltx.__tag.tables['g__tag_struct_'.i..'_prop']['rolemap'], "{(-
1021 )}{(-)}")
1022     local p,pNS=
1023         string.match(ltx.__tag.tables['g__tag_struct_'.i..'_prop']['parentrole'], "{(-
1024 )}{(-)}")
1025     local parent=ltx.__tag.tables['g__tag_struct_'.i..'_prop']['parentnum']
1026     if parent then
1027         __tag_log (i..'': '.. t..'': '..tNS,loglevel)
1028         __tag_log (i..'': '.. r..'': '..rNS,loglevel)
1029         __tag_log (i..'': '.. p..'': '..pNS,loglevel)
1030         __tag_log ('parent of ' ..i..'': '.. parent,loglevel )
1031         if p == 'STASHED' then
1032             check_update_stashed (i,loglevel,0)
1033             p,pNS=
1034                 string.match(ltx.__tag.tables['g__tag_struct_'.i..'_prop']['parentrole'], "{(-
1035 )}{(-)}")
1036         end
1037         local pt,ptNS=
1038             string.match(ltx.__tag.tables['g__tag_struct_'.parent..'_prop']['tag'], "{(-
1039 )}{(-)}")
1040         local pr,prNS=
1041             string.match(ltx.__tag.tables['g__tag_struct_'.parent..'_prop']['rolemap'], "{(-
1042 )}{(-)}")
1043         local pp,ppNS=
1044             string.match(ltx.__tag.tables['g__tag_struct_'.parent..'_prop']['parentrole'], "{(-
1045 )}{(-)}")
1046         if pp == 'STASHED' then
1047             check_update_stashed (parent,loglevel,0)
1048             pp,ppNS=
1049                 string.match(ltx.__tag.tables['g__tag_struct_'.parent..'_prop']['parentrole'], "{(-
1050 )}{(-)}")
1051         end
1052         __tag_log (parent..'': '.. pt..'': '..ptNS,loglevel)
1053         __tag_log (parent..'': '.. pr..'': '..prNS,loglevel)
1054         __tag_log (parent..'': '.. pp..'': '..ppNS,loglevel)
1055         -- now check the rule.
1056         -- at first rolemap of child against rolemap of parent.
1057         local state=ltx.__tag.func.role_get_parent_child_rule (pr,r)
1058         __tag_log ('rule of '..pr.."-">"..r..' is '..state,loglevel)
1059         -- if the state is 7 we check against parentrole of the parent
1060         if state == 7 then
1061             state=ltx.__tag.func.role_get_parent_child_rule (pp,r)
1062             __tag_log ('Parent-Child relation '..pp.."-">"..r..' is '..state,loglevel)
1063         end
1064         if state == 0 then
1065             __tag_log
1066                 ('Warning: Parent-Child relation '
1067                 ..ptNS..'': '..pt..' -> '..tNS..'': '..t..' is unknown',0)
1068             __tag_log
1069                 ('Structure ' ..parent..' -> '..i,0)
1070         end
1071         if state == -1 then
1072             __tag_log

```

```

1066      ('Warning: Parent-Child relation '
1067      ..ptNS..'':'..pt..' -> '..tNS..'':'..t..' is not allowed',0)
1068    __tag_log
1069      ('Structure ' ..parent..' -> '..i,0)
1070  end
1071  -- check also for MC
1072  state =ltx.__tag.func.role_get_parent_child_rule ( r , 'MC')
1073  local curtag=r
1074  if state == 7 then
1075    state =ltx.__tag.func.role_get_parent_child_rule ( p , 'MC')
1076    local curtag=p
1077  end
1078  if state == -1 then
1079    if ltx.__tag.struct[i] and NEXT(ltx.__tag.struct[i]) then
1080      __tag_log
1081      ('Warning: Real content (MC) is not allowed in ' ..curtag,0)
1082    end
1083  end
1084  __tag_log('=====','loglevel)
1085  end
1086  end -- end for
1087  end
1088
1089  ltx.__tag.func.check_parent_child_rules=check_parent_child_rules
1090

```

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

9 Link annotations

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

```

1091  local linkbincnt = 0
1092
1093  function ltx.__tag.func.linkbin()
1094    return linkbincnt
1095  end
1096
1097  if luatexbase.callbacktypes['linksplit'] then
1098    luatexbase.add_to_callback('linksplit', function(start_link, position)
1099      if start_link == nil then return end
1100      local structnum =
1101        node.get_attribute(start_link,luatexbase.attributes.g__tag_structnum_attr)
1102      if structnum and structnum > -1 then
1103        local s = ltx.__tag.tables['g__tag_struct_..'..structnum..'.._prop']['rolemap']
1104        if s and (string.find(s,'Link') or string.find(s,'Reference')) then
1105          local struct_insert_annot_shipout = token.create__tag_struct_insert_annot_shipout:
1106          local parentnum = tex.count['c@g__tag_parenttree_obj_int']
1107          start_link.link_attr =
1108            start_link.link_attr ..
1109            ' /LTEX_position /' .. position ..
1110            '/StructParent ' .. parentnum

```

```

1111 tex.sprint(catlatex,struct_insert_annot_shipout,'{'..
1112 structnum..'}{'..
1113 start_link.objnum..' 0 R'}{'..
1114 parentnum ..'}')
1115 -- the counter must be set explicitly as struct_insert_annot_shipout doesn't do it!
1116 tex.setcount('global','c@g__tag_parenttree_obj_int',parentnum +1)
1117 __tag_log(position .. " link part has object id " .. start_link.objnum .. " and str
1118 else
1119 local linkbin = token.get_macro("c__tag_struct_link_bin_tl")
1120 if linkbin then
1121 local struct_insert_annot_shipout = token.create'__tag_struct_insert_annot_shipout.'
1122 local parentnum = tex.count['c@g__tag_parenttree_obj_int']
1123 start_link.link_attr = string.gsub (start_link.link_attr, "/Contents%s+<%w+>", "")
1124 start_link.link_attr =
1125 start_link.link_attr ..
1126 ' /LTEX_position /' .. position ..
1127 '/StructParent ' .. parentnum .. '/Contents (artifact)'
1128 tex.sprint(catlatex,struct_insert_annot_shipout,'{'..
1129 linkbin..'}{'..
1130 start_link.objnum..' 0 R'}{'..
1131 parentnum ..'}')
1132 tex.setcount('global','c@g__tag_parenttree_obj_int',parentnum +1)
1133 linkbincnt=linkbincnt+1
1134 if linkbincnt == 1 then
1135 __tag_log('Info: some Link annotation(s) are not in Link or Reference structure el
1136 __tag_log(' moved into a container at the end of the structure tree\n',0)
1137 end
1138 else
1139 __tag_log('Warning: Link not in Link or Reference structure element',0)
1140 __tag_log('OBJR not created',0)
1141 __tag_log('',0)
1142 end
1143 end
1144 end
1145 end, 'tagpdf')
1146 end
1147 </lua>

```

Ulrike Fischer
Version 1.0e, released 2026-08-21

Part X

The tagpdf-roles module

Tags, roles and namespace code

```
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.

```
\tag_if_tag_known:nnTF \tag_if_tag_known:nnTF {<tag>} {<true code>} {<false code>}
```

Check if a specified structure tag has been declared in the current document.

```
1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-roles-code} {2026-08-21} {1.0e}
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_linked:N \g__tag_role_tags_NS_prop

(End of definition for \g__tag_role_tags_NS_prop.)

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

8 \prop_new:N \g__tag_role_tags_class_prop

(End of definition for \g__tag_role_tags_class_prop.)

`\g__tag_role_NS_prop` This holds the list of supported name spaces. The keys are the name tagpdf will use, the values the object reference. The urls identifier are stored in related dict object.

mathml <http://www.w3.org/1998/Math/MathML>

pdf2 <http://iso.org/pdf2/ssn>

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

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

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

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

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

9 \prop_new:N \g__tag_role_NS_prop

(End of definition for \g__tag_role_NS_prop.)

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

10 \prop_new:N \g__tag_role_index_prop

(End of definition for \g__tag_role_index_prop.)

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

11 \prop_new:N \l__tag_role_debug_prop

(End of definition for \l__tag_role_debug_prop.)

We need also a bunch of temporary variables.

`\l__tag_role_tag_tmpa_tl`

`\l__tag_role_tag_namespace_tmpa_tl`

12 \tl_new:N \l__tag_role_tag_tmpa_tl

`\l__tag_role_tag_namespace_tmpb_tl` %

13 \tl_new:N \l__tag_role_tag_namespace_tmpa_tl

`\l__tag_role_role_tmpa_tl`

14 \tl_new:N \l__tag_role_tag_namespace_tmpb_tl

`\l__tag_role_role_namespace_tmpa_tl`

15 \tl_new:N \l__tag_role_role_tmpa_tl

`\l__tag_role_role_namespace_tmpa_tl`

16 \tl_new:N \l__tag_role_role_namespace_tmpa_tl

`\l__tag_role_role_tmpa_seq`

17 \seq_new:N \l__tag_role_role_tmpa_seq

(End of definition for \l__tag_role_tag_tmpa_tl and others.)

1.2 Namespaces

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

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

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

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

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

```
\__tag_role_NS_new:nnn
```

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

```

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

```

(End of definition for __tag_role_NS_new:nnn.)

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

\c__tag_role_userNS_id_str

```

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

```

(End of definition for \c__tag_role_userNS_id_str.)

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

```

73 \bool_new:N \g__tag_role_add_mathml_bool
74 \__tag_role_NS_new:nnn {pdf} {http://iso.org/pdf/ssn}{}
75 \__tag_role_NS_new:nnn {pdf2} {http://iso.org/pdf2/ssn}{}
76 \__tag_role_NS_new:nnn {mathml}{http://www.w3.org/1998/Math/MathML}{}
77 \__tag_role_NS_new:nnn {latex} {https://www.latex-project.org/ns/dfltl}{}
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_tmpa_tl
138   {
139     \tl_set:Nn\l__tag_tmpa_tl{--UNKNOWN--}
140   }
141   \__tag_role_alloctag:nno {#1}{user} { \l__tag_tmpa_tl }
```

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

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

(End of definition for __tag_role_add_tag:nn.)

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

__tag_role_get:nnNN

```
154 \pdf_version_compare:NnT < {2.0}
155 {
156   \cs_new_protected:Npn \__tag_role_get:nnNN #1#2#3#4 {%#1 tag, #2 NS, #3 tlvar which hold th
157   {
158     \prop_get:NnNF \g__tag_role_rolemapping_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:nNt {\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_protected: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.

```

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 }

```

```

387     }
388   }
389 }

```

1.6 Parent-child rules

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

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

```

390 \intarray_new:Nn \g_tag_role_parent_child_intarray {6000}

```

(End of definition for `\g_tag_role_parent_child_intarray`.)

`\c_tag_role_rules_prop` These two properties map the rule strings to numbers and back. There are in tagpdf-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:nnm` The helper command is used to store the rule. It assumes that parent and child are given as 2-digit number!

```

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

```

```

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

```

(End of definition for __tag_store_parent_child_rule:nnn.)

1.6.1 Reading in the csv-files

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

```

430 \int_zero:N \l__tag_tmpa_int

```

Open the file depending on the PDF version

```

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

```

Now the main loop over the file

```

438 \ior_map_inline:Nn \g_tmpa_ior
439 {

```

ignore lines containing only comments

```

440     \tl_if_empty:nF{#1}
441     {

```

count the lines ...

```

442         \int_incr:N\l__tag_tmpa_int

```

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

```

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

```

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

```

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

```

now the data lines.

```

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

```

get the name of the child tag from the first column

```

455         \seq_pop_left:NN\l__tag_tmpa_seq\l__tag_tmpa_tl

```

get the number of the child, and store it in `\l__tag_tmpb_tl`

```
456 \prop_get:NnN \g__tag_role_index_prop { \l__tag_tmpa_tl } \l__tag_tmpb_tl
```

remove column 2+3

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

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

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

```
459 \seq_map_indexed_inline:Nn \l__tag_tmpa_seq
```

```
460 {
```

```
461 \exp_args:Nne
```

```
462 \__tag_store_parent_child_rule:nnn {##1}{\l__tag_tmpb_tl}{ ##2 }
```

```
463 }
```

```
464 }
```

```
465 }
```

```
466 }
```

close the read handle.

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

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

```
468 \prop_get:NnN \g__tag_role_index_prop {StructTreeRoot} \l__tag_tmpa_tl
```

```
469 \prop_gput:Nne \g__tag_role_index_prop {Root} { \l__tag_tmpa_tl }
```

```
470 \prop_get:NnN \g__tag_role_index_prop {Hn} \l__tag_tmpa_tl
```

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

```
472 {
```

```
473 \int_step_inline:nn{6}
```

```
474 {
```

```
475 \prop_gput:Nne \g__tag_role_index_prop {H#1} { \l__tag_tmpa_tl }
```

```
476 }
```

```
477 }
```

```
478 {
```

```
479 \int_step_inline:nn{10}
```

```
480 {
```

```
481 \prop_gput:Nne \g__tag_role_index_prop {H#1} { \l__tag_tmpa_tl }
```

```
482 }
```

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

```
483 \prop_get:NnN \g__tag_role_index_prop {mathml} \l__tag_tmpa_tl
```

```
484 \prop_get:NnN \g__tag_role_index_prop {math} \l__tag_tmpb_tl
```

```
485 \prop_get:NnN \g__tag_role_index_prop {mtext} \l__tag_tmpc_tl
```

```
486 \prop_map_inline:Nn \g__tag_role_NS_mathml_prop
```

```
487 {
```

```
488 \prop_gput:Nno \g__tag_role_index_prop {#1} { \l__tag_tmpa_tl }
```

```
489 }
```

```
490 \prop_gput:Nno \g__tag_role_index_prop {math} { \l__tag_tmpb_tl }
```

```
491 \prop_gput:Nno \g__tag_role_index_prop {mtext} { \l__tag_tmpc_tl }
```

```
492 }
```

```
493 \sys_if_engine luatex:T
```

```
494 {
```

```
495 \prop_map_inline:Nn \g__tag_role_index_prop
```

```
496 {
```

```
497 \lua_now:e { ltx.__tag.role.index['#1']=#2 }
```

```
498 }
```

```
499 }
```

1.6.2 Retrieving the parent-child rule

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{\int_use:N\c_document_cctab,ltx.__tag.func.role_get_parent_ch
511       }

```

Debugging messages, this can perhaps go into debug mode.

```

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

```

```

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

```

Get the rule from the intarray

```

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

```

and now get the relation

```

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

```

(End of definition for __tag_role_check_parent_child:nnnnN.)

\tag_check_child:nnTF

```

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

```

```

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

```

(End of definition for `\tag_check_child:nNnTF`. This function is documented on page 186.)

1.7 Key-val user interface

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

```

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

```

```

774         \exp_args:NNno \seq_set_split:Nnn \l__tag_tmpa_seq { / } {\l__tag_tmpa_tl/}
775         \tl_set:Ne \l__tag_role_tag_tmpa_tl { \seq_item:Nn \l__tag_tmpa_seq {1} }
776         \tl_set:Ne \l__tag_role_role_tmpa_tl { \seq_item:Nn \l__tag_tmpa_seq {2} }
777     }
778     \tl_if_empty:NT \l__tag_role_role_namespace_tmpa_tl
779     {
780         \prop_get:NoNTF
781         \g__tag_role_tags_NS_prop
782         { \l__tag_role_role_tmpa_tl }
783         \l__tag_role_role_namespace_tmpa_tl
784         {
785             \prop_get:NoNF
786             \g__tag_role_NS_prop
787             { \l__tag_role_role_namespace_tmpa_tl }
788             \l__tag_tmp_unused_tl
789             {
790                 \tl_set:Nn \l__tag_role_role_namespace_tmpa_tl {user}
791             }
792         }
793         {
794             \tl_set:Nn \l__tag_role_role_namespace_tmpa_tl {user}
795         }
796     }
797     \pdf_version_compare:NnTF < {2.0}
798     {
799         %TODO add check for emptyness?
800         \__tag_role_add_tag:oo
801         { \l__tag_role_tag_tmpa_tl }
802         { \l__tag_role_role_tmpa_tl }
803     }
804     {
805         \__tag_role_add_tag:oooo
806         { \l__tag_role_tag_tmpa_tl }
807         { \l__tag_role_tag_namespace_tmpa_tl }
808         { \l__tag_role_role_tmpa_tl }
809         { \l__tag_role_role_namespace_tmpa_tl }
810     }
811 }
812 ,role/map-tags .choice:
813 ,role/map-tags/false .code:n = { \socket_assign_plug:nn { tag/struct/tag } {latex-
tags} }
814 ,role/map-tags/pdf .code:n = { \socket_assign_plug:nn { tag/struct/tag } {pdf-
tags} }

815 ,role/user-NS .code:n =
816 {
817     \pdf_version_compare:NnF < {2.0}
818     {
819         \pdf_string_from_unicode:nnN{utf8/string}{https://www.latex-project.org/ns/local/#1}
820         \tl_if_empty:NF \l__tag_tmpa_str
821         {
822             \pdfdict_gput:nne
823             {g__tag_role/Namespace_user_dict}
824             {NS}

```

```

825         {\l__tag_tmpa_str}
826     }
827 }
828 }

```

deprecated names

```

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

```

(End of definition for tag (rolemap-key) and others. These functions are documented on page 186.)

\tag_if_tag_known:nTF

```

832 \prg_new_protected_conditional:Nnn \tag_if_tag_known:n { T, F, TF }
833 {
834   \prop_get:NnNTF\g__tag_role_tags_NS_prop{#1}\l__tag_tmp_unused_tl { \prg_return_true: } t
835 }
836 \endpackage

```

(End of definition for \tag_if_tag_known:nTF. This function is documented on page 186.)

Ulrike Fischer

Version 1.0e, released 2026-08-21

Part XI

The tagpdf-space module

Code related to real space chars

`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} {2026-08-21} {1.0e}
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         \AddToHook{shipout/firstpage}[tagpdf/space]{}
32         \keys_define:nn { __tag / setup }
33         {
34             activate/spaces/true .code:n = { \AddToHook{shipout/firstpage}[tagpdf/space]{\p
35             activate/spaces/false .code:n = { \RemoveFromHook{shipout/firstpage}[tagpdf/space
36             activate/spaces .default:n = true,
37         }
38     }
39     {
40         \keys_define:nn { __tag / setup }
41         {
42             activate/spaces .choices:nn = { true, false }
43             { \msg_warning:nnn {tag}{sys-no-interwordspace}{dvi} },
44             activate/spaces .default:n = true,
45         }
46     }
47 }
48
49
50 \sys_if_engine_luatex:T
51 {
52     \keys_define:nn { __tag / setup }
53     {
54         activate/spaces .choice:,
55         activate/spaces/true .code:n =
56         {
57             \bool_gset_true:N \g__tag_active_space_bool
58             \lua_now:e{!tx.__tag.func.markspaceon()}
59         },
60         activate/spaces/false .code:n =
61         {
62             \bool_gset_false:N \g__tag_active_space_bool
63             \lua_now:e{!tx.__tag.func.markspaceoff()}
64         },
65         activate/spaces .default:n = true,
66         debug/show/spaces .code:n =
67         { \lua_now:e{!tx.__tag.trace.showspace=true} },
68         debug/show/spacesOff .code:n =
69         { \lua_now:e{!tx.__tag.trace.showspace=nil} },
70     }
71 }

```

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

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

```

72 \sys_if_engine_luatex:T

```

```

73 {
74   \cs_new_protected:Nn \__tag_fakespace:
75   {
76     \group_begin:
77     \lua_now:e{\ltx.__tag.func.fakespace()}
78     \skip_horizontal:n{\c_zero_skip}
79     \group_end:
80   }
81 }

```

We need also a command to interrupt the insertion of real space chars in places where we want to insert manually special spaces. In pdftex this can be done with `\pdfinterwordsspaceoff` and `\pdfinterwordsspaceon`. 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: 82 \cs_new_protected:Npn \tag_spacechar_off: {}
\tag_spacechar_off: 83 \cs_new_protected:Npn \tag_spacechar_on: {}
84
85 \sys_if_engine_luatex:T
86 {
87   \cs_set_protected:Npn \tag_spacechar_off:
88   {
89     \lua_now:e
90     {
91       tex.setattribute
92       (
93         "global",
94         luatexbase.attributes.g__tag_interwordsspaceOff_attr,
95         1
96       )
97     }
98   }
99   \cs_set_protected:Npn \tag_spacechar_on:
100   {
101     \lua_now:e
102     {
103       tex.setattribute
104       (
105         "global",
106         luatexbase.attributes.g__tag_interwordsspaceOff_attr,
107         -2147483647
108       )
109     }
110   }
111 }
112 \sys_if_engine_pdftex:T
113 {
114   \sys_if_output_pdf:T
115   {
116     \cs_set_protected:Npn \tag_spacechar_off:
117     {
118       \pdfinterwordsspaceoff
119     }

```

```

120     \cs_set_protected:Npn \tag_spacechar_on:
121     {
122         \pdfinterwordspaceon
123     }
124 }
125
126 \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, 222, 223, 579, 642, 650
_	424, 435
A	
activate_ (setup-key)	44, <u>274</u>
activate-all (deprecated) (key)	1
activate-mc (deprecated) (key)	1
activate-struct (deprecated) (key)	1
activate-tree (deprecated) (key)	1
activate/all (key)	1, <u>241</u>
activate/collect-link-artifacts (key)	1, <u>339</u>
activate/mc (key)	1, <u>241</u>
activate/socket_ (setup-key)	274
activate/softhyphen (key)	1, <u>275</u>
activate/space_ (setup-key)	209
activate/spaces (key)	1
activate/spaces_ (setup-key)	6
activate/struct (key)	1, <u>241</u>
activate/struct-dest (key)	1, <u>241</u>
activate/tagunmarked (key)	1, <u>272</u>
activate/tree (key)	1, <u>241</u>
actualtext (key)	1, <u>758</u>
actualtext_ (mc-key)	83, <u>238</u> , <u>382</u>
add-new-tag_ (deprecated)	752
add-new-tag_ (setup-key)	186
\AddToHook	13, 16, 31, 34, 44, 57, 58, 227, 239, 242, 292, 439, 441, 442, 444
\AddToHookNext	323, 637
AF (key)	1, <u>971</u>
AFinline (key)	1, <u>971</u>
AFinline-o (key)	1, <u>971</u>
AFref (key)	1, <u>971</u>
alt (key)	1, <u>758</u>
alt_ (mc-key)	83, <u>238</u> , <u>382</u>
artifact_ (mc-key)	83, <u>238</u> , <u>382</u>
artifact-bool internal commands:	
__artifact-bool	182
artifact-type internal commands:	
__artifact-type	182
\AssignStructureRole	388, 393, 403, 408
\AssignTaggingSocketPlug	491, 498, 600, 601, 658, 667
\AtBeginDocument	584
attr-unknown	24, <u>84</u>
attribute (key)	1, <u>1627</u>
attribute-class (key)	1, <u>1593</u>
B	
benchmark commands:	
\benchmark_tic:	659, 661
\benchmark_toc:	662
bool commands:	
\bool_gset_eq:NN	508, 523, 535, 553, 608, 622
\bool_gset_false:N	62, 221, 276, 277, 370, 509, 536, 609
\bool_gset_true:N	36, 57, 87, 95, 175, 280, 295, 310, 311, 325, 879
\bool_if:NTF	9, 13, 18, 29, 38, 40, 64, 68, 69, 74, 80, 85, 100, 124, 135, 196, 203, 208, 226, 248, 264, 265, 303, 321, 329, 337, 351, 360, 369, 389, 421, 432, 478, 503, 518, 530, 548, 603, 617, 1222, 1254, 1285
\bool_if:nTF	529
\bool_lazy_all:nTF	262
\bool_lazy_and:nnTF	294, 309, 314, 324, 355, 373, 548, 619, 670, 697, 771, 882
\bool_new:N	7, 16, 20, 21, 35, 73, 82, 83, 84, 85, 86, 87, 88, 90, 92, 93, 94, 255, 312, 313, 499, 878
\bool_set_false:N	17, 164, 165, 166, 176, 188, 189, 190, 222, 344, 384, 473, 502, 529, 602
\bool_set_true:N	16, 89, 91, 174, 175, 176, 199, 200, 201, 256, 346, 383, 472
box commands:	
\box_dp:N	178, 182
\box_ht:N	168
\box_new:N	77, 78
\box_set_dp:Nn	176, 178
\box_set_eq:NN	191
\box_set_ht:Nn	175, 177
\box_use_drop:N	180, 184
\boxmaxdepth	96, 179
C	
c@g internal commands:	
\c@g__tag_MCID_abs_int	11, 15, 26, 35, 48, 55, 60, 66, 72, 133, 136, 178, 240, 243, 286, 293, 344

\c@g__tag_parenttree_obj_int	155 , 512	\cs_new_protected:Nn	
\c@g__tag_struct_abs_int			74 , 83 , 127 , 167 , 294 , 424 , 428
.....	6 , 20 , 40 , 58 , 91 ,	\cs_new_protected:Npn	
.....	114 , 115 , 118 , 124 , 127 , 149 , 166 ,		13 , 15 , 20 , 21 , 22 , 27 ,
.....	225 , 248 , 378 , 586 , 763 , 776 , 821 ,		30 , 31 , 36 , 42 , 43 , 43 , 60 , 61 , 61 , 65 ,
.....	833 , 847 , 863 , 878 , 886 , 955 , 966 ,		67 , 76 , 79 , 80 , 81 , 82 , 82 , 83 , 85 , 86 ,
.....	985 , 988 , 993 , 1029 , 1031 , 1036 ,		91 , 93 , 95 , 106 , 106 , 107 , 108 , 117 ,
.....	1048 , 1050 , 1055 , 1146 , 1157 , 1158 ,		120 , 122 , 126 , 137 , 142 , 147 , 151 ,
.....	1159 , 1160 , 1161 , 1163 , 1165 , 1171 ,		156 , 163 , 165 , 169 , 171 , 172 , 173 ,
.....	1176 , 1183 , 1186 , 1196 , 1206 , 1210 ,		187 , 187 , 207 , 210 , 211 , 212 , 213 ,
.....	1225 , 1238 , 1248 , 1261 , 1264 , 1279 ,		214 , 215 , 220 , 229 , 233 , 236 , 238 ,
.....	1280 , 1282 , 1293 , 1620 , 1623 , 1671		239 , 246 , 253 , 258 , 259 , 260 , 263 ,
catalog-supplemental-file (key)	... 1126		264 , 275 , 276 , 279 , 283 , 286 , 292 ,
cctab commands:			292 , 293 , 317 , 318 , 322 , 327 , 332 ,
\c_document_cctab	49 , 54 , 75 , 144 , 510		334 , 337 , 338 , 342 , 342 , 346 , 349 ,
\chapter	 197 , 357 , 375		349 , 350 , 358 , 362 , 369 , 372 , 376 ,
check commands:			391 , 393 , 407 , 410 , 417 , 418 , 418 ,
check_parent_child_rules	 987		425 , 429 , 437 , 440 , 451 , 456 , 474 ,
check_update_stashed	 987		475 , 480 , 481 , 482 , 483 , 500 , 503 ,
clist commands:			514 , 515 , 527 , 541 , 543 , 548 , 549 ,
\clist_const:Nn	 79 , 80		553 , 556 , 557 , 563 , 570 , 590 , 590 ,
\clist_count:n	 1528		592 , 598 , 598 , 605 , 613 , 614 , 614 ,
\clist_if_empty:NTF	 1632		620 , 640 , 655 , 656 , 657 , 670 , 891 ,
\clist_map_inline:nn	... 106 , 431 , 952		899 , 912 , 925 , 940 , 973 , 1001 , 1146 ,
\clist_new:N	 75		1147 , 1148 , 1347 , 1388 , 1441 , 1452 ,
\clist_set:Nn	 1597 , 1631		1461 , 1481 , 1485 , 1489 , 1493 , 1497 ,
\clist_use:nn	 1534		1503 , 1522 , 1526 , 1537 , 1561 , 1566
color commands:		\cs_set:Nn	 568 , 569 , 630 , 631
\color_select:n	 424 , 435	\cs_set:Npn	 47 , 52 , 78 , 98
cs commands:		\cs_set_eq:NN	 14 , 20 , 66 ,
\cs:w	 1470 , 1474		78 , 79 , 80 , 139 , 140 , 141 , 142 , 143 ,
\cs_end:	 1470 , 1474		144 , 145 , 146 , 147 , 148 , 181 , 182 ,
\cs_generate_variant:Nn			193 , 207 , 216 , 217 , 223 , 230 , 233 ,
.....	42 , 79 , 98 , 99 ,		234 , 235 , 236 , 371 , 372 , 373 , 374 ,
.....	100 , 101 , 102 , 103 , 104 , 105 , 105 ,		561 , 562 , 563 , 564 , 570 , 571 , 575 ,
.....	106 , 107 , 107 , 113 , 115 , 116 , 117 ,		576 , 577 , 578 , 632 , 633 , 661 , 662 , 1574
.....	126 , 134 , 150 , 151 , 152 , 153 , 153 ,	\cs_set_protected:Nn	
.....	154 , 155 , 156 , 163 , 164 , 168 , 181 ,	..	169 , 216 , 264 , 357 , 363 , 1303 , 1304
.....	206 , 226 , 226 , 253 , 259 , 263 , 274 ,	\cs_set_protected:Npn	 9 , 15 ,
.....	275 , 291 , 316 , 326 , 357 , 589 , 669 ,		16 , 22 , 29 , 35 , 38 , 40 , 48 , 49 , 52 , 58 ,
.....	717 , 717 , 972 , 1000 , 1021 , 1432 ,		62 , 63 , 65 , 67 , 71 , 72 , 83 , 83 , 87 , 99 ,
.....	1439 , 1451 , 1502 , 1546 , 1575 , 1576		102 , 107 , 116 , 120 , 143 , 160 , 169 ,
\cs_gset_eq:NN	 454		183 , 194 , 220 , 228 , 238 , 240 , 261 ,
\cs_if_exist:NTF	 253 , 639 , 659		277 , 295 , 297 , 303 , 352 , 356 , 360 ,
\cs_if_exist_p:N	.. 357 , 360 , 375 , 378		364 , 1150 , 1151 , 1341 , 1349 , 1390 , 1443
\cs_if_exist_use:NTF		\cs_to_str:N	
.....	237 , 239 , 412 , 1445	..	12 , 18 , 25 , 32 , 38 , 43 , 61 , 62 , 68 , 69
\cs_if_free:NTF	 48	\cs_undefine:N	 57
\cs_new:Nn			
.....	109 , 131 , 136 , 291 , 421 , 422 , 423		
\cs_new:Npn			
.....	9 , 15 , 25 , 117 , 118 , 161 , 227 ,		
.....	235 , 237 , 540 , 572 , 578 , 584 , 1435 , 1547		
\cs_new_eq:NN	 39		

D

debug/log (key)	 1 , 259
debug/show (key)	 258
debug/structures_ (show-key)	... 45 , 243
debug/uncompress (key)	 259

\DebugSocketsOn	47	hook commands:	
\DeclareOption	37, 38	\hook_gput_code:nnn	7, 11, 33, 57, 66, 80, 156, 237, 277, 278, 353, 371, 387, 391, 682, 689, 696, 703, 710, 717, 723, 730, 736, 743, 751, 764, 775, 788, 799, 812, 823, 836, 846, 859
dim commands:		\hook_new:n	348
\c_max_dim	167, 192	\hook_use:n	353
\c_zero_dim	175, 176, 177		
\directlua	230		
\documentclass	16		
\DocumentMetadata	15		
E			
E (key)	1, 758, 948	I	
\endinput	22	\IfPDFManagementActiveF	6
\ERRORusetaggingsocket	95, 110	\ignorespaces	44
exclude-header-footer _□ (deprecated)	556	int commands:	
exp commands:		\int_abs:n	166
\exp_args:Ne	120, 542	\int_case:nnTF	88, 103, 341, 1528
\exp_args:NNe	84, 87, 193, 213	\int_compare:nNnTF	22, 58, 70, 98, 114, 124, 125, 132, 137, 142, 157, 170, 171, 173, 230, 293, 351, 381, 400, 427, 430, 444, 450, 450, 458, 459, 464, 512, 528, 551, 558, 562, 565, 572, 578, 592, 600, 607, 615, 619, 622, 655, 734, 748, 1187
\exp_args:Nne	79, 337, 341, 425, 461	\int_compare:nTF	180, 498, 1613, 1615, 1617, 1641, 1667
\exp_args:NNno	774	\int_compare_p:nNn	776
\exp_args:No	289, 324	\int_decr:N	171, 196
\exp_last_unbraced:Ne	99, 102, 109	\int_eval:n	118, 136, 166, 197, 396, 643, 651, 773, 778, 781, 993, 1036, 1055, 1158, 1159, 1160, 1161, 1279, 1280, 1282, 1293, 1623
\exp_last_unbraced:No	135, 138, 152, 154, 157, 159, 218, 219, 238, 239, 628, 630, 632, 633, 641, 644, 1330	\int_gincr:N	178, 240, 286, 293, 336, 340, 344, 348, 354, 358, 362, 366, 512, 979, 1115, 1132, 1146, 1157
\exp_last_unbraced:NV	551	\int_gset:Nn	7, 82, 158
\exp_not:n	186, 205	\int_if_zero:nTF	171, 172, 196, 197, 639, 647
F			
file commands:		\int_incr:N	93, 163, 187, 442
\file_if_exist:nTF	322	\int_new:N	6, 76, 78, 81, 96, 155, 159, 315, 316, 317, 318, 971
\file_input:n	356	\int_rand:n	61, 62, 64, 66, 68, 70, 71
firstkid (key)	1, 758	\int_set:Nn	260, 263, 266, 267, 268
flag commands:		\int_step_inline:nn	473, 479
\flag_clear:n	237	\int_step_inline:nnn	25, 91, 248
\flag_height:n	137, 249	\int_step_inline:nnnn	149, 174, 177, 200, 483, 489
\flag_new:n	135	\int_to_arabic:n	166, 168
\flag_raise:n	250	\int_to_Hex:n	61, 62, 64, 66, 68, 70, 71
\fontencoding	6	\int_use:N	11, 15, 20, 26, 35, 40, 48, 49, 54, 55, 58, 60, 66, 72, 75, 89, 104, 124, 131, 133, 144, 162, 179, 186, 205, 225, 234, 241, 243, 291, 293, 344, 378, 424, 435, 455, 456, 464, 465, 510, 527, 586, 763, 821, 833,
\fontfamily	6		
\fontseries	6		
\fontshape	6		
\fontsize	6		
G			
group commands:			
\group_begin:	67, 76, 173, 309, 978, 1070, 1078, 1113, 1130, 1156		
\group_end:	74, 79, 213, 348, 996, 1074, 1084, 1123, 1141, 1299		
H			
\halign	47		
hbox commands:			
\hbox_set:Nn	169, 170		

847, 863, 878, 886, 955, 966, 982,
985, 988, 1029, 1031, 1048, 1050,
1119, 1122, 1136, 1140, 1165, 1171,
1176, 1183, 1186, 1210, 1225, 1238,
1248, 1261, 1264, 1547, 1620, 1671
`\int_zero:N` 90, 105, 430
intarray commands:
`\intarray_gset:Nnn`
..... 397, 406, 421, 425, 461
`\intarray_item:Nn` 463, 466, 554
`\intarray_new:Nn` 390, 453
`interwordspace_`(deprecated) 209, 6
ior commands:
`\ior_close:N` 330, 467
`\ior_map_inline:Nn` 326, 438
`\ior_open:Nn` 324, 433, 436
`\g_tmpa_ior`
..... 324, 326, 330, 433, 436, 438, 467
iow commands:
`\iow_newline:` 203, 303
`\iow_term:n` 196, 200, 203, 209, 213,
264, 355, 359, 363, 367, 371, 375, 379

K

kernel internal commands:
`\__kernel_pdffdict_name:n` 45
`\g__kernel_pdfmanagement_end_-
run_code_tl` 880
keys commands:
`\keys_define:nn`
..... 8, 32, 34, 40, 52, 120, 132,
182, 192, 195, 235, 238, 242, 244,
280, 380, 383, 397, 412, 476, 556,
625, 752, 758, 760, 891, 948, 1022,
1087, 1109, 1126, 1578, 1593, 1627
`\keys_set:nn` 10,
18, 18, 19, 117, 187, 190, 285, 316,
319, 338, 342, 426, 886, 1120, 1181
`\keys_set_known:nnnN` 765

L

`label (key)` 1, 758
`\label` 12
`label_`(mc-key) 83, 238, 382
`lang (key)` 1, 758
`lang_`(mc-key=) 238
`\llap` 424
`log (deprecated) (key)` 259
ltx. internal commands:
`ltx.__tag.func.alloctag` 314
`ltx.__tag.func.check_parent_-
child_rules` 987
`ltx.__tag.func.fakespace` 493

`ltx.__tag.func.fill_parent_tree_-
line` 874
`ltx.__tag.func.get_num_from` 323
`ltx.__tag.func.get_tag_from` 342
`ltx.__tag.func.mark_page_-
elements` 700
`ltx.__tag.func.mark_shipout` 857
`ltx.__tag.func.markspaceoff` 564
`ltx.__tag.func.markspaceon` 564
`ltx.__tag.func.mc_insert_kids` .. 637
`ltx.__tag.func.mc_num_of_kids` .. 372
`ltx.__tag.func.output_num_from` . 323
`ltx.__tag.func.output_parenttree` 874
`ltx.__tag.func.output_tag_from` . 342
`ltx.__tag.func.role_get_parent_-
child_rule` 980
`ltx.__tag.func.space_chars_-
shipout` 597
`ltx.__tag.func.store_mc_data` ... 357
`ltx.__tag.func.store_mc_in_page` 681
`ltx.__tag.func.store_mc_kid` 366
`ltx.__tag.func.store_mc_label` .. 362
`ltx.__tag.func.store_struct_-
mcabs` 669
`ltx.__tag.func.update_mc_-
attributes` 689
`ltx.__tag.tables.role_tag_-
attribute` 312
`ltx.__tag.trace.log` 226
`ltx.__tag.trace.show_all_mc_data` 283
`ltx.__tag.trace.show_mc_data` ... 268
`ltx.__tag.trace.show_prop` 243
`ltx.__tag.trace.show_seq` 234
`ltx.__tag.trace.show_struct_data` 289
lua commands:
`\lua_escape:n` 32
`\lua_now:n` 8, 12, 15, 18,
25, 26, 27, 32, 35, 38, 42, 43, 49, 50,
54, 58, 59, 61, 62, 63, 67, 68, 69, 69,
73, 77, 86, 87, 87, 89, 96, 101, 109,
111, 120, 126, 133, 138, 141, 150,
158, 162, 181, 189, 230, 237, 244,
252, 268, 282, 283, 298, 303, 313,
317, 327, 328, 399, 408, 497, 510, 675

M

`\MakeLinkTarget` 114, 152
`mathml (key)` 1, 971
`\maxdimen` 190
`mc-current` 23, 16
`mc-current_`(show-key) 45, 132
`mc-data_`(show-key) 45, 120
`mc-label-unknown` 23, 9
`mc-marks_`(show-key) 45, 192

`\pdf_object_ref_indexed:nn` 57, 74, 96, 127, 211, 267, 283, 426, 447, 508, 536, 1437
`\pdf_object_ref_last:` 116, 104, 118, 124, 306, 1513, 1519, 1656
`\pdf_object_unnamed_write:nn` 100, 111, 120, 248, 298, 1505, 1651
`\pdf_object_write:nnn` 257, 281, 311, 330, 337, 342
`\pdf_object_write_indexed:nnnn` 139, 461
`\pdf_pageobject_ref:n` 233, 498, 526
`\pdf_string_from_unicode:nnN` 42, 819
`\pdf_uncompress:` 269, 271
`\pdf_version:` 241, 244, 246
`\pdf_version_compare:NnTF` 20, 81, 148, 154, 171, 227, 257, 324, 351, 431, 471, 590, 797, 817
`\pdf_version_gset:n` 247
pdfannot commands:
`\pdfannot_dict_put:nnn` 99, 757, 781, 805, 829, 852
`\pdfannot_link_ref_last:` 771, 795, 819, 843, 866
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` 1071, 1072, 1079, 1080, 1081, 1114, 1131
`\pdfdict_use:n` 283, 332, 339
`\pdffakespace` 45, 303
pdffile commands:
`\pdffile_embed_file:nnn` 107, 1116, 1133
`\pdffile_embed_stream:nnN` 972, 980
`\pdffile_embed_stream:nnn` 100
`\pdfglyphptounicode` 30
`\pdfinterwordspaceoff` 211, 118
`\pdfinterwordspaceon` 211, 34, 122
pdfmanagement commands:
`\pdfmanagement_add:nnn` 52, 70, 71, 344, 346, 348, 393, 1137
`\pdfmanagement_remove:nn` 350
phoneme (key) 758
prg commands:
`\prg_do_nothing:` 39, 80, 91, 106, 371, 372, 373, 374, 454, 575, 576, 577, 578
`\prg_generate_conditional-variant:Nnn` 97
`\prg_new_conditional:Nnn` 68, 224
`\prg_new_conditional:Npnn` 255, 289, 312, 322, 521, 527, 538
`\prg_new_eq_conditional:NNn` 82, 231
`\prg_new_protected_conditional:Nnn` 832
`\prg_new_protected_conditional:Npnn` 304, 719
`\prg_replicate:nn` 165
`\prg_return_false:` 78, 228, 256, 274, 285, 295, 298, 308, 319, 329, 524, 536, 542, 749, 834
`\prg_return_true:` 79, 227, 271, 282, 294, 307, 316, 326, 525, 535, 541, 719, 750, 834
`\prg_set_conditional:Npnn` 260, 278
`\prg_set_protected_conditional:Npnn` 721
process commands:
`process_softhyphen_pre` process_softhyphen_post 927
\ProcessOptions 39
prop commands:
`\prop_clear:N` 176
`\prop_count:N` 203
`\prop_gclear:N` 897
`\prop_get:NnN` 127, 144, 145, 177, 198, 214, 268, 299, 456, 468, 470, 483, 484, 485, 546, 547, 621, 622, 929
`\prop_get:NnNTF` 44, 94, 130, 137, 144, 158, 183, 193, 205, 213, 236, 295, 334, 344, 364, 379, 395, 398, 419, 444, 445, 514, 559, 564, 599, 604, 609, 614, 650, 677, 721, 737, 780, 785, 788, 834, 901, 914, 927, 1232, 1331, 1397, 1464, 1506, 1605, 1645, 1649
`\prop_gput:Nnn` 24, 26, 27, 28, 31, 56, 88, 90, 91, 96, 97, 97, 99, 99, 100, 101, 101, 110, 111, 112, 113, 119, 121, 122, 142, 145, 269, 270, 286, 291, 383, 446, 448, 448, 449, 469, 475, 481, 488, 490, 491, 762, 898, 900, 1281, 1292, 1368, 1413, 1563, 1568, 1609, 1656
`\prop_gremove:Nn` 137, 159, 901
`\prop_gset_eq:NN` 158, 1278
`\prop_gset_from_keyval:Nn` 871
`\prop_if_exist:NnTF` 174, 209, 234, 280, 320, 442, 648, 675, 1353, 1394
`\prop_if_exist_p:N` 773
`\prop_item:Nn` 41, 99, 102, 103, 109, 115, 146, 256, 545, 1289, 1654, 1661
`\prop_map_function:NN` 254
`\prop_map_inline:Nn` 267, 272, 293, 326, 363, 381, 410, 486, 495, 884
`\prop_map_tokens:Nn` 344

\prop_new:N	8, 9, 10, 11, 11, 25, 33, 72, 139, 156, 870, 1159, 1556, 1559
\prop_new_linked:N	7, 17, 86, 91, 103, 140, 1557
\prop_put:Nnn	102, 188
\prop_show:N	67, 93, 148, 1275, 1296, 1623, 1650
property commands:	
\property_new:nnnn	122, 125, 129, 132, 136
\property_record:nn	59, 111
\property_ref:nn	115, 116
\property_ref:nnn	42, 115, 120, 181, 190, 233, 234, 365, 499, 500, 511, 1354, 1358
\providecommand	310
\ProvidesExplFile	3
\ProvidesExplPackage	3, 3, 3, 3, 3, 3, 3, 3, 7, 7, 20, 31, 1552
Q	
\quad	222, 223
quark commands:	
\q_no_value	606, 616, 662, 667, 689, 694
\quark_if_no_value:NTF	132, 178, 199, 215, 269, 300, 627, 638
\quark_if_no_value_p:N	549, 550, 620, 621, 698, 699
\q_stop	259, 292, 328
R	
raw_(mc-key)	83, 238, 382
ref (key)	1, 758, 948
\RemoveFromHook	35, 341
\renewcommand	472, 473
\RenewDocumentCommand	8
\RequirePackage	40, 362, 365, 371, 374
\rlap	435
role_(rolemap-key)	186, 752
role commands:	
role_get_parent_child_rule	980
role-MC-child-forbidden	104
role-missing	24, 86
role-namespace_(rolemap-key)	186, 752
role-parent-child-check	90
role-parent-child-forbidden	111
role-parent-child-result	24, 92
role-parent-child-unresolved	164
role-remapping	24, 213
role-struct-parent-child-forbidden	94
role-tag	24, 215
role-unknown	24, 86
role-unknown-NS	24, 86
role-unknown-tag	24, 86
role/new-attribute_(setup-key)	117, 1561
role/new-attribute*(setup-key)	117, 1561
role/new-tag_(setup-key)	752
root-AF (key)	1, 1087
root-supplemental-file (key)	1109
S	
\selectfont	6
seq commands:	
\seq_clear:N	331, 488
\seq_const_from_clist:Nn	41, 54
\seq_count:N	22, 25, 58, 343, 456, 1613, 1615, 1617, 1641, 1667
\seq_get:NN	550, 555, 723, 736
\seq_get:NNTF	481, 624, 1189, 1319, 1327
\seq_gpop:NN	1311, 1312
\seq_gpop:NNTF	106, 1313
\seq_gpop_left:NN	319
\seq_gpush:Nn	13, 16, 17, 89, 96, 1196, 1202, 1204
\seq_gput_left:Nn	42, 144, 285, 323, 1570
\seq_gput_right:Nn	37, 143, 146, 152, 248, 269, 308, 508
\seq_gset_eq:NN	157, 219, 338
\seq_if_empty:NTF	198, 450
\seq_if_in:NnTF	306
\seq_item:Nn	59, 114, 116, 123, 127, 134, 138, 145, 360, 367, 380, 531, 533, 540, 730, 731, 746, 747, 775, 776, 797, 798
\seq_log:N	173, 197, 238, 434, 595, 610
\seq_map_function:NN	263
\seq_map_indexed_inline:Nn	446, 459
\seq_map_inline:Nn	289, 332, 1603, 1643
\seq_map_pairwise_function:NNN	1454
\seq_new:N	12, 12, 13, 14, 14, 15, 17, 17, 18, 19, 20, 24, 73, 74, 141, 157, 1161, 1560
\seq_pop_left:NN	455, 457, 458
\seq_put_right:Nn	333
\seq_remove_all:Nn	336, 566
\seq_set_eq:NN	205, 206
\seq_set_from_clist:NN	1598, 1634
\seq_set_from_clist:Nn	85, 88, 194, 214, 443, 454
\seq_set_map:NNn	567
\seq_set_map_e:NNn	1599, 1635
\seq_set_split:Nnn	51, 104, 565, 723, 727, 739, 743, 774, 790, 794
\seq_show:N	60, 147, 205, 206, 239, 334, 335, 337, 518, 1208, 1276, 1297, 1307

\seq_use:Nn	50,	str commands:	
108, 109, 203, 222, 223, 395, 568, 1614		\str_case:nnTF	46, 679, 1215
Setup keys:		\str_const:Nn	59
activate-all (deprecated)	<u>1</u>	\str_if_eq:nnTF	117, 125, 540, 626, 691
activate-mc (deprecated)	<u>1</u>	\str_if_eq_p:nn	310, 531, 533
activate-struct (deprecated)	<u>1</u>	\str_new:N	71
activate-tree (deprecated)	<u>1</u>	\str_remove_once:Nn	563, 564
activate/all	<u>1</u> , <u>241</u>	\str_set_convert:Nnnn	105, 261, 296,
activate/collect-link-artifacts		396, 413, 815, 827, 841, 857, 872, 960	
.....	<u>1</u> , <u>339</u>	\str_use:N	272, 309
activate/mc	<u>1</u> , <u>241</u>	\c_tilde_str	57, 59
activate/softhyphen	<u>1</u> , <u>275</u>	\string	15, 16
activate/spaces	<u>1</u>	struct-faulty-nesting	<u>23</u> , <u>32</u>
activate/struct	<u>1</u> , <u>241</u>	struct-label-unknown	<u>23</u> , <u>38</u>
activate/struct-dest	<u>1</u> , <u>241</u>	struct-missing-tag	<u>23</u> , <u>35</u>
activate/tagunmarked	<u>1</u> , <u>272</u>	struct-no-objnum	<u>23</u> , <u>24</u>
activate/tree	<u>1</u> , <u>241</u>	struct-orphan	<u>23</u> , <u>25</u>
catalog-supplemental-file	<u>1126</u>	struct-Ref-unknown	<u>42</u>
debug/log	<u>1</u> , <u>259</u>	struct-show-closing	<u>23</u> , <u>40</u>
debug/show	<u>258</u>	struct-stack_(show-key)	<u>45</u> , <u>235</u>
debug/uncompress	<u>259</u>	struct-unknown	<u>23</u> , <u>22</u>
log (deprecated)	<u>259</u>	struct-used-twice	<u>23</u> , <u>36</u>
no-struct-dest (deprecated)	<u>1</u>	Structure keys:	
page/tabsorder	<u>1</u> , <u>342</u>	actualtext	<u>1</u> , <u>758</u>
root-AF	<u>1</u> , <u>1087</u>	AF	<u>1</u> , <u>971</u>
root-supplemental-file	<u>1109</u>	AFinline	<u>1</u> , <u>971</u>
tabsorder (deprecated)	<u>1</u> , <u>342</u>	AFinline-o	<u>1</u> , <u>971</u>
tagunmarked (deprecated)	<u>1</u> , <u>272</u>	AFref	<u>1</u> , <u>971</u>
uncompress (deprecated)	<u>259</u>	alt	<u>1</u> , <u>758</u>
shipout commands:		attribute	<u>1</u> , <u>1627</u>
\g_shipout_readonly_int		attribute-class	<u>1</u> , <u>1593</u>
.....	<u>131</u> , <u>131</u> , <u>241</u> , <u>396</u> , <u>527</u>	E	<u>1</u> , <u>758</u> , <u>948</u>
show-kids	<u>24</u> , <u>64</u>	firstkid	<u>1</u> , <u>758</u>
show-spaces_(deprecated)	<u>209</u> , <u>6</u>	label	<u>1</u> , <u>758</u>
show-struct	<u>24</u> , <u>64</u>	lang	<u>1</u> , <u>758</u>
\ShowTagging	<u>21</u> , <u>45</u> , <u>114</u>	mathml	<u>1</u> , <u>971</u>
skip commands:		parent	<u>1</u> , <u>758</u>
\skip_horizontal:n	<u>78</u>	phoneme	<u>758</u>
\c_zero_skip	<u>78</u>	ref	<u>1</u> , <u>758</u> , <u>948</u>
socket commands:		stash	<u>1</u> , <u>758</u>
\socket_assign_plug:nn		tag	<u>1</u> , <u>758</u>
200, 204, 205, 209, 210, 757, 813, 814		texsource	<u>1</u> , <u>971</u>
\socket_if_exist:nTF	<u>645</u>	title	<u>1</u> , <u>758</u>
\socket_new:nn	<u>183</u> , <u>184</u> , <u>718</u>	title-o	<u>1</u> , <u>758</u>
\socket_new_plug:nnn ...	<u>185</u> , <u>719</u> , <u>735</u>	\SuspendTagging	<u>47</u>
\socket_use:n	<u>28</u> , <u>65</u>	sys commands:	
\socket_use:nn		\c_sys_backend_str	<u>46</u>
..	<u>70</u> , <u>205</u> , <u>339</u> , <u>811</u> , <u>1257</u> , <u>1375</u> , <u>1420</u>	\c_sys_engine_str	<u>12</u> , <u>14</u>
\socket_use:nnn	<u>75</u>	\sys_if_engine luatex:TF ..	<u>23</u> , <u>36</u> ,
\socket_use:nw	<u>86</u>	50, 72, 83, 85, 105, 187, 222, 281,	
\socket_use_expandable:n	<u>81</u>	296, 311, 326, 354, 355, 391, 493, 501	
\socket_use_expandable:nw	<u>101</u>	\sys_if_engine luatex_p:	<u>671</u>
stash (key)	<u>1</u> , <u>758</u>	\sys_if_engine pdftex:TF ...	<u>26</u> , <u>112</u>
stash_(mc-key)	<u>84</u> , <u>182</u>	\sys_if_output_pdf:TF	<u>11</u> , <u>28</u> , <u>114</u>

sys-no-interwordspace [24](#), [228](#)

T

tabsorder (deprecated) (key) [1](#), [342](#)

tag (key) [1](#), [758](#)

tag_(mc-key) [83](#), [238](#), [382](#)

tag_(rolemap-key) [186](#), [752](#)

tag commands:

\tag_attr_new:nn [117](#), [1574](#), [1576](#)

\tag_attr_new_entry:nn [1561](#)

\tag_check_benchmark_on: [657](#)

\tag_check_child:nn [719](#), [721](#)

\tag_check_child:nnTF [186](#), [719](#)

\tag_get:n [20](#), [84](#), [113](#),
[115](#), [132](#), [133](#), [89](#), [92](#), [235](#), [235](#), [579](#), [611](#)

\tag_get:nN [20](#), [21](#), [132](#), [236](#), [236](#)

\tag_get:nnN [21](#), [133](#), [238](#), [238](#)

\tag_if_active: [255](#), [260](#)

\tag_if_active:TF [20](#), [18](#), [252](#), [253](#), [446](#)

\tag_if_active_p: [20](#), [252](#), [882](#)

\tag_if_box_tagged:N [289](#)

\tag_if_box_tagged:NTF [21](#), [288](#)

\tag_if_box_tagged_p:N [21](#), [288](#)

\tag_if_in:n [21](#), [303](#), [304](#)

\tag_if_in:nTF [21](#)

\tag_if_in_p:n [21](#)

\tag_if_struct_exist:n [20](#), [278](#)

\tag_if_struct_exist:nTF ... [20](#), [278](#)

\tag_if_struct_exist_p:n ... [20](#), [278](#)

\tag_if_tag_known:n [832](#)

\tag_if_tag_known:nTF [186](#), [832](#)

\tag_mc_add_missing_to_stream:Nn
..... [83](#), [66](#), [187](#), [223](#)

\tag_mc_artifact_group_begin:n .
..... [82](#), [60](#), [60](#), [63](#)

\tag_mc_artifact_group_end:
..... [82](#), [60](#), [61](#), [71](#)

\tag_mc_begin:n [11](#),
[82](#), [25](#), [66](#), [114](#), [169](#), [169](#), [293](#), [293](#),
[297](#), [303](#), [423](#), [434](#), [511](#), [539](#), [589](#), [655](#)

\tag_mc_begin_pop:n [82](#),
[76](#), [80](#), [81](#), [102](#), [520](#), [550](#), [619](#), [665](#)

\tag_mc_end:
[82](#), [31](#), [75](#), [93](#), [216](#), [216](#), [293](#), [294](#),
[357](#), [363](#), [425](#), [436](#), [517](#), [546](#), [594](#), [663](#)

\tag_mc_end_push:
..... [82](#), [65](#), [80](#), [80](#), [83](#), [505](#), [532](#), [605](#), [653](#)

\tag_mc_if_in: [82](#), [231](#)

\tag_mc_if_in:TF [82](#), [42](#), [68](#), [224](#)

\tag_mc_if_in_p: [82](#), [68](#), [224](#)

\tag_mc_new_stream:n [83](#), [15](#), [15](#), [67](#), [67](#)

\tag_mc_reset_box:N [82](#), [79](#), [79](#), [228](#), [228](#)

\tag_mc_use:n [82](#), [36](#), [36](#), [36](#), [38](#)

\c_tag_para_semantic_tl ... [311](#), [466](#)

\c_tag_para_textblock_tl .. [311](#), [457](#)

\tag_resume:n
... [7](#), [73](#), [158](#), [194](#), [207](#), [217](#), [516](#), [545](#)

\tag_socket_use:n
..... [47](#), [48](#), [61](#), [62](#), [441](#), [442](#)

\tag_socket_use:nn ... [47](#), [48](#), [61](#), [67](#)

\tag_socket_use:nnn ... [47](#), [48](#), [61](#), [72](#)

\tag_socket_use_expandable:n ...
..... [47](#), [48](#), [61](#), [78](#)

\tag_spacechar_off: ... [82](#), [82](#), [87](#), [116](#)

\tag_spacechar_on: ... [82](#), [83](#), [99](#), [120](#)

\tag_start: [7](#), [158](#), [169](#), [182](#), [211](#)

\tag_start:n ... [7](#), [158](#), [207](#), [215](#), [217](#)

\tag_stop: ... [7](#), [56](#), [158](#), [160](#), [181](#), [210](#)

\tag_stop:n ... [7](#), [158](#), [193](#), [214](#), [216](#)

\tag_struct_begin:n . [113](#), [48](#), [538](#),
[588](#), [611](#), [654](#), [1146](#), [1146](#), [1150](#), [1151](#)

\tag_struct_end:
..... [113](#), [26](#), [53](#), [547](#), [595](#),
[616](#), [664](#), [1146](#), [1147](#), [1303](#), [1304](#), [1344](#)

\tag_struct_end:n ... [113](#), [1148](#), [1341](#)

\tag_struct_gput:nnn
..... [114](#), [954](#), [1441](#), [1441](#), [1443](#), [1451](#)

\tag_struct_gput_ref:nnn [114](#)

\tag_struct_insert_annot:nn
..... [113](#), [155](#), [770](#),
[794](#), [818](#), [842](#), [865](#), [1537](#), [1537](#), [1546](#)

\tag_struct_map_function:N
..... [114](#), [1452](#), [1452](#)

\tag_struct_object_ref:n ... [113](#),
[905](#), [918](#), [933](#), [944](#), [1434](#), [1435](#), [1439](#)

\tag_struct_parent_int:
..... [113](#), [155](#), [760](#), [771](#), [784](#), [795](#),
[808](#), [819](#), [832](#), [843](#), [855](#), [866](#), [1537](#), [1547](#)

\tag_struct_recordtarget: ... [93](#), [93](#)

\tag_struct_use:n
..... [113](#), [115](#), [58](#), [1347](#), [1347](#), [1349](#)

\tag_struct_use_num:n
..... [113](#), [233](#), [1388](#), [1388](#), [1390](#), [1432](#)

\tag_suspend:n
... [7](#), [68](#), [158](#), [183](#), [193](#), [216](#), [512](#), [540](#)

\tag_tool:n [13](#), [13](#), [14](#), [16](#), [20](#)

tag internal commands:

__tag_activate_linkbin: .. [220](#), [239](#)

__tag_activate_mark_space [564](#)

\g__tag_active_mc_bool
..... [40](#), [82](#), [244](#), [251](#), [265](#), [314](#)

\l__tag_active_mc_bool
..... [88](#), [165](#), [175](#), [189](#), [200](#), [268](#), [314](#)

\l__tag_active_socket_bool
..... [64](#), [69](#), [74](#),
[80](#), [85](#), [88](#), [100](#), [166](#), [176](#), [190](#), [201](#), [282](#)

\g__tag_active_space_bool
..... [13](#), [57](#), [62](#), [82](#)

\g__tag_active_struct_bool	__tag_check_if_mc_tmb_missing_-
..... 82 , 246 , 253 , 264 , 296 , 324 , 478	p: 527
\l__tag_active_struct_bool	__tag_check_if_mc_tme_missing: 538
..... 88 , 164 , 174 , 188 , 199 , 267 , 324	__tag_check_if_mc_tme_missing:TF
\g__tag_active_struct_dest_bool	 153 , 211 , 228 , 538
..... 82 , 250 , 257 , 295	__tag_check_if_mc_tme_missing_-
\g__tag_active_tree_bool	p: 538
.... 9 , 68 , 82 , 245 , 252 , 266 , 351 , 389	__tag_check_info_closing_-
__tag_add_missing_mcs:Nn	struct:n 349 , 349 , 357 , 1315
..... 97 , 165 , 165 , 217	__tag_check_init_mc_used:
__tag_add_missing_mcs_to_-	 451 , 451 , 454 , 460
stream:Nn . 65 , 65 , 66 , 187 , 187 , 223	__tag_check_mc_if_nested:
\g__tag_attr_class_used_prop	 174 , 310 , 410 , 410
..... 291 , 293 , 1555 , 1609	__tag_check_mc_if_open:
\g__tag_attr_class_used_seq	 220 , 369 , 410 , 418
..... 289 , 1560 , 1570	__tag_check_mc_in_galley:TF . . . 521
\g__tag_attr_entries_prop	__tag_check_mc_in_galley_p: . . . 521
..... 295 , 1555 ,	__tag_check_mc_pushed_popped:nn
1563 , 1568 , 1605 , 1645 , 1650 , 1654	 90 , 97 , 110 , 113 , 118 , 425 , 425
__tag_attr_new_entry:nn	__tag_check_mc_tag:N
526 , 1561 , 1561 , 1574 , 1575 , 1582 , 1590	 193 , 328 , 437 , 437
__tag_attr_new_used_entry:nn	__tag_check_mc_used:n
..... 1566 , 1586	 145 , 266 , 456 , 456
\g__tag_attr_objref_prop	\g__tag_check_mc_used_intarray .
..... 1555 , 1649 , 1656 , 1661	 451 , 461 , 463 , 466
\l__tag_attr_value_tl 1555 ,	__tag_check_no_open_struct:
1639 , 1658 , 1663 , 1665 , 1669 , 1673	 358 , 358 , 1317 , 1325
__tag_backend_create_bdc_node . . 438	__tag_check_para_begin_show:nn 418
__tag_backend_create_bmc_node . . 409	__tag_check_para_end_show:nn . . 429
__tag_backend_create_emc_node . . 380	\c__tag_check_pdfversion_tl
__tag_check_add_tag_role:nn	 241 , 244 , 246 , 247
..... 129 , 372 , 372	__tag_check_show_MCID_by_page:
__tag_check_add_tag_role:nnn	 475 , 475
..... 169 , 391	__tag_check_struct_forbidden_-
__tag_check_benchmark_tic: . 356 ,	parent_child:nnn . . . 137 , 163 , 664
360 , 364 , 368 , 372 , 376 , 380 , 655 , 661	__tag_check_struct_used:n
__tag_check_benchmark_toc: . 358 ,	 362 , 362 , 1356
362 , 366 , 370 , 374 , 378 , 382 , 656 , 662	__tag_check_structure_has_tag:n
__tag_check_forbidden_parent_-	 332 , 332 , 1186
child:nnnn 120 , 120 , 134 , 171	__tag_check_structure_tag:N
__tag_check_if_active_mc: 312	 342 , 342 , 732 , 755 , 806
__tag_check_if_active_mc:TF	__tag_check_typeof_v:n
..... 85 , 104 ,	 108 , 109 , 112 ,
171 , 189 , 218 , 299 , 305 , 311 , 359 , 365	147 , 155 , 162 , 200 , 209 , 230 , 230 , 264
__tag_check_if_active_struct: . 322	__tag_check_unresolved_parent_-
__tag_check_if_active_struct:TF	child:nnnn 169 , 169
..... 40 , 311 , 1153 , 1154 ,	\g__tag_css_bool . . 878 , 879 , 882 , 893
1308 , 1309 , 1343 , 1351 , 1392 , 1540	\g__tag_css_prop
__tag_check_if_mc_in_galley: . . 521	 870 , 871 , 884 , 897 , 898 , 900 , 901
__tag_check_if_mc_in_galley:TF	__tag_debug_mc_begin_ignore:n .
..... 198 , 219	 352 , 556
__tag_check_if_mc_tmb_missing: 527	__tag_debug_mc_begin_insert:n .
__tag_check_if_mc_tmb_missing:TF	 307 , 549
..... 110 , 207 , 224 , 527	__tag_debug_mc_end_ignore: 377 , 570

__tag_debug_mc_end_insert:	367, 563	153, 163, 166, 175, 601, 609, 622,
__tag_debug_struct_begin-		638, 642, 645, 702, 728, 731, 741, 744
ignore:n	598, 1301	\l__tag_get_parent_tmpc_tl
__tag_debug_struct_begin-		60, 144, 152, 154
insert:n	590, 1298	__tag_get_tag_from
__tag_debug_struct_end_check:n	620, 1343	\l__tag_get_tmpc_tl
__tag_debug_struct_end_ignore:	613, 1338	193, 198, 216, 218, 219, 236, 238,
__tag_debug_struct_end_insert:	605, 1336	239, 1235, 1241, 1467, 1469, 1473, 1479
__tag_exclude_headfoot_begin:	500, 561, 562	__tag_gincr_para_begin_int:
__tag_exclude_headfoot_end:	514, 563, 564	334, 338, 356, 372
__tag_exclude_struct_headfoot-		__tag_gincr_para_end_int:
begin:n	527, 568, 569	334, 346, 364, 374
__tag_exclude_struct_headfoot-		__tag_gincr_para_main_begin-
end:	543, 570, 571	int:
__tag_fakespace	493	334, 334, 352, 371
__tag_fakespace:	72, 74, 307	__tag_gincr_para_main_end_int:
__tag_finish_structure:	13, 16, 348, 349	334, 342, 360, 373
\l__tag_get_child_tmpa_tl	60, 605, 610, 677, 679, 689,	__tag_headfoot_tagged_begin:n
\l__tag_get_child_tmpb_tl	692, 703, 1357, 1361, 1363, 1369, 1379	598, 630, 631
\l__tag_get_child_tmpc_tl	60, 606, 611, 678, 690	__tag_headfoot_tagged_end:
__tag_get_data_mc_counter:	9, 9	614, 632, 633
__tag_get_data_mc_tag:	237, 237, 291, 291	__tag_hook_kernel_after_foot:
__tag_get_data_struct_C:nN	557, 557	483, 496, 564, 571, 578, 633
__tag_get_data_struct_counter:	583, 584	__tag_hook_kernel_after_head:
__tag_get_data_struct_id:	572, 572	481, 488, 563, 570, 577, 632
__tag_get_data_struct_num:	577, 578	__tag_hook_kernel_before_foot:
__tag_get_data_struct_num:N	553, 553	482, 494, 562, 569, 576, 631
__tag_get_data_struct_tag:	540, 540	__tag_hook_kernel_before_head:
__tag_get_data_struct_tag:N	548, 548	480, 486, 561, 568, 575, 630
__tag_get_mathsubtype	304	\g__tag_in_mc_bool
__tag_get_mc_abs_cnt:	14, 15, 19, 20, 102,	18, 175, 221, 226, 311, 370, 508,
126, 155, 166, 183, 210, 246, 254,		509, 523, 535, 536, 553, 608, 609, 622
272, 286, 307, 321, 331, 414, 422, 442		__tag_insert_bdc_node
__tag_get_mc_cnt_type_tag	298	438
__tag_get_num_from	323	__tag_insert_bmc_node
\l__tag_get_parent_tmpa_tl	60, 127, 132, 136, 139, 149, 152,	409
162, 165, 175, 600, 608, 621, 627,		__tag_insert_emc_node
631, 634, 701, 703, 727, 730, 740, 743		380
\l__tag_get_parent_tmpb_tl	60, 150,	__tag_log
		226
		\l__tag_loglevel_int
		81, 125, 132, 170, 171, 260,
		263, 266, 267, 268, 351, 381, 400,
		428, 431, 458, 512, 551, 558, 562,
		565, 572, 592, 600, 607, 615, 619, 622
		__tag_mark_spaces
		498
		__tag_mc_artifact_begin_marks:n
		21, 43, 79, 325
		\l__tag_mc_artifact_bool
		20, 176, 185, 196, 222, 321
		\l__tag_mc_artifact_type_tl
		19, 189, 193, 197,
		201, 205, 209, 213, 217, 323, 325, 344
		__tag_mc_bdc:nn
		232, 235, 281
		__tag_mc_bdc_mcid:n
		121, 237, 253
		__tag_mc_bdc_mcid:nn
		237, 238, 255, 260
		__tag_mc_bdc_shipout:nn
		236, 246

__tag_mc_begin_marks:nn		21, 21, 42, 78, 332
__tag_mc_bmc:n		232, 233, 277
__tag_mc_bmc_artifact:		275, 275, 288
__tag_mc_bmc_artifact:n		275, 279, 289
\l__tag_mc_botmarks_seq		97, 19, 88, 109, 159, 206, 206, 214, 219, 223, 523, 540
__tag_mc_check_parent_child:n	.	122, 122, 181, 207, 341
__tag_mc_disable_marks:		76, 76
__tag_mc_emc:		156, 232, 234, 372
__tag_mc_end_marks:	..	21, 61, 80, 373
\l__tag_mc_firstmarks_seq		96, 19, 85, 108, 194, 197, 198, 205, 205, 206, 222, 523, 531, 533
\g__tag_mc_footnote_marks_seq	...	12
__tag_mc_get_marks:	.	82, 82, 197, 218
__tag_mc_handle_artifact:N		117, 275, 283, 323
__tag_mc_handle_mc_label:n		27, 27, 200, 335
__tag_mc_handle_mcid:nn		237, 258, 263, 329
__tag_mc_handle_stash:n	50, 140, 142, 143, 168, 210, 264, 264, 274, 344	
__tag_mc_if_in:		68, 82, 224, 231
__tag_mc_if_in:TF	68, 87, 224, 412, 420	
__tag_mc_if_in:p:		68, 224
__tag_mc_insert_extra_tmb:n		106, 106, 169
__tag_mc_insert_extra_tme:n		106, 151, 170
__tag_mc_insert_mcid_kids:n		131, 131, 150, 321
__tag_mc_insert_mcid_single_kids:n		131, 136, 322
\l__tag_mc_key_label_tl		23, 198, 200, 316, 332, 333, 335, 422
\l__tag_mc_key_properties_tl		23, 177, 251, 266, 267, 281, 301, 302, 331, 392, 401, 402, 407, 418, 419
\l__tag_mc_key_stash_bool		20, 29, 38, 184, 203, 337
\g__tag_mc_key_tag_tl		19, 23, 180, 225, 237, 243, 291, 313, 371, 388
\l__tag_mc_key_tag_tl	23, 179, 193, 195, 224, 242, 312, 328, 330, 332, 387	
\l__tag_mc_lang_tl		22, 185, 190, 314, 319
__tag_mc_lua_set_mc_type_attr:n		83, 83, 107, 195
__tag_mc_lua_unset_mc_type_attr:		83, 109, 223
\g__tag_mc_main_marks_seq		12
\g__tag_mc_marks		11, 23, 32, 45, 52, 63, 69, 86, 89, 195, 215
\g__tag_mc_multicol_marks_seq	...	12
\g__tag_mc_parenttree_prop		17, 18, 101, 184, 270
__tag_mc_set_label_used:n	31, 31, 51	
\g__tag_mc_stack_seq		18, 89, 96, 106, 434
__tag_mc_store:nnn	..	91, 91, 105, 132
g__tag_MCID_abs_int		7
\g__tag_mode_lua_bool		35, 36, 124, 135, 248, 303, 329, 360, 369, 503, 518, 530, 548, 603, 617
\l__tag_name_link_tl	...	639, 641, 642
__tag_pairs_prop		243
\l__tag_para_attr_class_tl		311
\g__tag_para_begin_int		311, 340, 358, 424, 459, 464
\l__tag_para_bool		311, 382, 399, 414, 472, 473, 502, 529, 602
\g__tag_para_end_int		311, 348, 366, 435, 459, 465
\l__tag_para_flattened_bool		311, 395, 410
\l__tag_para_main_attr_class_tl	311	
\g__tag_para_main_begin_int		311, 336, 354, 450, 455
\g__tag_para_main_end_int		311, 344, 362, 450, 456
__tag_para_main_store_struct:		376, 376
\g__tag_para_main_struct_tl	311, 378	
\l__tag_para_main_tag_tl	311, 392, 407	
\l__tag_para_show_bool		311, 383, 384, 415, 421, 432
\l__tag_para_tag_default_tl		311
\l__tag_para_tag_tl	311, 387, 402, 416	
\l__tag_parent_child_check_tl		156, 157, 169, 172, 500, 654, 655, 662, 665, 733, 734, 746, 748
__tag_parenttree_add_objr:nn	..	163, 163, 503, 531
\l__tag_parenttree_content_tl	..	170, 195, 207, 227, 235, 256, 259
\g__tag_parenttree_objr_tl		162, 165, 256
__tag_pdf_name_e:n		117, 117
__tag_pdf_object_ref		468
__tag_prop_gput:Nnn		9, 29, 89, 98, 109, 111, 120, 121, 128, 132, 139, 142, 146, 149, 150, 201, 205, 217, 220, 282, 285, 314, 315, 384, 1362, 1510, 1517

__tag_prop_item:Nn ... [9](#), [52](#), [139](#), [146](#)
 __tag_prop_new:N [9](#), [9](#),
 [11](#), [19](#), [24](#), [32](#), [120](#), [139](#), [139](#), [153](#), [1158](#)
 __tag_prop_new_linked:N
 [15](#), [17](#), [139](#), [140](#)
 __tag_prop_show:N [9](#), [65](#), [139](#), [148](#), [156](#)
 \c__tag_property_mc_clist .. [79](#), [245](#)
 __tag_property_record:nn
 [29](#), [108](#), [108](#), [117](#), [241](#), [489](#), [764](#)
 __tag_property_ref_lastpage:nn
 . [83](#), [118](#), [118](#), [160](#), [174](#), [177](#), [479](#), [493](#)
 \c__tag_property_struct_clist [79](#), [766](#)
 g__tag_role/RoleMap_dict [18](#)
 \g__tag_role_add_mathml_bool ...
 [73](#), [265](#), [762](#), [829](#)
 __tag_role_add_tag:nn
 [127](#), [127](#), [153](#), [280](#), [365](#), [800](#)
 __tag_role_add_tag:nnnn
 [167](#), [167](#), [226](#), [312](#), [805](#)
 __tag_role_alloctag:nnn [81](#),
 [85](#), [95](#), [107](#), [117](#), [126](#), [141](#), [184](#), [277](#), [308](#)
 __tag_role_check_parent_-
 child:nnnnN [151](#),
 [164](#), [590](#), [592](#), [607](#), [640](#), [717](#), [729](#), [742](#)
 \l__tag_role_debug_prop [11](#)
 __tag_role_get:nnNN [154](#),
 [156](#), [164](#), [227](#), [229](#), [253](#), [748](#), [799](#), [1197](#)
 __tag_role_get_parent_child_-
 rule:nnN
 [202](#), [500](#), [503](#), [541](#), [589](#), [623](#), [701](#)
 \g__tag_role_index_prop
 [187](#), [10](#), [448](#), [456](#), [468](#),
 [469](#), [470](#), [475](#), [481](#), [483](#), [484](#), [485](#),
 [488](#), [490](#), [491](#), [495](#), [546](#), [547](#), [599](#), [609](#)
 \g__tag_role_NS<ns>_class_prop [187](#)
 \g__tag_role_NS<ns>_prop [187](#)
 \g__tag_role_NS_mathml_prop [267](#), [486](#)
 __tag_role_NS_new:nnn
 . [189](#), [20](#), [22](#), [30](#), [74](#), [75](#), [76](#), [77](#), [78](#), [80](#)
 \g__tag_role_NS_prop
 [187](#), [9](#), [26](#), [56](#), [193](#), [326](#), [344](#), [786](#)
 \g__tag_role_parent_child_-
 intarray [390](#), [397](#), [406](#), [421](#), [425](#), [555](#)
 __tag_role_read_namespace:n [337](#),
 [337](#), [341](#), [342](#), [343](#), [345](#), [347](#), [349](#), [350](#)
 __tag_role_read_namespace:nn ..
 [318](#), [318](#), [339](#), [348](#)
 __tag_role_read_namespace_-
 line:nw [255](#), [259](#), [292](#), [328](#)
 \l__tag_role_role_namespace_-
 tmpa_tl [12](#),
 [757](#), [778](#), [783](#), [787](#), [790](#), [794](#), [809](#)
 \l__tag_role_role_tmpa_tl
 [12](#), [756](#), [776](#), [782](#), [802](#), [808](#)
 \g__tag_role_rolemap_prop
 [187](#), [18](#), [144](#), [146](#), [149](#), [158](#),
 [214](#), [217](#), [220](#), [269](#), [272](#), [385](#), [604](#), [614](#)
 \c__tag_role_rule_checkparent_tl
 [157](#), [173](#), [655](#), [734](#)
 \c__tag_role_rules_num_prop
 [391](#), [514](#), [564](#)
 \c__tag_role_rules_prop [391](#), [395](#), [419](#)
 \l__tag_role_tag_namespace_tmpa_-
 tl [12](#), [755](#), [807](#)
 \l__tag_role_tag_namespace_tmpb_-
 tl [14](#)
 \l__tag_role_tag_namespace_tmpb_-
 tluuuuu% [12](#)
 \l__tag_role_tag_tmpa_tl
 [12](#), [754](#), [775](#), [801](#), [806](#)
 \g__tag_role_tags_class_prop ...
 [187](#), [8](#), [90](#), [99](#), [112](#), [121](#), [137](#), [268](#)
 \g__tag_role_tags_NS_prop .. [187](#),
 [7](#), [88](#), [97](#), [110](#), [119](#), [130](#), [344](#), [379](#),
 [383](#), [445](#), [721](#), [737](#), [781](#), [788](#), [834](#), [1331](#)
 \l__tag_role_tmpa_seq [12](#)
 \l__tag_role_update_bool
 [208](#), [255](#), [256](#), [264](#), [344](#), [346](#)
 \c__tag_role_userNS_id_str
 [188](#), [59](#), [80](#)
 \g__tag_root_default_tl [274](#)
 \g__tag_saved_in_mc_bool
 [499](#), [508](#), [523](#), [535](#), [553](#), [608](#), [622](#)
 __tag_seq_gput_left:Nn
 [9](#), [40](#), [144](#), [152](#), [280](#)
 __tag_seq_gput_right:Nn [9](#),
 [35](#), [139](#), [143](#), [151](#), [243](#), [253](#), [264](#), [303](#)
 __tag_seq_item:Nn ... [9](#), [47](#), [139](#), [145](#)
 __tag_seq_new:N
 [9](#), [9](#), [22](#), [121](#), [139](#), [141](#), [154](#), [1160](#)
 __tag_seq_show:N [9](#), [58](#), [139](#), [147](#), [155](#)
 __tag_show_spacemark [479](#)
 \l__tag_showspace_bool ... [7](#), [16](#), [17](#)
 \g__tag_softthyphen_bool
 [94](#), [276](#), [277](#), [280](#), [295](#), [310](#), [325](#)
 __tag_space_chars_shipout [597](#)
 __tag_start_para_ints:
 [177](#), [202](#), [350](#), [350](#)
 __tag_stop_para_ints:
 [167](#), [191](#), [350](#), [369](#)
 __tag_store_parent_child_-
 rule:nnn [391](#), [393](#), [417](#), [462](#)
 g__tag_struct_1_prop [119](#)
 __tag_struct_add_AF:nn
 ... [984](#), [1001](#), [1021](#), [1028](#), [1048](#), [1093](#)
 __tag_struct_add_inline_AF:nn .
 ... [973](#), [1000](#), [1062](#), [1066](#), [1073](#), [1083](#)
 \l__tag_struct_addkid_tl [88](#), [808](#), [1272](#)

\g__tag_struct_AFobj_int [971](#), [979](#), [982](#)
 __tag_struct_check_parent_-
 child:nn [614](#), [614](#), [669](#), [705](#), [714](#), [1259](#)
 __tag_struct_check_parent_-
 child_aux:nnnnN . [589](#), [590](#), [649](#), [657](#)
 \g__tag_struct_cont_mc_prop
 [11](#), [93](#), [94](#), [96](#), [99](#), [256](#)
 \g__tag_struct_dest_num_prop
 [90](#), [98](#), [914](#), [927](#)
 \l__tag_struct_elem_stash_bool .
 [87](#), [768](#), [1222](#), [1255](#), [1285](#)
 __tag_struct_exchange_kid_-
 command:N [317](#), [317](#), [326](#), [357](#)
 __tag_struct_fill_kid_key:n
 [136](#), [327](#), [327](#), [459](#)
 __tag_struct_format_P:nnN [421](#)
 __tag_struct_format_parentnum:nnN
 [424](#), [424](#)
 __tag_struct_format_parentrole:nnN
 [421](#), [422](#)
 __tag_struct_format_Ref [140](#)
 __tag_struct_format_Ref:nnN [428](#), [428](#)
 __tag_struct_format_rolemap:nnN
 [421](#), [421](#)
 __tag_struct_format_tag:nnN [421](#), [423](#)
 __tag_struct_get_dict_content:nN
 [138](#), [407](#), [407](#), [460](#)
 __tag_struct_get_id:n
 . [96](#), [101](#), [114](#), [115](#), [160](#), [161](#), [466](#), [574](#)
 __tag_struct_get_role:nnNN
 [146](#), [159](#), [207](#), [207](#),
 [226](#), [597](#), [602](#), [674](#), [686](#), [698](#), [724](#), [737](#)
 __tag_struct_gput_data_attribute:nn
 [1503](#), [1503](#)
 __tag_struct_gput_data_C:nn
 [1526](#), [1526](#)
 __tag_struct_gput_data_ref:nn .
 [1481](#), [1502](#)
 __tag_struct_gput_data_ref_-
 aux:nnn [1460](#),
 [1461](#), [1483](#), [1487](#), [1491](#), [1495](#), [1499](#)
 __tag_struct_gput_data_ref_-
 dest:nn [1489](#)
 __tag_struct_gput_data_ref_-
 dest_parent:nn [1493](#)
 __tag_struct_gput_data_ref_-
 label:nn [1485](#)
 __tag_struct_gput_data_ref_-
 num:nn [1497](#)
 __tag_struct_gput_data_tag:nn .
 [1522](#), [1522](#)
 __tag_struct_insert_annot:nn ..
 [474](#), [474](#), [1542](#)
 __tag_struct_insert_annot_-
 shipout:nnn [515](#), [515](#)
 __tag_struct_kid_mc_gput_-
 right:nn ... [227](#), [239](#), [240](#), [259](#), [267](#)
 __tag_struct_kid_OBJR_gput_-
 right:nnn [292](#), [292](#), [295](#), [316](#), [490](#), [518](#)
 __tag_struct_kid_struct_gput_-
 left:nn [276](#), [276](#), [277](#), [291](#)
 __tag_struct_kid_struct_gput_-
 right:nn
 [260](#), [260](#), [261](#), [275](#), [1359](#), [1404](#)
 g__tag_struct_kids_l_seq [119](#)
 \g__tag_struct_label_num_prop ..
 [86](#), [762](#), [901](#)
 \l__tag_struct_lang_tl
 [478](#), [1144](#), [1168](#), [1173](#)
 \c__tag_struct_link_bin_tl [225](#), [233](#)
 __tag_struct_mcid_dict:n
 [96](#), [99](#), [227](#), [246](#)
 \c__tag_struct_null_tl [10](#), [361](#)
 \g__tag_struct_objR_seq [8](#)
 \l__tag_struct_parenttag_NS_tl .
 [78](#), [798](#), [801](#), [805](#), [1228](#)
 \l__tag_struct_parenttag_tl
 [78](#), [797](#), [800](#), [804](#), [806](#), [1228](#)
 __tag_struct_prop_gput:nnn
 [105](#), [106](#), [107](#), [113](#), [123](#),
 [128](#), [133](#), [138](#), [143](#), [150](#), [176](#), [180](#),
 [189](#), [195](#), [200](#), [363](#), [376](#), [390](#), [820](#),
 [832](#), [846](#), [862](#), [877](#), [885](#), [965](#), [987](#),
 [1030](#), [1049](#), [1094](#), [1164](#), [1170](#), [1175](#),
 [1209](#), [1224](#), [1237](#), [1247](#), [1263](#), [1407](#),
 [1476](#), [1524](#), [1531](#), [1534](#), [1619](#), [1670](#)
 \g__tag_struct_ref_by_dest_prop [103](#)
 __tag_struct_Ref_dest:nN . [891](#), [912](#)
 __tag_struct_Ref_dest_parent:nN
 [891](#), [925](#)
 __tag_struct_Ref_label:nN [891](#), [899](#)
 __tag_struct_Ref_num:nN .. [891](#), [940](#)
 __tag_struct_Ref_obj:nN .. [891](#), [891](#)
 \g__tag_struct_roletag_NS_tl [78](#)
 \l__tag_struct_roletag_NS_tl ...
 [81](#), [1201](#), [1213](#), [1251](#)
 \g__tag_struct_roletag_stack_seq
 [15](#), [17](#), [306](#), [1204](#), [1311](#)
 \l__tag_struct_roletag_tl
 [78](#), [1200](#), [1203](#), [1205](#), [1213](#), [1215](#), [1251](#)
 __tag_struct_set_attribute: ...
 [25](#), [39](#), [1207](#), [1322](#)
 __tag_struct_set_tag_info:nnn .
 [171](#), [173](#), [187](#), [206](#), [1182](#)
 \g__tag_struct_stack_current_tl
 [18](#),
 [27](#), [33](#), [36](#), [67](#), [73](#), [100](#), [116](#), [148](#),

154, 162, 208, 268, 272, 299, 342,
 545, 574, 580, 1206, 1270, 1274,
 1275, 1296, 1315, 1321, 1360, 1366,
 1372, 1378, 1405, 1411, 1417, 1423
 \l__tag_struct_stack_parent_
 tmpa_tl .. 18, 483, 492, 509, 778,
 1180, 1187, 1191, 1233, 1260, 1267,
 1271, 1273, 1276, 1288, 1289, 1297
 \g__tag_struct_stack_seq 12,
 22, 25, 482, 555, 723, 736, 1190,
 1196, 1208, 1307, 1313, 1319, 1456
 \c__tag_struct_StructElem_
 entries_seq 41
 \c__tag_struct_StructTreeRoot_
 entries_seq 41
 \g__tag_struct_tag_NS_tl 78, 731,
 747, 750, 754, 1185, 1199, 1295, 1333
 \g__tag_struct_tag_stack_seq ...
 14, 50, 238, 239, 550,
 595, 610, 624, 1202, 1312, 1327, 1455
 \g__tag_struct_tag_tl 78,
 179, 180, 183, 312, 313, 441, 442,
 730, 732, 746, 749, 753, 755, 1184,
 1198, 1203, 1329, 1331, 1373, 1418
 __tag_struct_use_check_parent_
 child:nn . 670, 670, 717, 1377, 1422
 __tag_struct_write_obj 140
 __tag_struct_write_obj:n
 151, 440, 440
 \l__tag_tag_stop_int 158, 162, 163,
 171, 172, 179, 186, 187, 196, 197, 205
 \g__tag_tagunmarked_bool 93, 272, 274
 \l__tag_tmp_unused_tl 63,
 130, 337, 344, 395, 398, 402, 419,
 422, 445, 561, 563, 564, 565, 721,
 724, 737, 740, 788, 788, 791, 834, 1645
 \l__tag_tmp_unused_tl_uuuu\l__-
 tag_get_tmpe_tl 60
 \l__tag_tmpe_box
 60, 169, 175, 176, 180, 191, 192
 \l__tag_tmpe_clist
 60, 1597, 1598, 1631, 1632, 1634
 \l__tag_tmpe_int 60,
 90, 93, 98, 101, 105, 114, 430, 442, 444
 \l__tag_tmpe_prop 60, 176, 189, 203, 205
 \l__tag_tmpe_seq 51,
 58, 59, 60, 331, 333, 335, 336, 337,
 338, 443, 446, 454, 455, 457, 458,
 459, 488, 508, 518, 565, 566, 567,
 568, 723, 727, 730, 731, 739, 743,
 746, 747, 774, 775, 776, 790, 794,
 797, 798, 1599, 1603, 1613, 1614,
 1615, 1617, 1635, 1641, 1643, 1667
 \l__tag_tmpe_str
 42, 43, 48, 60, 262, 267, 272,
 297, 302, 309, 397, 402, 414, 419,
 816, 819, 820, 823, 825, 828, 835,
 842, 849, 858, 865, 873, 880, 961, 968
 \l__tag_tmpe_tl ... 42, 43, 47, 49,
 50, 51, 56, 60, 86, 88, 93, 94, 94, 96,
 102, 106, 106, 108, 109, 113, 114,
 116, 116, 117, 137, 138, 139, 141,
 143, 144, 146, 177, 178, 180, 183,
 184, 186, 191, 198, 199, 205, 205,
 206, 209, 211, 214, 215, 220, 268,
 269, 271, 275, 277, 288, 297, 299,
 300, 302, 306, 308, 308, 314, 319,
 320, 323, 359, 361, 367, 379, 398,
 455, 456, 457, 458, 460, 465, 468,
 469, 470, 475, 477, 481, 483, 485,
 488, 514, 516, 525, 546, 549, 556,
 564, 566, 575, 599, 601, 604, 606,
 620, 624, 624, 628, 632, 645, 653,
 655, 656, 658, 662, 667, 698, 702,
 723, 725, 736, 738, 751, 753, 771,
 772, 774, 802, 804, 928, 983, 986,
 1311, 1312, 1313, 1319, 1321, 1327,
 1330, 1331, 1333, 1400, 1506, 1508,
 1509, 1513, 1605, 1611, 1622, 1649
 \l__tag_tmpe_box
 60, 170, 177, 178, 182, 184
 \l__tag_tmpe_seq
 60, 1598, 1599, 1634, 1635
 \l__tag_tmpe_tl 200, 60, 89,
 104, 118, 120, 295, 301, 444, 456,
 462, 484, 490, 518, 547, 550, 550,
 551, 556, 568, 609, 611, 614, 616,
 621, 625, 672, 680, 682, 683, 685,
 689, 694, 699, 703, 752, 754, 803,
 805, 901, 905, 914, 918, 927, 930, 933
 \l__tag_tmpe_tl 60, 485, 491
 __tag_tree_fill_parenttree: ...
 171, 172, 253
 __tag_tree_final_checks: 20, 20, 354
 \g__tag_tree_id_pad_int .. 78, 82, 166
 __tag_tree_lua_fill_parenttree:
 233, 233, 250
 \g__tag_tree_openaction_struct_
 tl 32, 38, 57
 __tag_tree_parenttree_rerun_
 msg: 171, 220, 255
 __tag_tree_update_openaction: .
 42, 75
 __tag_tree_write_classmap:
 286, 286, 369
 __tag_tree_write_idtree: .. 86, 361

<code>__tag_tree_write_namespaces:</code> ..	<code>\ifpackageloaded</code>	22
..... 322, 322, 373	<code>\@maxdepth</code>	179
<code>__tag_tree_write_parenttree:</code> ..	<code>\@secondoftwo</code>	31, 55
..... 246, 246, 357	<code>\c@chapter</code>	360, 378
<code>__tag_tree_write_rolemap:</code>	<code>\r@tag@LastPage</code>	57
..... 263, 263, 365	tex commands:	
<code>__tag_tree_write_structelements:</code>	<code>\tex_botmarks:D</code>	89
..... 147, 147, 377	<code>\tex_firstmarks:D</code>	86
<code>__tag_tree_write_structtreeroot:</code>	<code>\tex_kern:D</code>	182
..... 126, 126, 381	<code>\tex_marks:D</code>	23, 32, 45, 52, 63, 69
<code>\g__tag_unique_cnt_int</code>	<code>\tex_special:D</code>	52
96, 1115, 1119, 1122, 1132, 1136, 1140	<code>\tex_splitbotmarks:D</code>	215
<code>__tag_whatsits:</code>	<code>\tex_splitfirstmarks:D</code>	195
..... 36, 43, 48, 49, 52, 293, 294	texsource (key)	1, 971
<code>tag-namespace_(rolemap-key)</code>	<code>\tiny</code>	424, 435
<code>tag/check/parent-child</code>	<code>title</code> (key)	1, 758
<code>tag/check/parent-child-end</code>	<code>title-o</code> (key)	1, 758
<code>tag/struct/1</code> internal commands:	tl commands:	
<code>__tag/struct/1</code>	<code>\c_empty_tl</code>	365, 385
<code>tag/tree/namespaces</code> internal commands:	<code>\c_space_tl</code> 55, 56, 58, 60, 100, 104,	
<code>__tag/tree/namespaces</code>	116, 167, 191, 197, 198, 216, 230,	
<code>tag/tree/parenttree</code> internal commands:	232, 234, 259, 299, 400, 417, 437,	
<code>__tag/tree/parenttree</code>	465, 895, 905, 918, 933, 944, 1011,	
<code>tag/tree/rolemap</code> internal commands:	1288, 1372, 1417, 1513, 1614, 1660	
<code>__tag/tree/rolemap</code>	<code>\tl_clear:N</code>	
<code>tagabspace</code>	.. 88, 89, 106, 177, 222, 223, 288, 409	
<code>tagmcabs</code>	<code>\tl_const:Nn</code> ...	10, 225, 241, 325, 326
<code>\tagmcbegin</code>	<code>\tl_count:n</code>	79, 83, 166
<code>\tagmcend</code>	<code>\tl_gput_left:Nn</code>	880
<code>\tagmcid</code>	<code>\tl_gput_right:Nn</code>	165, 1009
<code>\tagmcifinTF</code>	<code>\tl_gset:Nn</code>	20,
<code>\tagmcuse</code>	33, 38, 116, 225, 243, 275, 287, 320,	
<code>\tagpdfparaOff</code>	371, 378, 388, 730, 731, 746, 747,	
<code>\tagpdfparaOn</code>	753, 754, 1016, 1206, 1321, 1329, 1333	
<code>\tagpdfsetup</code> ..	<code>\tl_gset_eq:NN</code>	180, 313
8, 44, 69, 116, 117, 186, 6	<code>\tl_head:N</code>	655, 682
<code>\tagpdfsuppressmarks</code>	<code>\tl_if_blank:nTF</code>	95
<code>\tagstart</code>	<code>\tl_if_empty:NTF</code>	43,
<code>\tagstop</code>	43, 109, 185, 198, 287, 307, 314,	
<code>tagstruct</code>	333, 439, 656, 683, 772, 778, 820, 1168	
<code>\tagstructbegin</code>	<code>\tl_if_empty:nTF</code>	
114, 152, 186, 187, 45, 224, 232, 277	 51, 69, 77, 89, 142, 196,	
<code>\tagstructend</code>	210, 259, 262, 266, 279, 294, 295,	
44, 45, 226, 234, 278	297, 374, 393, 411, 440, 643, 643,	
<code>tagstructobj</code>	651, 670, 839, 855, 870, 896, 976, 1046	
8, 122	<code>\tl_if_empty_p:n</code>	310, 673
<code>\tagstructuse</code>	<code>\tl_if_eq:NNTF</code>	361, 523, 703
44, 45	<code>\tl_if_eq:NnTF</code>	108
<code>\tagtool</code>	<code>\tl_if_eq:nnTF</code>	212, 244, 274, 278
14, 20	<code>\tl_if_exist:NTF</code>	291, 1004
<code>\tagtool_(deprecated)</code>	<code>\tl_if_head_eq_charcode:nNTF</code>	49
13	<code>\tl_if_in:nnTF</code>	185
<code>tagunmarked</code> (deprecated) (key) ...	<code>\tl_new:N</code>	12, 13, 14,
1, 272	15, 16, 19, 19, 22, 22, 23, 24, 25, 26,	
<code>test/lang_(setup-key)</code>		
476		
TEX and L ^A T _E X 2 _ε commands:		
<code>\@M</code>		166
<code>\@bsphack</code>		110
<code>\@currentHref</code>		121, 95, 99
<code>\@esphack</code>		112
<code>\@gobble</code>		31, 55

32, 33, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 78, 79, 80, 81, 82, 84, 88, 162, 170, 274, 319, 321, 328, 330, 332, 333, 500, 641, 1014, 1144, 1558	
\tl_put_right:Nn	94, 104, 118, 195, 207, 226, 251, 256, 266, 267, 281, 297, 301, 302, 392, 401, 402, 407, 417, 418, 419, 426, 430, 435, 893, 903, 916, 931, 942, 1469, 1658, 1665
\tl_remove_once:Nn	1508, 1509
\tl_replace_once:Nnn	320
\tl_set:Nn	42, 83, 85, 86, 89, 116, 139, 160, 162, 180, 183, 189, 193, 197, 201, 205, 209, 213, 217, 218, 219, 224, 235, 238, 239, 242, 243, 244, 249, 250, 271, 275, 298, 302, 306, 316, 322, 329, 331, 359, 387, 387, 392, 402, 407, 441, 477, 508, 516, 518, 551, 552, 560, 566, 568, 568, 570, 601, 606, 611, 616, 629, 642, 645, 655, 658, 662, 667, 672, 682, 685, 689, 694, 707, 775, 776, 778, 790, 794, 797, 798, 804, 805, 808, 1180, 1357, 1473, 1611, 1639
\tl_set_eq:NN	179, 312
\tl_show:N	1270, 1271, 1663, 1669
\tl_tail:n	543
\tl_to_str:n	33, 48, 149, 202, 217, 546, 579
\tl_trim_spaces:n	49
\tl_use:N	293, 992, 1035, 1054, 1099
tree-mcid-index-wrong	24, 226
tree-statistic	23, 54
tree-struct-still-open	23, 47
U	
uncompress (deprecated) (key)	259
unittag_ (deprecated)	380
\unskip	44
use commands:	
\use:N	235, 475, 1272
\use:n	41, 360
\use_i:nn	99, 102, 109, 138, 154, 159, 218, 238, 365, 385, 551, 628, 632, 633, 644, 1330
\use_ii:nn	93, 108, 135, 152, 157, 219, 239, 344, 630, 641
\use_none:n	79, 92, 107, 230
\use_none:nn	78, 1447
\UseExpandableTaggingSocket	47, 61
\UseSocket	47
\UseStructureName	224, 322, 325, 326, 329, 331, 654
\UseTaggingSocket	47, 48, 61, 686, 693, 700, 707, 714, 721, 727, 734, 740, 747, 755, 768, 779, 792, 803, 816, 827, 840, 850, 863
V	
\vbadness	166, 190
vbox commands:	
\vbox_set_split_to_ht:NNn	192
\vbox_set_to_ht:Nnn	168
\vbox_unpack_drop:N	181
\vfuzz	167
viewer/startstructure_ (setup-key)	34
W	
wrong-pdfversion	220