

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

Ulrike Fischer[†]

Released 2026-10-08

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 v1.0h, last revised 2026-10-08.

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

2	Description of log messages	21
2.1	\ShowTagging command	21
2.2	Messages in checks and commands	22
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	25
3.1	Messages related to mc-chunks	25
3.2	Messages related to structures	26
3.3	Attributes	28
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	86
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	156
7.1	Variables	156
7.2	Commands and keys	157
IX	The <code>tagpdf</code> driver for <code>luatex</code>	161
1	Loading the lua	161

2	User commands to access data	166
3	Logging functions	166
4	Helper functions	168
4.1	Retrieve data functions	168
4.2	Functions to insert the pdf literals	171
5	Function for the real space chars	173
6	Function for the tagging	176
7	Parenttree	181
8	parent-child rules	184
9	Link annotations	186
X	The tagpdf-roles module	
	Tags, roles and namespace code	188
1	Code related to roles and structure names	189
1.1	Variables	189
1.2	Namespaces	191
1.3	Adding a new tag	193
1.3.1	pdf 1.7 and earlier	194
1.3.2	The pdf 2.0 version	195
1.4	Helper command to read the data from files	197
1.5	Reading the default data	199
1.6	Parent-child rules	200
1.6.1	Reading in the csv-files	201
1.6.2	Retrieving the parent-child rule	203
1.7	Key-val user interface	209
XI	The tagpdf-space module	
	Code related to real space chars	213
1	Code for interword spaces	213
	Index	217

Ulrike Fischer
Version 1.0h, released 2026-10-08

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. for debugging. `activate-tree` (*deprecated*) (*setup key*)

`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 activates automatic handling of hyphens inserted by hyphenation. It is only used in `luamode`. It accepts the following values:

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

- **char** Replaces the hyphens by U+00AD if the font contains this glyph. Note that in some newer fonts U+00AD is an invisible character and so this could 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} {2026-10-08} {1.0h}
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} {2026-10-08} {1.0h}
21 { debug code for tagpdf }
22 \@ifpackageloaded{tagpdf}{\PackageWarning{tagpdf-debug}{tagpdf~not~loaded,~quitting}}\end{package}

```

< /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} {2026-10-08} {1.0h}
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_parent_tmpa_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_softthyphen_bool` This boolean controls if the code should try to automatically handle hyphens from hyphenation. It is currently only used in luamode.

```

94 \bool_new:N \g__tag_softthyphen_bool
95 \bool_gset_true:N \g__tag_softthyphen_bool

```

(End of definition for `\g__tag_softthyphen_bool`.)

`\g__tag_unique_cnt_int` If tagpdf has to create unique names (e.g. for object names when embedding files) it can use this integer to get 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>
<code>__tag_prop_remove: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 \cs_set_eq:NN \__tag_prop_gremove:cn \prop_gremove:cn
150 % cnx temporary needed for latex-lab-graphic code
151 \cs_generate_variant:Nn \__tag_prop_gput:Nnn { Nen, Nee, Nne, Nno, cnn, cen, cne, cno }
152 \cs_generate_variant:Nn \__tag_seq_gput_right:Nn { Ne , No, cn, ce }
153 \cs_generate_variant:Nn \__tag_seq_gput_left:Nn { ce }
154 \cs_generate_variant:Nn \__tag_prop_new:N { c }
155 \cs_generate_variant:Nn \__tag_seq_new:N { c }
156 \cs_generate_variant:Nn \__tag_seq_show:N { c }
157 \cs_generate_variant:Nn \__tag_prop_show:N { c }
158 \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
`\tag_resume:n` stop the counting of paragraphs. The commands keep track of the nesting with a local
`\tag_stop:` counter. Tagging only is only restarted at the outer level, if the current level is 1. The
`\tag_start:` commands with argument allow to give a label. This is only used in debugging messages
`\tag_stop:n` to allow to follow the nesting. The label is not expand so can e.g. be a single command
`\tag_start:n` token.

When stop/start pairs are nested we do not want the inner start command to restart tagging. To control this we use a local int: The stop command will increase it. The starting will decrease it and only restart tagging, if it is zero. This will replace the label version.

```

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

```

```

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

```

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

```

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

```

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

11 Handling link artifacts

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

```

220 <*package>
221 \cs_new_protected:Npn \__tag_activate_linkbin:
222 {

```

```

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

```

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*) 242 `\keys_define:nn { __tag / setup }`

`activate/struct-dest` (*setup key*) 243 `{`

```

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

```

old, deprecated names

```

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

```

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

```

259   debug/show      .choice:,

```

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

`log` (deprecated) (*setup key*) 260 `debug/log` `.choice:,`

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

```

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

```

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

```

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

```

`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.

```

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

```

deprecated name

```

275 tagunmarked .bool_gset:N = \g__tag_tagunmarked_bool,

```

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

```

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

```

```

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

```

collect-link-artifacts (*setup key*)

```

340 activate/collect-link-artifacts .choice:,
341 activate/collect-link-artifacts/off .code:n =
342 {\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.

```

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

```

deprecated name

```
353     tabsorder .meta:n = {page/tabsorder={#1}},  
354   }
```

13 loading of engine/more dependent code

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

Ulrike Fischer

Version 1.0h, released 2026-10-08

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.

`\tag_get:nN` `\tag_get:nN {<keyword>}{<t1 var>}`

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 `<t1 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.

`\tag_get:nnN` `\tag_get:nnN {<number>} {<keyword>}{<t1 var>}`

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

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

`\tag_if_box_tagged:NNTF` `*`

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

`\tag_if_in_p:n` `*` `\tag_if_in:n {<standard structure type>}`

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

2.3 Messages from the ptagging code

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

2.4 Warning messages from the lua-code

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

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

2.5 Info messages from the lua-code

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

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

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

2.6 Debug mode messages and code

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

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

2.7 Messages

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

<u>struct-unknown</u> <u>struct-no-objnum</u> <u>struct-orphan</u> <u>struct-faulty-nesting</u> <u>struct-missing-tag</u> <u>struct-used-twice</u> <u>struct-label-unknown</u> <u>struct-show-closing</u>	Various messages related to structure. Check the definition in the code for their meaning and the arguments they take.
<u>tree-struct-still-open</u>	Message issued at the end of the compilation if there are (beside Root) other open structures on the stack.
<u>tree-statistic</u>	Message issued at the end of the compilation showing the number of objects to write
<u>show-struct</u> <u>show-kids</u>	These two messages are used in debug mode to show the current structures in the log and terminal.
<u>attr-unknown</u>	Message if an attribute is unknown.
<u>role-missing</u> <u>role-unknown</u> <u>role-unknown-tag</u> <u>role-unknown-NS</u> <u>role-tag</u> <u>new-tag</u> <u>role-parent-child-result</u> <u>role-remapping</u>	Messages related to role mapping.
<u>tree-mcid-index-wrong</u>	Used in the tree code, typically indicates the document must be rerun.
<u>sys-no-interwordspace</u>	Message if an engine doesn't support inter word spaces
<u>para-hook-count-wrong</u>	Message if the number of begin paragraph and end paragraph differ. This normally means faulty structure. <pre> 1 <@@=tag> 2 <*header> 3 \ProvidesExplPackage {tagpdf-checks-code} {2026-10-08} {1.0h} 4 {part of tagpdf - code related to checks, conditionals, debugging and messages} 5 </header> </pre>

3 Messages

3.1 Messages related to mc-chunks

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

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

(End of definition for mc-nested. This function is documented on page 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 24.)

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 24.)

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 24.)

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 24.)

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 24.)

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 24.)

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 24.)

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 24.)

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 24.)

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 24.)

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

```

(End of definition for role-missing and others. These functions are documented on page 24.)

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

```

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

```

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

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

```

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

```

(End of definition for role-parent-child-result. This function is documented on page 24.)

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

```

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

```

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

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

```

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

```

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

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

```

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

```

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

_tag_check_forbidden_parent_child:nnnn

```

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

```

```

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

```

(End of definition for __tag_check_forbidden_parent_child:nnnn.)

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

```

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

```

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

__tag_check_unresolved_parent_child:nnnn

```

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

```

(End of definition for __tag_check_unresolved_parent_child:nnnn.)

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

```

183 \socket_new:nn{tag/check/parent-child}{1}
184 \socket_new:nn{tag/check/parent-child-end}{0}
185 \socket_new_plug:nnn {tag/check/parent-child-end}{check}
186 {

```

```

187 \sys_if_engine luatex:T
188 {
189     \lua_now:e
190     {
191         ltx.__tag.func.check_parent_child_rules ( 2 )
192     }
193 }
194 }

```

And a key to disable the check

```

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

```

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

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

```

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

```

(End of definition for role-remapping. This function is documented on page 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_latex and lua_latex 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 \end{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 21.)

\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 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 1.0h, released 2026-10-08

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

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

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

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

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

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

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

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

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

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

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

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

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

5 Extension commands

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

The commands and keys should be 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

<code>\pdffakespace</code>	(lua-only) This provides a lua-version of the <code>\pdffakespace</code> 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} {2026-10-08} {1.0h}
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`
`\tagtool` (deprecated)

This command is deprecated and will be removed.

```

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. The command does nothing in vertical mode.

```

303 \bool_if:NT \g__tag_mode_lua_bool
304 {
305   \NewDocumentCommand\pdffakespace { }
306   {
307     \mode_if_vertical:F {\__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.

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

```

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

```

TODO: check the use in ltagging in para/restore

```

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

```

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

```

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            .meta:n     = {para/tag=#1}
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 1.0h, released 2026-10-08

```

Part IV

The tagpdf-tree module

Commands trees and main dictionaries

```
1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-tree-code} {2026-10-08} {1.0h}
4 {part of tagpdf - code related to writing trees and dictionaries to the pdf}
5 </header>
```

1 Trees, pdfmanagement and finalization code

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

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

1.1 Check structure

__tag_tree_final_checks:

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

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

1.2 Catalog: MarkInfo and StructTreeRoot and OpenAction

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

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

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

(End of definition for `\__tag/struct/1.`)

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

```
32 \tl_new:N \g__tag_tree_openaction_struct_tl
```

```
33 \tl_gset:Nn \g__tag_tree_openaction_struct_tl { 2 }
```

(End of definition for `\g__tag_tree_openaction_struct_tl.`)

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

```
34 \keys_define:nn { __tag / setup }
```

```
35 {
```

```
36   viewer/startstructure .code:n =
```

```
37   {
```

```
38     \tl_gset:Ne \g__tag_tree_openaction_struct_tl {#1}
```

```
39   }
```

```
40   ,viewer/startstructure .default:n = { \int_use:N \c@g__tag_struct_abs_int }
```

```
41 }
```

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

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

```
42 \cs_new_protected:Npn \__tag_tree_update_openaction:
```

```
43 {
```

```
44   \prop_get:cnNT
```

```
45   { \__kernel_pdffdict_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     }
196     \tl_put_right:Ne\l__tag_parenttree_content_tl
197     {
198       \int_eval:n {##1-1}\c_space_tl
199       [\c_space_tl %]
200     }
201     \int_step_inline:nnnn %####1
202     {0}
203     {1}
204     { \prop_count:N \l__tag_tmpa_prop -1 }
205     {

```

```

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 1.0h, released 2026-10-08

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</code> $\langle box \rangle$
------------------------------------	--

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

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

New: 2024-11-18

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

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

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

2 Public keys

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

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

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

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

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

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

stash (mc-key)	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} {2026-10-08} {1.0h}
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`

`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 `cname` 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
\tag_mc_artifact_group_end: creates a group. It pushes and pops mc-chunks at the begin and end.

```

60 <base>\cs_new_protected:Npn \tag_mc_artifact_group_begin:n #1 {}
61 <base>\cs_new_protected:Npn \tag_mc_artifact_group_end: {}
62 <*shared>
63 \cs_set_protected:Npn \tag_mc_artifact_group_begin:n #1
64 {
65   \tag_mc_end_push:
66   \tag_mc_begin:n {artifact=#1}
67   \group_begin:
68   \tag_suspend:n{artifact-group}
69 }
70
71 \cs_set_protected:Npn \tag_mc_artifact_group_end:
72 {
73   \tag_resume:n{artifact-group}
74   \group_end:
75   \tag_mc_end:
76   \tag_mc_begin_pop:n{}
77 }
78 </shared>

```

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

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

```

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

```

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

\tag_mc_end_push:
\tag_mc_begin_pop:n

```

80 <base>\cs_new_protected:Npn \tag_mc_end_push: {}
81 <base>\cs_new_protected:Npn \tag_mc_begin_pop:n #1 {}
82 <*shared>
83 \cs_set_protected:Npn \tag_mc_end_push:
84 {
85   \__tag_check_if_active_mc:T
86   {
87     \__tag_mc_if_in:TF
88     {
89       \seq_gpush:Ne \g__tag_mc_stack_seq { \tag_get:n {mc_tag} }
90       \__tag_check_mc_pushed_popped:nn
91       { pushed }
92       { \tag_get:n {mc_tag} }
93       \tag_mc_end:
94     }
95     {
96       \seq_gpush:Nn \g__tag_mc_stack_seq {-1}
97       \__tag_check_mc_pushed_popped:nn { pushed }{-1}
98     }
99   }

```

```

100 }
101
102 \cs_set_protected:Npn \tag_mc_begin_pop:n #1
103 {
104   \__tag_check_if_active_mc:T
105   {
106     \seq_gpop:NNTF \g__tag_mc_stack_seq \l__tag_tmpa_tl
107     {
108       \tl_if_eq:NnTF \l__tag_tmpa_tl {-1}
109       {
110         \__tag_check_mc_pushed_popped:nn {popped}{-1}
111       }
112       {
113         \__tag_check_mc_pushed_popped:nn {popped}{\l__tag_tmpa_tl}
114         \tag_mc_begin:n {tag=\l__tag_tmpa_tl,#1}
115       }
116     }
117     {
118       \__tag_check_mc_pushed_popped:nn {popped}{empty~stack,~nothing}
119     }
120   }
121 }

```

(End of definition for \tag_mc_end_push: and \tag_mc_begin_pop:n. These functions are documented on page 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 </shared>

Ulrike Fischer
Version 1.0h, released 2026-10-08

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} {2026-10-08} {1.0h}
4 {part of tagpdf - code related to marking chunks - generic mode}
5 </generic>
6 <*debug>
7 \ProvidesExplPackage {tagpdf-debug-generic} {2026-10-08} {1.0h}
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,
`\l__tag_mc_botmarks_seq` so we will store it locally in two sequences. `topmarks` is unusable in LaTeX so we ignore it.

```

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
`\__tag_mc_artifact_begin_marks:n` set two marks to be able to detect the case when no mark is on a page/galley. MC-begin
`\__tag_mc_end_marks:` commands will set (b,-,data) and (b,+,data), MC-end commands will set (e,-,data) and (e,+,data).

```

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 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:Nc \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:Nc
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
 __tag_mc_bdc:nn change 2023-08-18: we are delaying
 the writing to the shipout.

```

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 This create a BDC mark with an /MCID key. Most of the work here is to get the current
 __tag_mc_bdc_mcid:n number value for the MCID: they must be numbered by page starting with 0 and then
 __tag_mc_handle_mcid:nn successively. The first argument is the tag, e.g. P or Span, the second is used to pass
 __tag_mc_handle_mcid:oo 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.

```

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. `\tag_mc_end:` 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:Nc \l__tag_mc_key_tag_tl { #1 }
388     \tl_gset:Nc \g__tag_mc_key_tag_tl { #1 }
389   },
390   raw .code:n =
391   {
392     \tl_put_right:Nc \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:Nc \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 1.0h, released 2026-10-08

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

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

tag : the type (a string)

raw : more properties (string)

label: a string.

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

kids: a array of tables

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

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

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

```
1 <@@=tag>
2 <*luamode>
3 \ProvidesExplPackage {tagpdf-mc-code-lua} {2026-10-08} {1.0h}
4   {tagpdf - mc code only for the luamode }
5 </luamode>
6 <*debug>
7 \ProvidesExplPackage {tagpdf-debug-lua} {2026-10-08} {1.0h}
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
\tag\_mc\_if\_in\_p:
\tag\_mc\_if\_in:TF
68 \prg_new_conditional:Nnn \_tag\_mc\_if\_in: {p,T,F,TF}
69 {
70     \int_compare:nNnTF
71         { -2147483647 }
72         =
73         {\lua_now:e
74             {
75                 tex.print(\int_use:N \c_document_cctab, tex.getattribute(luatexbase.attributes.g__t
76             }
77         }
78         { \prg_return_false: }
79         { \prg_return_true: }
80     }
81
82 \prg_new_eq_conditional:Nnn \tag\_mc\_if\_in: \_tag\_mc\_if\_in: {p,T,F,TF}

```

(End of definition for `\_tag\_mc\_if\_in:TF` and `\tag\_mc\_if\_in:TF`. This function is documented on page 82.)

`\_tag\_mc\_lua\_set\_mc\_type\_attr:n` This takes a tag name, and sets the attributes globally to the related number.

```

\_tag\_mc\_lua\_set\_mc\_type\_attr:o
\_tag\_mc\_lua\_unset\_mc\_type\_attr:
83 \cs_new_protected:Nn \_tag\_mc\_lua\_set\_mc\_type\_attr:n % #1 is a tag name
84 {
85     %TODO ltx.__tag.func.get_num_from("#1") seems not to return a suitable number??
86     \tl_set:Nl \_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 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 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   }
158   \lua_now:e
159   {
160     ltx.__tag.func.store_struct_mcab
161     (
162       \g__tag_struct_stack_current_tl,#1
163     )
164   }
165 }
166 </luamode | debug>
167 <*luamode>
168 \cs_generate_variant:Nn \__tag_mc_handle_stash:n { e }

```

(End of definition for __tag_mc_handle_stash:n.)

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

```

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

```

set the default tag to the structure:

```

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

```

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 83.)

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

```

237 \cs_new:Npn \_tag_get_data_mc_tag: { \g__tag_mc_key_tag_tl }

```

(End of definition for _tag_get_data_mc_tag:.)

1.2 Key definitions

```

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:Nc \l__tag_tmpa_str
                262           { #1 }
                263           { default }
                264           { utf16/hex }
                265           \tl_put_right:Nc \l__tag_mc_key_properties_tl { /Alt~< }
                266           \tl_put_right:Nc \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
                267           \lua_now:e
                268           {
                269               ltx.__tag.func.store_mc_data
                270               (
                271                   \_tag_get_mc_abs_cnt:,"alt","/Alt~<\str_use:N \l__tag_tmpa_str>"
                272               )
                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 1.0h, released 2026-10-08

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

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

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},
}
```

<code>\tag_attr_new:nn</code>	<code>\tag_attr_new:nn {⟨name⟩}{⟨Content⟩}</code>
-------------------------------	---

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} {2026-10-08} {1.0h}
4 {part of tagpdf - code related to storing structure}
5 </header>
```

3 Variables

<code>\c@g__tag_struct_abs_int</code>	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`.)

<code>\g__tag_struct_objR_seq</code>	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

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

```

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

```

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

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

```

86 \prop_new_linked:N \g__tag_struct_label_num_prop

```

(End of definition for `\g__tag_struct_label_num_prop`.)

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

```

87 \bool_new:N \l__tag_struct_elem_stash_bool

```

(End of definition for `\l__tag_struct_elem_stash_bool`.)

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

```

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

```

(End of definition for `\l__tag_struct_addkid_tl`.)

3.2 Variables used by tagging code of basic elements

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

```

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

```

(End of definition for `\g__tag_struct_dest_num_prop`.)

`\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 rolemap (child) against rolemap (parent) and if =7 rolemap(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   {rolemap}
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     \tag_gset_split:NNe
724       \g__tag_struct_tag_tl
725       \g__tag_struct_tag_NS_tl
726       {#1/\l__tag_tmp_unused_tl}
727   }
728   {
729     \tag_gset_split:NNe
730       \g__tag_struct_tag_tl
731       \g__tag_struct_tag_NS_tl
732       {#1}
733   }
734   \__tag_check_structure_tag:N \g__tag_struct_tag_tl
735 }
736
737 \socket_new_plug:nnn { tag/struct/tag }{ pdf-tags }
738 {
739   \prop_get:NeNTF \g__tag_role_tags_NS_prop {#1} \l__tag_tmp_unused_tl
740   {
741     \tag_gset_split:NNe
742       \g__tag_struct_tag_tl
743       \g__tag_struct_tag_NS_tl
744       {#1/\l__tag_tmp_unused_tl}
745   }
746   {
747     \tag_gset_split:NNe
748       \g__tag_struct_tag_tl
749       \g__tag_struct_tag_NS_tl
750       {#1/\l__tag_tmp_unused_tl}
751   }
752   %\tl_gset:Ne \g__tag_struct_tag_tl { \seq_item:Nn\l__tag_tmpa_seq {1} }
753   %\tl_gset:Ne \g__tag_struct_tag_NS_tl{ \seq_item:Nn\l__tag_tmpa_seq {2} }
754   \__tag_role_get:ooNN
755     { \g__tag_struct_tag_tl }
756     { \g__tag_struct_tag_NS_tl}
757     \l__tag_tmpa_tl
758     \l__tag_tmpb_tl
759     \tl_gset:Ne \g__tag_struct_tag_tl {\l__tag_tmpa_tl}
760     \tl_gset:Ne \g__tag_struct_tag_NS_tl{\l__tag_tmpb_tl}
761     \__tag_check_structure_tag:N \g__tag_struct_tag_tl
762 }
763 \socket_assign_plug:nn { tag/struct/tag } {latex-tags}

```

```

label (struct key)
stash (struct key) 764 \keys_define:nn { __tag / struct }
parent (struct key) 765 {
firstkid (struct key) 766   label .code:n      =
tag (struct key) 767   {
title (struct key) 768     \prop_gput:Nee\g__tag_struct_label_num_prop
title-o (struct key) 769     {#1}{\int_use:N \c@g__tag_struct_abs_int}
alt (struct key) 770     \__tag_property_record:eo
actualtext (struct key) 771     {tagpdfstruct-#1}
lang (struct key) 772     { \c__tag_property_struct_clist }
ref (struct key) 773   },
E (struct key) 774   stash .bool_set:N    = \l__tag_struct_elem_stash_bool,
phoneme (struct key) 775   parent .code:n      =
776   {
777     \bool_lazy_and:nnTF
778     {
779       \prop_if_exist_p:c { g__tag_struct\_int_eval:n {#1}_prop }
780     }
781     {
782       \int_compare_p:nNn {#1}<{\c@g__tag_struct_abs_int}
783     }
784     { \tl_set:Ne \l__tag_struct_stack_parent_tmpa_tl { \int_eval:n {#1} } }
785     {
786       \msg_warning:nnee { tag } { struct-unknown }
787       { \int_eval:n {#1} }
788       { parent~key~ignored }
789     }
790   },
791   parent .default:n    = {-1},

```

this allows to change the tag of a parent if a structure is stashed (and so doesn't know its parent) so that the check doesn't give a warning.

```

792   parent-tag .code:n =
793   {
794     \prop_get:NeNTF \g__tag_role_tags_NS_prop {#1} \l__tag_tmp_unused_tl
795     {
796       \seq_set_split:Nne \l__tag_tmpa_seq { / }
797       {#1/\l__tag_tmp_unused_tl}
798     }
799     {
800       \seq_set_split:Nne \l__tag_tmpa_seq { / }
801       {#1/}
802     }
803     \tl_set:Ne \l__tag_struct_parenttag_tl { \seq_item:Nn\l__tag_tmpa_seq {1} }
804     \tl_set:Ne \l__tag_struct_parenttag_NS_tl { \seq_item:Nn\l__tag_tmpa_seq {2} }
805     \__tag_role_get:ooNN
806     { \l__tag_struct_parenttag_tl }
807     { \l__tag_struct_parenttag_NS_tl }
808     \l__tag_tmpa_tl
809     \l__tag_tmpb_tl
810     \tl_set:No \l__tag_struct_parenttag_tl { \l__tag_tmpa_tl }
811     \tl_set:No \l__tag_struct_parenttag_NS_tl { \l__tag_tmpb_tl }
812     \__tag_check_structure_tag:N \l__tag_struct_parenttag_tl
813   },

```

```

814 firstkid .code:n = { \tl_set:Nn \l__tag_struct_addkid_tl {left} },
815 tag .code:n = % S property
816 {
817   \socket_use:nn { tag/struct/tag }{#1}
818 },
819 title .code:n = % T property
820 {
821   \str_set_convert:Nnnn
822     \l__tag_tmpa_str
823     { #1 }
824     { default }
825     { utf16/hex }
826   \__tag_struct_prop_gput:nne
827     { \int_use:N \c@g__tag_struct_abs_int }
828     { T }
829     { <\l__tag_tmpa_str> }
830 },
831 title-o .code:n = % T property
832 {
833   \str_set_convert:Nonn
834     \l__tag_tmpa_str
835     { #1 }
836     { default }
837     { utf16/hex }
838   \__tag_struct_prop_gput:nne
839     { \int_use:N \c@g__tag_struct_abs_int }
840     { T }
841     { <\l__tag_tmpa_str> }
842 },
843 alt .code:n = % Alt property
844 {
845   \tl_if_empty:oF{#1}
846   {
847     \str_set_convert:Noon
848       \l__tag_tmpa_str
849       { #1 }
850       { default }
851       { utf16/hex }
852     \__tag_struct_prop_gput:nne
853       { \int_use:N \c@g__tag_struct_abs_int }
854       { Alt }
855       { <\l__tag_tmpa_str> }
856   }
857 },
858 alttext .meta:n = {alt=#1},
859 actualtext .code:n = % ActualText property
860 {
861   \tl_if_empty:oF{#1}
862   {
863     \str_set_convert:Noon
864       \l__tag_tmpa_str
865       { #1 }
866       { default }
867       { utf16/hex }

```

```

868         \_tag_struct_prop_gput:nne
869         { \int_use:N \c@g__tag_struct_abs_int }
870         { ActualText }
871         { <\l__tag_tmpa_str>}
872     }
873 },
874 phoneme .code:n = % Phoneme property
875 {
876     \tl_if_empty:oF{#1}
877     {
878         \str_set_convert:Noon
879         \l__tag_tmpa_str
880         { #1 }
881         { default }
882         { utf16/hex }
883         \_tag_struct_prop_gput:nne
884         { \int_use:N \c@g__tag_struct_abs_int }
885         { Phoneme }
886         { <\l__tag_tmpa_str>}
887     }
888 },
889 lang .code:n = % Lang property
890 {
891     \_tag_struct_prop_gput:nne
892     { \int_use:N \c@g__tag_struct_abs_int }
893     { Lang }
894     { (#1) }
895 },
896 }

```

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`

this 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

```

897 \cs_new_protected:Npn \_tag_struct_Ref_obj:nN #1 #2 %#1 a object reference
898 {
899     \tl_put_right:Ne#2
900     {
901         \c_space_tl#1
902     }
903 }
904
905 \cs_new_protected:Npn \_tag_struct_Ref_label:nN #1 #2 %#1 a label
906 {
907     \prop_get:NnNTF \g__tag_struct_label_num_prop {#1} \l__tag_tmpb_tl
908     {
909         \tl_put_right:Ne#2
910         {
911             \c_space_tl\tag_struct_object_ref:e{ \l__tag_tmpb_tl }
912         }
913     }
914 }

```

```

913     }
914     {
915         \msg_warning:nnn {tag}{struct-Ref-unknown}{Label~'#1'}
916     }
917 }
918 \cs_new_protected:Npn \__tag_struct_Ref_dest:nN #1 #2 %#1 a dest name
919 {
920     \prop_get:NnNTF \g__tag_struct_dest_num_prop {#1} \l__tag_tmpb_tl
921     {
922         \tl_put_right:Ne#2
923         {
924             \c_space_tl\tag_struct_object_ref:e{ \l__tag_tmpb_tl }
925         }
926     }
927     {
928         \msg_warning:nnn {tag}{struct-Ref-unknown}{Destination~'#1'}
929     }
930 }
931 \cs_new_protected:Npn \__tag_struct_Ref_dest_parent:nN #1 #2 %#1 a dest name
932 {
933     \prop_get:NnNTF \g__tag_struct_dest_num_prop {#1} \l__tag_tmpb_tl
934     { %do not overwrite \l__tag_tmpa_tl!
935         \prop_get:cnN
936         { g__tag_struct_\l__tag_tmpb_tl _prop }{parentnum}\l__tag_tmpb_tl
937         \tl_put_right:Ne#2
938         {
939             \c_space_tl\tag_struct_object_ref:e{ \l__tag_tmpb_tl }
940         }
941     }
942     {
943         \msg_warning:nnn {tag}{struct-Ref-unknown}{Destination~'#1'}
944     }
945 }
946 \cs_new_protected:Npn \__tag_struct_Ref_num:nN #1 #2 %#1 a structure number
947 {
948     \tl_put_right:Ne#2
949     {
950         \c_space_tl\tag_struct_object_ref:e{ #1 }
951     }
952 }
953

```

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

ref (struct key)

E (struct key)

```

954 \keys_define:nn { __tag / struct }
955 {
956     ref .code:n      = % ref property
957     {
958         \clist_map_inline:on {#1}
959         {
960             \tag_struct_gput:nne
961             {\int_use:N \c@g__tag_struct_abs_int}{ref_label}{ ##1 }
962         }
963     },

```

```

964 E .code:n          = % E property
965 {
966     \str_set_convert:Nnon
967     \l__tag_tmpa_str
968     { #1 }
969     { default }
970     { utf16/hex }
971     \__tag_struct_prop_gput:nne
972     { \int_use:N \c@g__tag_struct_abs_int }
973     { E }
974     { <\l__tag_tmpa_str> }
975 },
976 }

```

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

```

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

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

```

```

1007 \cs_new_protected:Npn \__tag_struct_add_AF:nn #1 #2
1008 % #1 struct num #2 object reference
1009 {
1010     \tl_if_exist:cTF
1011     {
1012         g__tag_struct_#1_AF_tl
1013     }
1014     {
1015         \tl_gput_right:ce
1016         { g__tag_struct_#1_AF_tl }
1017         { \c_space_tl #2 }
1018     }
1019     {
1020         \tl_new:c
1021         { g__tag_struct_#1_AF_tl }
1022         \tl_gset:ce
1023         { g__tag_struct_#1_AF_tl }
1024         { #2 }
1025     }
1026 }
1027 \cs_generate_variant:Nn \__tag_struct_add_AF:nn {en,ee}
1028 \keys_define:nn { __tag / struct }
1029 {
1030     AF .code:n          = % AF property
1031     {
1032         \pdf_object_if_exist:eTF {#1}
1033         {
1034             \__tag_struct_add_AF:ee
1035             { \int_use:N \c@g__tag_struct_abs_int }{\pdf_object_ref:e {#1}}
1036             \__tag_struct_prop_gput:nne
1037             { \int_use:N \c@g__tag_struct_abs_int }
1038             { AF }
1039             {
1040                 [
1041                     \tl_use:c
1042                     { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
1043                 ]
1044             }
1045         }
1046         {
1047             % message?
1048         }
1049     },
1050     AFref .code:n       = % AF property
1051     {
1052         \tl_if_empty:eF {#1}
1053         {
1054             \__tag_struct_add_AF:ee { \int_use:N \c@g__tag_struct_abs_int }{#1}
1055             \__tag_struct_prop_gput:nne
1056             { \int_use:N \c@g__tag_struct_abs_int }
1057             { AF }
1058             {
1059                 [
1060                     \tl_use:c

```

```

1061         { g__tag_struct_int_eval:n { \c@g__tag_struct_abs_int}_AF_tl }
1062     ]
1063 }
1064 }
1065 },
1066 ,AFinline .code:n =
1067 {
1068     \__tag_struct_add_inline_AF:nn {#1}{txt}
1069 }
1070 ,AFinline-o .code:n =
1071 {
1072     \__tag_struct_add_inline_AF:on {#1}{txt}
1073 }
1074 ,texsource .code:n =
1075 {
1076     \group_begin:
1077     \pdfdict_put:nnn { l_pdffile/Filespec } {Desc}{(TeX~source)}
1078     \pdfdict_put:nnn { l_pdffile/Filespec } {AFRelationship} { /Source }
1079     \__tag_struct_add_inline_AF:on {#1}{tex}
1080     \group_end:
1081 }
1082 ,mathml .code:n =
1083 {
1084     \group_begin:
1085     \pdfdict_put:nnn { l_pdffile/Filespec } {Desc}{(mathml~representation)}
1086     \pdfdict_put:nnn { l_pdffile/Filespec } {AFRelationship} { /Supplement }
1087     \pdfdict_put:nne { l_pdffile } {Subtype}
1088     { \pdf_name_from_unicode_e:n{application/mathml+xml} }
1089     \__tag_struct_add_inline_AF:on {#1}{xml}
1090     \group_end:
1091 }
1092 }

```

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!

```

1093 \keys_define:nn { __tag / setup }
1094 {
1095     root-AF .code:n =
1096     {
1097         \pdf_object_if_exist:nTF {#1}
1098         {
1099             \__tag_struct_add_AF:ee { 1 }{\pdf_object_ref:n {#1}}
1100             \__tag_struct_prop_gput:nne
1101             { 1 }
1102             { AF }
1103             {
1104                 [
1105                     \tl_use:c
1106                     { g__tag_struct_1_AF_tl }
1107                 ]
1108             }
1109         }
1110     }
1111 }

```

```

1112     }
1113   },
1114 }

```

`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

```

1115 \keys_define:nn { __tag / setup }
1116 {
1117   root-supplemental-file .code:n =
1118   {
1119     \group_begin:
1120     \pdfdict_put:nnn {l_pdffile/Filespec} {AFRelationship}{/Supplement}
1121     \int_gincr:N \g__tag_unique_cnt_int
1122     \pdffile_embed_file:eee
1123     {#1}
1124     {#1}
1125     {__tag_latex_css_\int_use:N\g__tag_unique_cnt_int}
1126     \keys_set:nn
1127     {__tag / setup}
1128     {root-AF={__tag_latex_css_\int_use:N\g__tag_unique_cnt_int}}
1129     \group_end:
1130   }
1131 }

```

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

```

1132 \keys_define:nn { __tag / setup }
1133 {
1134   catalog-supplemental-file .code:n =
1135   {
1136     \group_begin:
1137     \pdfdict_put:nnn {l_pdffile/Filespec} {AFRelationship}{/Supplement}
1138     \int_gincr:N \g__tag_unique_cnt_int
1139     \pdffile_embed_file:eee
1140     {#1}
1141     {#1}
1142     {__tag_latex_css_\int_use:N\g__tag_unique_cnt_int}
1143     \pdfmanagement_add:nne
1144     {Catalog}
1145     {AF}
1146     {\pdf_object_ref:e{__tag_latex_css_\int_use:N\g__tag_unique_cnt_int }}
1147     \group_end:
1148   }
1149 }

```

6 User commands

We allow to set a language by default

`\l__tag_struct_lang_tl`

```

1150 \tl_new:N \l__tag_struct_lang_tl
1151 \</package>

```

(End of definition for \l__tag_struct_lang_tl.)

\tag_struct_begin:n

\tag_struct_end:

```

1152 <base>\cs_new_protected:Npn \tag_struct_begin:n #1 {\int_gincr:N \c@g__tag_struct_abs_int}
1153 <base>\cs_new_protected:Npn \tag_struct_end:{}
1154 <base>\cs_new_protected:Npn \tag_struct_end:n{}
1155 <*package|debug>
1156 <package>\cs_set_protected:Npn \tag_struct_begin:n #1 % #1 key-val
1157 <debug>\cs_set_protected:Npn \tag_struct_begin:n #1 % #1 key-val
1158 {
1159 <package>\__tag_check_if_active_struct:T
1160 <debug>\__tag_check_if_active_struct:TF
1161 {
1162   \group_begin:
1163   \int_gincr:N \c@g__tag_struct_abs_int
1164   \__tag_prop_new:c { g__tag_struct\_int_eval:n { \c@g__tag_struct_abs_int }_prop }
1165   <debug> \prop_new:c { g__tag_struct_debug\_int_eval:n { \c@g__tag_struct_abs_int }_prop }
1166   \__tag_seq_new:c { g__tag_struct_kids\_int_eval:n { \c@g__tag_struct_abs_int }_seq }
1167   <debug> \seq_new:c { g__tag_struct_debug_kids\_int_eval:n { \c@g__tag_struct_abs_int }_seq }
1168   \pdf_object_new_indexed:nn { __tag/struct }
1169   { \c@g__tag_struct_abs_int }
1170   \__tag_struct_prop_gput:nnn
1171   { \int_use:N \c@g__tag_struct_abs_int }
1172   { Type }
1173   { /StructElem }
1174   \tl_if_empty:NF \l__tag_struct_lang_tl
1175   {
1176     \__tag_struct_prop_gput:nne
1177     { \int_use:N \c@g__tag_struct_abs_int }
1178     { Lang }
1179     { (\l__tag_struct_lang_tl) }
1180   }
1181   \__tag_struct_prop_gput:nnn
1182   { \int_use:N \c@g__tag_struct_abs_int }
1183   { Type }
1184   { /StructElem }
1185
1186   \tl_set:Nn \l__tag_struct_stack_parent_tmpa_tl {-1}
1187   \keys_set:nn { __tag / struct } { #1 }
1188
1189   \__tag_struct_set_tag_info:eoo
1190   { \int_use:N \c@g__tag_struct_abs_int }
1191   { g__tag_struct_tag_tl }
1192   { g__tag_struct_tag_NS_tl }
1193   \__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.

```

1193   \int_compare:nNnT { \l__tag_struct_stack_parent_tmpa_tl } = { -1 }
1194   {
1195     \seq_get:NNF
1196     \g__tag_struct_stack_seq
1197     \l__tag_struct_stack_parent_tmpa_tl
1198     {

```

```

1199         \msg_error:nn { tag } { struct-faulty-nesting }
1200     }
1201 }
1202 \seq_gpush:NV \g__tag_struct_stack_seq \c@g__tag_struct_abs_int
1203 \__tag_role_get:ooNN
1204 { \g__tag_struct_tag_tl }
1205 { \g__tag_struct_tag_NS_tl }
1206 \l__tag_struct_roletag_tl
1207 \l__tag_struct_roletag_NS_tl

```

We push the role tag on the stack:

```

1208 \seq_gpush:Ne \g__tag_struct_tag_stack_seq
1209 {{\g__tag_struct_tag_tl}{\l__tag_struct_roletag_tl}}
1210 \seq_gpush:Ne \g__tag_struct_roletag_stack_seq
1211 {\l__tag_struct_roletag_tl}
1212 \tl_gset:NV \g__tag_struct_stack_current_tl \c@g__tag_struct_abs_int
1213 \__tag_struct_set_attribute:
1214 %\seq_show:N \g__tag_struct_stack_seq

```

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

```

1215 \__tag_struct_prop_gput:nne
1216 { \int_use:N \c@g__tag_struct_abs_int }
1217 { rolemap }
1218 {
1219     {\l__tag_struct_roletag_tl}{\l__tag_struct_roletag_NS_tl}
1220 }

```

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.

```

1221 \str_case:onTF { \l__tag_struct_roletag_tl }
1222 {
1223     {Part} {}
1224     {Div} {}
1225     {NonStruct} {}
1226 }
1227 {
1228     \bool_if:NTF \l__tag_struct_elem_stash_bool
1229     {
1230         \__tag_struct_prop_gput:nne
1231         { \int_use:N \c@g__tag_struct_abs_int }
1232         { parentrole }
1233         {
1234             {\l__tag_struct_parenttag_tl}{\l__tag_struct_parenttag_NS_tl}
1235         }
1236     }
1237 }
1238 \prop_get:cnNT
1239 { g__tag_struct_ \l__tag_struct_stack_parent_tmpa_tl _prop }
1240 { parentrole }
1241 \l__tag_get_tmpc_tl

```

```

1242         {
1243             \__tag_struct_prop_gput:nno
1244             { \int_use:N \c@g__tag_struct_abs_int }
1245             { parentrole }
1246             {
1247                 \l__tag_get_tmpc_tl
1248             }
1249         }
1250     }
1251 }
1252 {
1253     \__tag_struct_prop_gput:nne
1254     { \int_use:N \c@g__tag_struct_abs_int }
1255     { parentrole }
1256     {
1257         {\l__tag_struct_roletag_tl}{\l__tag_struct_roletag_NS_tl}
1258     }
1259 }
1260 \bool_if:NF
1261     \l__tag_struct_elem_stash_bool
1262     {

```

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.

```

1263         \socket_use:nn{tag/check/parent-child}
1264         {
1265             \__tag_struct_check_parent_child:oo
1266             { \l__tag_struct_stack_parent_tmpa_tl }
1267             { \int_use:N \c@g__tag_struct_abs_int }
1268         }

```

Set the Parent structure number.

```

1269         \__tag_struct_prop_gput:nne
1270         { \int_use:N \c@g__tag_struct_abs_int }
1271         { parentnum }
1272         {
1273             \l__tag_struct_stack_parent_tmpa_tl
1274         }

1275         %record this structure as kid:
1276         %\tl_show:N \g__tag_struct_stack_current_tl
1277         %\tl_show:N \l__tag_struct_stack_parent_tmpa_tl
1278         \use:c { __tag_struct_kid_struct_gput_ \l__tag_struct_addkid_tl :ee }
1279         { \l__tag_struct_stack_parent_tmpa_tl }
1280         { \g__tag_struct_stack_current_tl }
1281         %\prop_show:c { g__tag_struct_\g__tag_struct_stack_current_tl _prop }
1282         %\seq_show:c {g__tag_struct_kids_\l__tag_struct_stack_parent_tmpa_tl _seq}
1283     }

```

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.

```

1284 <debug>         \prop_gset_eq:cc
1285 <debug>         { g__tag_struct_debug_\int_eval:n {\c@g__tag_struct_abs_int}_prop }

```

```

1286 <debug>          { g__tag_struct_int_eval:n { \c@g__tag_struct_abs_int}_prop }
1287 <debug>          \prop_gput:cne
1288 <debug>          { g__tag_struct_debug_int_eval:n { \c@g__tag_struct_abs_int}_prop }
1289 <debug>          { parentnum }
1290 <debug>          {
1291 <debug>              \bool_if:NTF \l__tag_struct_elem_stash_bool
1292 <debug>              {no~parent:~stashed}
1293 <debug>              {
1294 <debug>                  \l__tag_struct_stack_parent_tmpa_tl \c_space_tl =~
1295 <debug>                  \prop_item:cn{ g__tag_struct_l__tag_struct_stack_parent_tmpa_tl _p
1296 <debug>              }
1297 <debug>          }
1298 <debug>          \prop_gput:cne
1299 <debug>          { g__tag_struct_debug_int_eval:n { \c@g__tag_struct_abs_int}_prop }
1300 <debug>          { NS }
1301 <debug>          { \g__tag_struct_tag_NS_tl }

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

```

```

1340         }
1341     }
1342     <debug>\__tag_debug_struct_end_insert:
1343     }
1344     <debug>{\__tag_debug_struct_end_ignore:}
1345     }
1346
1347     \cs_set_protected:Npn \tag_struct_end:n #1
1348     {
1349     <debug>    \__tag_check_if_active_struct:T{\__tag_debug_struct_end_check:n{#1}}
1350     \tag_struct_end:
1351     }
1352     </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.

```

1353     <base>\cs_new_protected:Npn \tag_struct_use:n #1 {}
1354     <*package|debug>
1355     \cs_set_protected:Npn \tag_struct_use:n #1 %#1 is the label
1356     {
1357     \__tag_check_if_active_struct:T
1358     {
1359     \prop_if_exist:cTF
1360     { g__tag_struct_\property_ref:enn{tagpdfstruct-#1}{tagstruct}{unknown}_prop } %
1361     {
1362     \__tag_check_struct_used:n {#1}
1363     \tl_set:Nx \l__tag_get_child_tmpa_tl
1364     { \property_ref:enn{tagpdfstruct-#1}{tagstruct}{1} }

```

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

```

1365     \__tag_struct_kid_struct_gput_right:ee
1366     { \g__tag_struct_stack_current_tl }
1367     { \l__tag_get_child_tmpa_tl }

```

add the current structure to the labeled one as parents

```

1368     \__tag_prop_gput:cne
1369     { g__tag_struct_\l__tag_get_child_tmpa_tl _prop }
1370     { parentnum }
1371     {
1372     \g__tag_struct_stack_current_tl
1373     }

```

debug code

```

1374     <debug>    \prop_gput:cne
1375     <debug>    { g__tag_struct_debug_\l__tag_get_child_tmpa_tl _prop }
1376     <debug>    { parentnum }
1377     <debug>    {
1378     <debug>    \g__tag_struct_stack_current_tl\c_space_tl=~
1379     <debug>    \g__tag_struct_tag_tl
1380     <debug>    }

```

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

```

1381      \socket_use:nn{tag/check/parent-child}
1382      {
1383          \__tag_struct_use_check_parent_child:oo
1384          { \g__tag_struct_stack_current_tl }
1385          { \l__tag_get_child_tmpa_tl }
1386      }
1387  }
1388  {
1389      \msg_warning:nnn{ tag }{struct-label-unknown}{#1}
1390  }
1391 }
1392 }
1393 \end{package}

```

(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.

```

1394 \base\cs_new_protected:Npn \tag_struct_use_num:n #1 {}
1395 \end{package}
1396 \cs_set_protected:Npn \tag_struct_use_num:n #1 {%#1 is structure number
1397 {
1398     \__tag_check_if_active_struct:T
1399     {
1400         \prop_if_exist:cTF
1401         { g__tag_struct_#1_prop } %
1402         {
1403             \prop_get:cnNT
1404             {g__tag_struct_#1_prop}
1405             {parentnum}
1406             \l__tag_tmpa_tl
1407             {
1408                 \msg_warning:nnn { tag } {struct-used-twice} {#1}
1409             }
1410         }
1411     }
1412 }

```

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

```

1410     \__tag_struct_kid_struct_gput_right:ee
1411     { \g__tag_struct_stack_current_tl }
1412     { #1 }

```

add the current structure to #1 as parent

```

1413     \__tag_struct_prop_gput:nne
1414     { #1 }
1415     { parentnum }
1416     {
1417         \g__tag_struct_stack_current_tl
1418     }
1419 \end{package}

```

```

1420 <debug>          { g__tag_struct_debug_#1_prop }
1421 <debug>          { parentnum }
1422 <debug>          {
1423 <debug>          \g__tag_struct_stack_current_tl\c_space_tl=~
1424 <debug>          \g__tag_struct_tag_tl
1425 <debug>          }

```

check if the tag is allowed as child.

```

1426          \socket_use:nn{tag/check/parent-child}
1427          {
1428          \__tag_struct_use_check_parent_child:oo
1429          {\g__tag_struct_stack_current_tl}
1430          {#1}
1431          }
1432          }
1433          {
1434          \msg_warning:nnn{ tag }{struct-label-unknown}{#1}
1435          }
1436          }
1437      }
1438 \cs_generate_variant:Nn\tag_struct_use_num:n {o}
1439 </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.

```

1440 <*package>
1441 \cs_new:Npn \tag_struct_object_ref:n #1
1442 {
1443   \pdf_object_ref_indexed:nn {\__tag/struct}{ #1 }
1444 }
1445 \cs_generate_variant:Nn \tag_struct_object_ref:n {e}
1446 </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.

- The keyword **tag** updates the tag of a structure (the S entry).
- The keyword **alt** *replaces* the current value of Alt. The argument is not purified, only converted to utf18/hex. If needed \pdf_purify:nN should be applied first.
- The keyword **title** *replaces* the current value of T. The argument is not purified, only converted to utf18/hex. If needed \pdf_purify:nN should be applied first.
- 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`.

```

1447 <base>\cs_new_protected:Npn \tag_struct_gput:nnn #1 #2 #3{
1448 <*package>
1449 \cs_set_protected:Npn \tag_struct_gput:nnn #1 #2 #3
1450 {
1451   \cs_if_exist_use:cF {__tag_struct_gput_data_#2:nn}
1452   { %warning??
1453     \use_none:nn
1454   }
1455   {#1}{#3}
1456 }
1457 \cs_generate_variant:Nn \tag_struct_gput:nnn {ene,nne}

```

`\tag_struct_map_function:N`

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

```

1458 \cs_new_protected:Npn \tag_struct_map_function:N #1
1459 {
1460   \seq_map_pairwise_function:NNN
1461   \g__tag_struct_tag_stack_seq
1462   \g__tag_struct_stack_seq
1463   #1
1464 }
1465 </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`

```

1466 <*package>
1467 \cs_new_protected:Npn \__tag_struct_gput_data_ref_aux:nnn #1 #2 #3
1468   % #1 receiving struct num, #2 key word #3 value
1469 {
1470   \prop_get:cnNTF
1471   { g__tag_struct_#1_prop }

```

```

1472     {Ref}
1473     \l__tag_get_tmpc_tl
1474     {
1475         \tl_put_right:No \l__tag_get_tmpc_tl
1476         {\cs:w __tag_struct_Ref_#2:nN \cs_end: {#3},}
1477     }
1478     {
1479         \tl_set:No \l__tag_get_tmpc_tl
1480         {\cs:w __tag_struct_Ref_#2:nN \cs_end: {#3},}
1481     }
1482     \__tag_struct_prop_gput:nno
1483     { #1 }
1484     { Ref }
1485     { \l__tag_get_tmpc_tl }
1486 }
1487 \cs_new_protected:Npn \__tag_struct_gput_data_ref:nn #1 #2
1488 {
1489     \__tag_struct_gput_data_ref_aux:nnn {#1}{obj}{#2}
1490 }
1491 \cs_new_protected:Npn \__tag_struct_gput_data_ref_label:nn #1 #2
1492 {
1493     \__tag_struct_gput_data_ref_aux:nnn {#1}{label}{#2}
1494 }
1495 \cs_new_protected:Npn \__tag_struct_gput_data_ref_dest:nn #1 #2
1496 {
1497     \__tag_struct_gput_data_ref_aux:nnn {#1}{dest}{#2}
1498 }
1499 \cs_new_protected:Npn \__tag_struct_gput_data_ref_dest_parent:nn #1 #2
1500 {
1501     \__tag_struct_gput_data_ref_aux:nnn {#1}{dest_parent}{#2}
1502 }
1503 \cs_new_protected:Npn \__tag_struct_gput_data_ref_num:nn #1 #2
1504 {
1505     \__tag_struct_gput_data_ref_aux:nnn {#1}{num}{#2}
1506 }
1507
1508 \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

```

1509 \cs_new_protected:Npn \__tag_struct_gput_data_attribute:nn #1 #2
1510 {
1511     \pdf_object_unnamed_write:nn {dict} {#2}
1512     \prop_get:cnTF { g__tag_struct_#1_prop }{A} \l__tag_tmpa_tl
1513     {
1514         \tl_remove_once:Nn\l__tag_tmpa_tl{[]}
1515         \tl_remove_once:Nn\l__tag_tmpa_tl{]}
1516         \__tag_prop_gput:cne { g__tag_struct_#1_prop }
1517         { A }
1518         {
1519             [ \l__tag_tmpa_tl \c_space_tl \pdf_object_ref_last: ]
1520         }
1521     }
1522     {

```

```

1523         \__tag_prop_gput:cne { g__tag_struct_#1_prop }
1524         { A }
1525         { \pdf_object_ref_last: }
1526     }
1527 }

```

(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.

```

1528 \cs_new_protected:Npn \__tag_struct_gput_data_tag:nn #1 #2
1529 {
1530     \__tag_struct_prop_gput:nne{#1}{S}{#2}
1531 }

```

(End of definition for __tag_struct_gput_data_tag:nn.)

__tag_struct_gput_data_alt:nn This is used by \tag_struct_gput:nnn to update the Alt of a structure.

```

1532 \cs_new_protected:Npn \__tag_struct_gput_data_alt:nn #1 #2
1533 {
1534     \str_set_convert:Nnnn
1535         \l__tag_tmpa_str
1536         { #2 }
1537         { default }
1538         { utf16/hex }
1539     \__tag_struct_prop_gput:nne{#1}{Alt}{<\l__tag_tmpa_str>}
1540 }

```

(End of definition for __tag_struct_gput_data_alt:nn.)

__tag_struct_gput_data_title:nn This is used by \tag_struct_gput:nnn to update the title key of a structure.

```

1541 \cs_new_protected:Npn \__tag_struct_gput_data_title:nn #1 #2
1542 {
1543     \str_set_convert:Nnnn
1544         \l__tag_tmpa_str
1545         { #2 }
1546         { default }
1547         { utf16/hex }
1548     \__tag_struct_prop_gput:nne{#1}{T}{<\l__tag_tmpa_str>}
1549 }

```

(End of definition for __tag_struct_gput_data_title:nn.)

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

```

1550 \cs_new_protected:Npn \__tag_struct_gput_data_C:nn #1 #2
1551 {
1552     \int_case:nnF {\clist_count:n{#2}}
1553     {
1554         {0}{ }
1555         {1}{\__tag_struct_prop_gput:nne{#1}{C}{#2}}
1556     }
1557     {
1558         \__tag_struct_prop_gput:nne{#1}{C}{[\clist_use:nn{#2}{ }]}
1559     }
1560 }

```

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

`\tag_struct_parent_int:`

```

1561 \cs_new_protected:Npn \tag_struct_insert_annot:nn #1 #2 %#1 should be an object reference
1562                                     %#2 struct parent num
1563 {
1564   \__tag_check_if_active_struct:T
1565   {
1566     \__tag_struct_insert_annot:nn {#1}{#2}
1567   }
1568 }
1569
1570 \cs_generate_variant:Nn \tag_struct_insert_annot:nn {xx,ee}
1571 \cs_new:Npn \tag_struct_parent_int: {\int_use:c { c@g__tag_parenttree_obj_int }}
1572
1573 \</package>
1574

```

(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

```

1575 \<*header>
1576 \ProvidesExplPackage {tagpdf-attr-code} {2026-10-08} {1.0h}
1577 {part of tagpdf - code related to attributes and attribute classes}
1578 \</header>

```

7.1 Variables

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

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

`\g__tag_attr_objref_prop`

`\l__tag_attr_value_tl`

```

1579 \<*package>
1580 \prop_new:N \g__tag_attr_entries_prop
1581 \prop_new_linked:N \g__tag_attr_class_used_prop
1582 \tl_new:N \l__tag_attr_value_tl
1583 \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.

```

1584 \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:nn` This allows to define attributes. Defined attributes are stored in a global property. `\role/new-attribute` expects two brace group, the name and the content. The content typically needs an `/O` key for the owner. An example look like this.

`\__tag_attr_new_entry:nn` `role/new-attribute (setup-key)` TODO: consider to put them directly in the ClassMap, that is perhaps more effective.

`role/new-attribute* (setup-key)`

`newattribute (deprecated)`

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

1585 \cs_new_protected:Npn \__tag_attr_new_entry:nn #1 #2 %#1:name, #2: content
1586 {
1587   \prop_gput:Nen \g__tag_attr_entries_prop
1588     {\pdf_name_from_unicode_e:n{#1}}{#2}
1589 }
1590 \cs_new_protected:Npn \__tag_attr_new_used_entry:nn #1 #2 %#1:name, #2: content
1591 {
1592   \prop_gput:Nen \g__tag_attr_entries_prop
1593     {\pdf_name_from_unicode_e:n{#1}}{#2}
1594   \seq_gput_left:Nn\g__tag_attr_class_used_seq
1595     {\pdf_name_from_unicode_e:n{#1}}
1596 }
1597
1598 \cs_set_eq:NN \tag_attr_new:nn \__tag_attr_new_entry:nn
1599 \cs_generate_variant:Nn \__tag_attr_new_entry:nn {ee}
1600 \cs_generate_variant:Nn \tag_attr_new:nn {ee}
1601
1602 \keys_define:nn { __tag / setup }
1603 {
1604   role/new-attribute .code:n =
1605   {
1606     \__tag_attr_new_entry:nn #1
1607   },
1608   role/new-attribute* .code:n =
1609   {
1610     \__tag_attr_new_used_entry:nn #1
1611   }
}

deprecated name
1612 ,newattribute .code:n =
1613 {
1614   \__tag_attr_new_entry:nn #1
1615 },
1616 }
```

(End of definition for `\tag_attr_new:nn` and others. These functions are documented on page 117.)

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

```
1617 \keys_define:nn { __tag / struct }
```

```

1618 {
1619     attribute-class .code:n =
1620     {
1621         \clist_set:Ne \l__tag_tmpa_clist { #1 }
1622         \seq_set_from_clist:NN \l__tag_tmpb_seq \l__tag_tmpa_clist

we convert the names into pdf names with slash
1623         \seq_set_map_e:NNn \l__tag_tmpa_seq \l__tag_tmpb_seq
1624         {
1625             \pdf_name_from_unicode_e:n {##1}
1626         }
1627         \seq_map_inline:Nn \l__tag_tmpa_seq
1628         {
1629             \prop_get:NnNF \g__tag_attr_entries_prop {##1}\l__tag_tmpa_tl
1630             {
1631                 \msg_error:nnn { tag } { attr-unknown } { ##1 }
1632             }
1633             \prop_gput:Nnn\g__tag_attr_class_used_prop { ##1} {}
1634         }
1635         \tl_set:Ne \l__tag_tmpa_tl
1636         {
1637             \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}
1638             \seq_use:Nn \l__tag_tmpa_seq { \c_space_tl }
1639             \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}
1640         }
1641         \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 0 }
1642         {
1643             \__tag_struct_prop_gput:nne
1644             { \int_use:N \c@g__tag_struct_abs_int }
1645             { C }
1646             { \l__tag_tmpa_tl }
1647             %\prop_show:c { g__tag_struct\int_eval:n {\c@g__tag_struct_abs_int}_prop }
1648         }
1649     }
1650 }

```

attribute (struct key)

```

1651 \tl_new:N\l__tag_attr_array_end_tl
1652 \keys_define:nn { __tag / struct }
1653 {
1654     attribute .code:n = % A property (attribute, value currently a dictionary)
1655     {
1656         \clist_set:Ne \l__tag_tmpa_clist { #1 }
1657         \clist_if_empty:NF \l__tag_tmpa_clist
1658         {
1659             \seq_set_from_clist:NN \l__tag_tmpb_seq \l__tag_tmpa_clist

```

we convert the names into pdf names with slash

```

1660         \seq_set_map_e:NNn \l__tag_tmpa_seq \l__tag_tmpb_seq
1661         {
1662             \pdf_name_from_unicode_e:n {##1}
1663         }
1664         \prop_get:cnNTF
1665         { g__tag_struct_ \int_use:N \c@g__tag_struct_abs_int _prop }

```

```

1666     {A}
1667     \l__tag_tmpa_tl
1668 { %there was already an attribute
1669   \tl_remove_once:Nn\l__tag_tmpa_tl{[]}
1670   \tl_remove_once:Nn\l__tag_tmpa_tl{[]}}
1671
1672   \tl_set:Nn \l__tag_attr_value_tl {}
1673   \tl_put_right:No \l__tag_attr_value_tl {\l__tag_tmpa_tl}
1674   \tl_set:Nn \l__tag_attr_array_end_tl {}
1675 }
1676 {
1677   %no attribute
1678   \int_compare:nNnTF { \seq_count:N \l__tag_tmpa_seq } > {1}
1679   {
1680     \tl_set:Nn \l__tag_attr_value_tl {}
1681     \tl_set:Nn \l__tag_attr_array_end_tl {}
1682   }
1683   {
1684     \tl_set:Nn \l__tag_attr_value_tl {}
1685     \tl_set:Nn \l__tag_attr_array_end_tl {}
1686   }
1687 }
1688 \seq_map_inline:Nn \l__tag_tmpa_seq
1689 {
1690   \prop_get:NnNF \g__tag_attr_entries_prop {##1}\l__tag_tmp_unused_tl
1691   {
1692     \msg_error:nnn { tag } { attr-unknown } { ##1 }
1693   }
1694   \prop_get:NnNF \g__tag_attr_objref_prop {##1}\l__tag_tmpa_tl
1695   {%\prop_show:N \g__tag_attr_entries_prop
1696     \pdf_object_unnamed_write:ne
1697     { dict }
1698     {
1699       \prop_item:Nn\g__tag_attr_entries_prop {##1}
1700     }
1701     \prop_gput:Nne \g__tag_attr_objref_prop {##1} {\pdf_object_ref_last:}
1702   }
1703   \tl_put_right:Ne \l__tag_attr_value_tl
1704   {
1705     \c_space_tl
1706     \prop_item:Nn \g__tag_attr_objref_prop {##1}
1707   }
1708 }
1709 \tl_put_right:No \l__tag_attr_value_tl
1710 {
1711   \l__tag_attr_array_end_tl
1712 }
1713 \__tag_struct_prop_gput:nne
1714 { \int_use:N \c@g__tag_struct_abs_int }
1715 { A }
1716 { \l__tag_attr_value_tl }
1717 }
1718 },
1719 attribute-unnamed .code:n =

```

```

1720     {
1721         \clist_map_inline:nn {#1}
1722         {
1723             \__tag_struct_gput_data_attribute:nn
1724             { \int_use:N \c@g__tag_struct_abs_int }{##1}
1725         }
1726     },
1727     attribute* .code:n =
1728     {
1729         \__tag_prop_gremove:cn
1730         { g__tag_struct_ \int_use:N \c@g__tag_struct_abs_int _prop }{A}
1731         \keys_set:nn { __tag / struct }{ attribute = {#1} }
1732     },
1733     attribute-unnamed* .code:n =
1734     {
1735         \__tag_prop_gremove:cn
1736         { g__tag_struct_ \int_use:N \c@g__tag_struct_abs_int _prop }{A}
1737         \keys_set:nn { __tag / struct }{ attribute-unnamed = {#1} }
1738     }
1739 }
1740 \end{package}

```

Ulrike Fischer
Version 1.0h, released 2026-10-08

Part IX

The tagpdf driver for luatex

```
1 <@@=tag>
2 <*luatex>
3 \ProvidesExplFile {tagpdf-luatex.def} {2026-10-08} {1.0h}
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_prop_gremove:cn #1 #2
36   {
37     \prop_gremove:cn {#1} { #2 }
38     \lua_now:e { ltx.__tag.tables['#1'] ["#2"] = nil }
39   }
40
41 \cs_set_protected:Npn \__tag_seq_gput_right:Nn #1 #2
42   {
43     \seq_gput_right:Nn #1 { #2 }
44     \lua_now:e { table.insert(ltx.__tag.tables['\cs_to_str:N#1'], "#2") }
45   }

```

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

```

46 \cs_set_protected:Npn \__tag_seq_gput_left:Nn #1 #2
47   {
48     \seq_gput_left:Nn #1 { #2 }
49     \lua_now:e { table.insert(ltx.__tag.tables['\cs_to_str:N#1'], "#2") }
50   }
51
52 %Hm not quite sure about the naming
53 \cs_set:Npn \__tag_seq_item:cn #1 #2
54   {
55     \lua_now:e { tex.sprint(\int_use:N\c_document_cctab,ltx.__tag.tables['#1']["#2"]) }
56   }
57
58 \cs_set:Npn \__tag_prop_item:cn #1 #2
59   {
60     \lua_now:e { tex.sprint(\int_use:N\c_document_cctab,ltx.__tag.tables['#1']["#2"]) }
61   }
62
63 %for debugging commands that show both the seq/prop and the lua tables
64 \cs_set_protected:Npn \__tag_seq_show:N #1
65   {
66     \seq_show:N #1
67     \lua_now:e { ltx.__tag.trace.log ("lua~sequence~array~\cs_to_str:N#1",1) }
68     \lua_now:e { ltx.__tag.trace.show_seq (ltx.__tag.tables['\cs_to_str:N#1']) }
69   }
70
71 \cs_set_protected:Npn \__tag_prop_show:N #1
72   {
73     \prop_show:N #1
74     \lua_now:e {ltx.__tag.trace.log ("lua~property~table~\cs_to_str:N#1",1) }
75     \lua_now:e {ltx.__tag.trace.show_prop (ltx.__tag.tables['\cs_to_str:N#1']) }
76   }
77
78 \cs_set_protected:Npn \__tag_prop_log:N #1
79   {
80     \prop_log:N #1
81     \lua_now:e {ltx.__tag.trace.log ("lua~property~table~\cs_to_str:N#1",1) }
82     \lua_now:e {ltx.__tag.trace.show_prop (ltx.__tag.tables['\cs_to_str:N#1']) }

```

```
83 }
```

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

```
84 </luatex>
```

The module declaration

```
85 <*lua>
```

```
86 -- tagpdf.lua
```

```
87 -- Ulrike Fischer
```

```
88
```

```
89 local ProvidesLuaModule = {
```

```
90     name          = "tagpdf",
```

```
91     version       = "1.0h",      --TAGVERSION
```

```
92     date          = "2026-10-08", --TAGDATE
```

```
93     description   = "tagpdf lua code",
```

```
94     license       = "The LATEX Project Public License 1.3c"
```

```
95 }
```

```
96
```

```
97 if luatexbase and luatexbase.provides_module then
```

```
98     luatexbase.provides_module (ProvidesLuaModule)
```

```
99 end
```

```
100
```

```
101 --[[
```

```
102 The code has quite probably a number of problems
```

```
103 - more variables should be local instead of global
```

```
104 - the naming is not always consistent due to the development of the code
```

```
105 - the traversing of the shipout box must be tested with more complicated setups
```

```
106 - it should probably handle more node types
```

```
107 -
```

```
108 --]]
```

```
109
```

Some comments about the lua structure.

```
110 --[[
```

```
111 the main table is named ltx.__tag. It contains the functions and also the data
```

```
112 collected during the compilation.
```

```
113
```

```
114 ltx.__tag.mc      will contain mc connected data.
```

```
115 ltx.__tag.role    will contain data related to parent-child relations.
```

```
116 ltx.__tag.struct  will contain structure related data.
```

```
117 ltx.__tag.page    will contain page data
```

```
118 ltx.__tag.tables  contains also data from mc and struct (from older code). This needs cleaning
```

```
119                 There are certainly dublettes, but I don't dare yet ...
```

```
120 ltx.__tag.func     will contain (public) functions.
```

```
121 ltx.__tag.trace    will contain tracing/logging functions.
```

```
122 local functions starts with __
```

```
123 functions meant for users will be in ltx.tag
```

```
124
```

```
125 functions
```

```
126 ltx.__tag.func.get_num_from (tag):   takes a tag (string) and returns the id number
```

```
127 ltx.__tag.func.output_num_from (tag): takes a tag (string) and prints (to tex) the id number
```

```
128 ltx.__tag.func.get_tag_from (num):   takes a num and returns the tag
```

```
129 ltx.__tag.func.output_tag_from (num): takes a num and prints (to tex) the tag
```

```
130 ltx.__tag.func.store_mc_data (num,key,data): stores key=data in ltx.__tag.mc[num]
```

```

131 ltx.__tag.func.store_mc_label (label,num): stores label=num in ltx.__tag.mc.labels
132 ltx.__tag.func.store_mc_kid (mcnum,kid,page): stores the mc-kids of mcnum on page page
133 ltx.__tag.func.store_mc_in_page(mcnum,mcpagencnt,page): stores in the page table the number of
134 ltx.__tag.func.store_struct_mcabs (structnum,mcnum): stores relations structnum<->mcnum (abs
135 ltx.__tag.func.mc_insert_kids (mcnum): inserts the /K entries for mcnum by wandering through
136 ltx.__tag.func.mark_page_elements(box,mcpagencnt,mccntprev,mcopen,name,mctypeprev) : the main
137 ltx.__tag.func.mark_shipout (): a wrapper around the core function which inserts the last EM
138 ltx.__tag.func.fill_parent_tree_line (page): outputs the entries of the parenttree for this
139 ltx.__tag.func.output_parenttree(): outputs the content of the parenttree
140 ltx.__tag.func.pdf_object_ref(name,index): outputs the object reference for the object name
141 ltx.__tag.func.markspaceon(), ltx.__tag.func.markspaceoff(): (de)activates the marking of po
142 ltx.__tag.func.linkbin() returns the number of OBJR stored in the linkbin structure
143 ltx.__tag.trace.show_mc_data (num,loglevel): shows ltx.__tag.mc[num] is the current log leve
144 ltx.__tag.trace.show_all_mc_data (max,loglevel): shows a maximum about mc's if the current l
145 ltx.__tag.trace.show_seq: shows a sequence (array)
146 ltx.__tag.trace.show_struct_data (num): shows data of structure num
147 ltx.__tag.trace.show_prop: shows a prop
148 ltx.__tag.trace.log
149 ltx.__tag.trace.showspace : boolean
150
151 ltx.tag.get_structnum: number, shows the current structure number
152 ltx.tag.get_structnum_next: number, shows the next structure number
153 --]]
154

```

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 `softhyphen` attribute 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.

```

155 local mctypeattributeid = luatexbase.new_attribute ("g__tag_mc_type_attr")
156 local mccntattributeid  = luatexbase.new_attribute ("g__tag_mc_cnt_attr")
157 local structnumattributeid = luatexbase.new_attribute ("g__tag_structnum_attr")
158 local iwspaceOffattributeid = luatexbase.new_attribute ("g__tag_interwordspaceOff_attr")
159 local iwspaceattributeid = luatexbase.new_attribute ("g__tag_interwordspace_attr")
160 local iwfontattributeid = luatexbase.new_attribute ("g__tag_interwordfont_attr")
161 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

```

162 local tagunmarkedbool= token.create("g__tag_tagunmarked_bool")
163 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.

```

164 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.

```

165 local catlatex      = luatexbase.registernumber("catcodetable@latex")
166 local tableinsert   = table.insert
167 local nodeid        = node.id
168 local nodecopy      = node.copy
169 local nodegetattribute = node.get_attribute
170 local nodesetattribute = node.set_attribute
171 local nodehasattribute = node.has_attribute
172 local nodenew       = node.new
173 local nodetail      = node.tail
174 local nodeslide     = node.slide
175 local noderemove    = node.remove
176 local nodetraverseid = node.traverse_id
177 local nodetraverse  = node.traverse
178 local nodeinsertafter = node.insert_after
179 local nodeinsertbefore = node.insert_before
180 local pdfpageref    = pdf.pageref
181
182 local fonthashes    = fonts.hashes
183 local identifiers   = fonthashes.identifiers
184 local fontid        = font.id
185
186 local HLIST         = node.id("hlist")
187 local VLIST         = node.id("vlist")
188 local RULE          = node.id("rule")
189 local DISC          = node.id("disc")
190 local GLUE          = node.id("glue")
191 local GLYPH         = node.id("glyph")
192 local KERN          = node.id("kern")
193 local PENALTY       = node.id("penalty")
194 local LOCAL_PAR     = node.id("local_par")
195 local MATH          = node.id("math")
196
197 local NEXT = next
198 local explicit_disc = 1
199 local regular_disc = 3

```

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

```

200 ltx      = ltx      or { }
201 ltx.tag   = ltx.tag   or { } -- user commands
202 ltx.__tag = ltx.__tag or { }
203 ltx.__tag.mc      = ltx.__tag.mc      or { } -- mc data
204 ltx.__tag.role    = ltx.__tag.role    or { } -- parent-child data
205 ltx.__tag.role.states = ltx.__tag.role.states or { } -- the states
206 ltx.__tag.role.index = ltx.__tag.role.index or { } -- standard types to index
207                                     --- numbers
208 ltx.__tag.role.matrix = ltx.__tag.role.matrix or { } -- implements the matrix
209 ltx.__tag.struct      = ltx.__tag.struct or { } -- struct data
210 ltx.__tag.tables      = ltx.__tag.tables or { } -- tables created with new prop and new seq.
211                                     -- wasn't a so great idea ...
212                                     -- g__tag_role_tags_seq used by tag<-> is in this table
213                                     -- used for pure lua tables too now!
214 ltx.__tag.page      = ltx.__tag.page      or { } -- page data, currently only i->{0->mcnum,1->mc

```

```

215 ltx.__tag.trace      = ltx.__tag.trace or { } -- show commands
216 ltx.__tag.func       = ltx.__tag.func  or { } -- functions
217 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.

```

\
218 local __tag_get_struct_num =
219   function()
220     local a = token.get_macro("g__tag_struct_stack_current_tl")
221     return a
222   end
223
224 local __tag_get_struct_counter =
225   function()
226     local a = tex.getcount("c@g__tag_struct_abs_int")
227     return a
228   end
229
230 local __tag_get_struct_num_next =
231   function()
232     local a = tex.getcount("c@g__tag_struct_abs_int") + 1
233     return a
234   end
235
236 ltx.tag.get_struct_num = __tag_get_struct_num
237 ltx.tag.get_struct_counter = __tag_get_struct_counter
238 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.

```

239 local __tag_log =
240   function (message, loglevel)
241     if (loglevel or 3) <= tex.count["l__tag_loglevel_int"] then
242       texio.write_nl("tagpdf: ".. message)
243     end
244   end
245
246 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.

```

247 function ltx.__tag.trace.show_seq (seq)

```

```

248 if (type(seq) == "table") then
249   for i,v in ipairs(seq) do
250     __tag_log ("[" .. i .. "] => " .. tostring(v),1)
251   end
252 else
253   __tag_log ("sequence " .. tostring(seq) .. " not found",1)
254 end
255 end

```

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

__tag_pairs_prop
ltx.__tag.trace.show_prop

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

```

256 local __tag_pairs_prop =
257   function (prop)
258     local a = {}
259     for n in pairs(prop) do tableinsert(a, n) end
260     table.sort(a)
261     local i = 0           -- iterator variable
262     local iter = function () -- iterator function
263       i = i + 1
264       if a[i] == nil then return nil
265       else return a[i], prop[a[i]]
266     end
267   end
268   return iter
269 end
270
271
272 function ltx.__tag.trace.show_prop (prop)
273 if (type(prop) == "table") then
274   for i,v in __tag_pairs_prop (prop) do
275     __tag_log ("[" .. i .. "] => " .. tostring(v),1)
276   end
277 else
278   __tag_log ("prop " .. tostring(prop) .. " not found or not a table",1)
279 end
280 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

```

281 function ltx.__tag.trace.show_mc_data (num,loglevel)
282 if ltx.__tag and ltx.__tag.mc and ltx.__tag.mc[num] then
283   for k,v in pairs(ltx.__tag.mc[num]) do
284     __tag_log ("mc"..num..": "..tostring(k)..=>"..tostring(v),loglevel)
285   end
286   if ltx.__tag.mc[num]["kids"] then
287     __tag_log ("mc" .. num .. " has " .. #ltx.__tag.mc[num]["kids"] .. " kids",loglevel)
288     for k,v in ipairs(ltx.__tag.mc[num]["kids"]) do
289       __tag_log ("mc " .. num .. " kid "..k.." =>" .. v.kid.." on page " ..v.page,loglevel)
290     end
291   end
292 else

```

```

293 __tag_log ("mc"..num.." not found",loglevel)
294 end
295 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.

```

296 function ltx.__tag.trace.show_all_mc_data (min,max,loglevel)
297   for i = min, max do
298     ltx.__tag.trace.show_mc_data (i,loglevel)
299   end
300   texio.write_nl("")
301 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.

```

302 function ltx.__tag.trace.show_struct_data (num)
303   if ltx.__tag and ltx.__tag.struct and ltx.__tag.struct[num] then
304     for k,v in ipairs(ltx.__tag.struct[num]) do
305       __tag_log ("struct "..num..": "..tostring(k)..">"..tostring(v),1)
306     end
307   else
308     __tag_log ("struct "..num.." not found ",1)
309   end
310 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.

```

311 local __tag_get_mc_cnt_type_tag = function (n)
312   local mccnt      = nodegetattribute(n,mccntattributeid) or -1
313   local mctype     = nodegetattribute(n,mctypeattributeid) or -1
314   local tag        = ltx.__tag.func.get_tag_from(mctype)
315   return mccnt,mctype,tag
316 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.

```

317 local function __tag_get_mathsubtype (mathnode)
318   if mathnode.subtype == 0 then
319     subtype = "beginmath"
320   else
321     subtype = "endmath"
322   end
323   return subtype
324 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.

```
325 ltx.__tag.tables.role_tag_attribute = {}
326 ltx.__tag.tables.role_attribute_tag = {}
```

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

`ltx.__tag.func.alloctag`

```
327 local __tag_alloctag =
328   function (tag)
329     if not ltx.__tag.tables.role_tag_attribute[tag] then
330       table.insert(ltx.__tag.tables.role_attribute_tag,tag)
331       ltx.__tag.tables.role_tag_attribute[tag]=#ltx.__tag.tables.role_attribute_tag
332       __tag_log ("Add "..tag.." "..ltx.__tag.tables.role_tag_attribute[tag],3)
333     end
334   end
335 ltx.__tag.func.alloctag = __tag_alloctag
```

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

`__tag_get_num_from`

`ltx.__tag.func.get_num_from`

`ltx.__tag.func.output_num_from`

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

```
336 local __tag_get_num_from =
337   function (tag)
338     if ltx.__tag.tables.role_tag_attribute[tag] then
339       a= ltx.__tag.tables.role_tag_attribute[tag]
340     else
341       a= -1
342     end
343     return a
344   end
345
346 ltx.__tag.func.get_num_from = __tag_get_num_from
347
348 function ltx.__tag.func.output_num_from (tag)
349   local num = __tag_get_num_from (tag)
350   tex.sprint(catlatex,num)
351   if num == -1 then
352     __tag_log ("Unknown tag "..tag.." used")
353   end
354 end
```

(End of definition for `__tag_get_num_from`, `ltx.__tag.func.get_num_from`, and `ltx.__tag.func.output_num_from`.)

`__tag_get_tag_from`

`ltx.__tag.func.get_tag_from`

`ltx.__tag.func.output_tag_from`

These functions are the opposites to the previous function: they take as argument a number (the attribute value) and return the string `tag`. The first function outputs the string for lua, while the `output` function outputs to tex.

```
355 local __tag_get_tag_from =
356   function (num)
357     if ltx.__tag.tables.role_attribute_tag[num] then
358       a = ltx.__tag.tables.role_attribute_tag[num]
```

```

359     else
360       a= "UNKNOWN"
361     end
362     return a
363   end
364
365   ltx.__tag.func.get_tag_from = __tag_get_tag_from
366
367   function ltx.__tag.func.output_tag_from (num)
368     tex.sprint(catlatex,__tag_get_tag_from (num))
369   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.

```

370   function ltx.__tag.func.store_mc_data (num,key,data)
371     ltx.__tag.mc[num] = ltx.__tag.mc[num] or { }
372     ltx.__tag.mc[num][key] = data
373     __tag_log ("INFO TEX-STORE-MC-DATA: "..num.." => "..tostring(key).. " => "..tostring(data),3)
374   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.

```

375   function ltx.__tag.func.store_mc_label (label,num)
376     ltx.__tag.mc["labels"] = ltx.__tag.mc["labels"] or { }
377     ltx.__tag.mc.labels[label] = num
378   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.

```

379   function ltx.__tag.func.store_mc_kid (mcnum,kid,page)
380     __tag_log("INFO TAG-STORE-MC-KID: "..mcnum.." => " .. kid.." on page " .. page,3)
381     ltx.__tag.mc[mcnum]["kids"] = ltx.__tag.mc[mcnum]["kids"] or { }
382     local kidtable = {kid=kid,page=page}
383     tableinsert(ltx.__tag.mc[mcnum]["kids"], kidtable )
384   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.

```

385   function ltx.__tag.func.mc_num_of_kids (mcnum)
386     local num = 0
387     if ltx.__tag.mc[mcnum] and ltx.__tag.mc[mcnum]["kids"] then
388       num = #ltx.__tag.mc[mcnum]["kids"]
389     end
390     __tag_log ("INFO MC-KID-NUMBERS: " .. mcnum .. "has " .. num .. "KIDS",4)
391     return num
392   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
393 local __tag_backend_create_emc_node
394 if tex.outputmode == 0 then
395   if token.get_macro("c_sys_backend_str") == "dvipdfmx" then
396     function __tag_backend_create_emc_node ()
397       local emcnode = nodenew("whatsit","special")
398       emcnode.data = "pdf:code EMC"
399       return emcnode
400     end
401   else -- assume a dvips variant
402     function __tag_backend_create_emc_node ()
403       local emcnode = nodenew("whatsit","special")
404       emcnode.data = "ps:SDict begin mark /EMC pdfmark end"
405       return emcnode
406     end
407   end
408 else -- pdf mode
409   function __tag_backend_create_emc_node ()
410     local emcnode = nodenew("whatsit","pdf_literal")
411     emcnode.data = "EMC"
412     emcnode.mode=1
413     return emcnode
414   end
415 end
416
417 local function __tag_insert_emc_node (head,current)
418   local emcnode= __tag_backend_create_emc_node()
419   head = node.insert_before(head,current,emcnode)
420   return head
421 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

```
422 local __tag_backend_create_bmc_node
423 if tex.outputmode == 0 then
424   if token.get_macro("c_sys_backend_str") == "dvipdfmx" then
425     function __tag_backend_create_bmc_node (tag)
426       local bmcnode = nodenew("whatsit","special")
427       bmcnode.data = "pdf:code /"..tag.." BMC"
428       return bmcnode
429     end
430   else -- assume a dvips variant
431     function __tag_backend_create_bmc_node (tag)
432       local bmcnode = nodenew("whatsit","special")
433       bmcnode.data = "ps:SDict begin mark/"..tag.." /BMC pdfmark end"
434       return bmcnode
435     end
436   end
437 else -- pdf mode
438   function __tag_backend_create_bmc_node (tag)
439     local bmcnode = nodenew("whatsit","pdf_literal")
440     bmcnode.data = "/"..tag.." BMC"
```

```

441     bmcnode.mode=1
442     return bmcnode
443 end
444 end
445
446 local function __tag_insert_bmc_node (head,current,tag)
447     local bmcnode = __tag_backend_create_bmc_node (tag)
448     head = node.insert_before(head,current,bmcnode)
449     return head
450 end

```

(End of definition for __tag_backend_create_bmc_node and __tag_insert_bmc_node.)

__tag_backend_create_bdc_node
__tag_insert_bdc_node

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

```

451 local __tag_backend_create_bdc_node
452
453 if tex.outputmode == 0 then
454     if token.get_macro("c_sys_backend_str") == "dvipdfmx" then
455         function __tag_backend_create_bdc_node (tag,dict)
456             local bdcnode = nodenew("whatsit","special")
457             bdcnode.data = "pdf:code /"..tag.."<<"..dict..">> BDC"
458             return bdcnode
459         end
460     else -- assume a dvips variant
461         function __tag_backend_create_bdc_node (tag,dict)
462             local bdcnode = nodenew("whatsit","special")
463             bdcnode.data = "ps:SDict begin mark/"..tag.."<<"..dict..">> /BDC pdfmark end"
464             return bdcnode
465         end
466     end
467 else -- pdf mode
468     function __tag_backend_create_bdc_node (tag,dict)
469         local bdcnode = nodenew("whatsit","pdf_literal")
470         bdcnode.data = "/"..tag.."<<"..dict..">> BDC"
471         bdcnode.mode=1
472         return bdcnode
473     end
474 end
475
476 local function __tag_insert_bdc_node (head,current,tag,dict)
477     bdcnode= __tag_backend_create_bdc_node (tag,dict)
478     head = node.insert_before(head,current,bdcnode)
479     return head
480 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.

```

481 local function __tag_pdf_object_ref (name,index)
482     local object
483     if ltx.pdf.object_id then
484         object = ltx.pdf.object_id (name,index) ..' 0 R'
485     else

```

```

486     local tokename = 'c__pdf_object_..'..name..'/'..'index..'..'int'
487     object = token.create(tokename).mode ..' 0 R'
488 end
489 return object
490 end
491 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.

```

492 local function __tag_show_spacemark (head,current,color,height)
493     local markcolor = color or "1 0 0"
494     local markheight = height or 10
495     local pdfstring
496     if tex.outputmode == 0 then
497         -- ignore dvi mode for now
498     else
499         pdfstring = node.new("whatsit","pdf_literal")
500         pdfstring.data =
501             string.format("q " ..markcolor.." RG " ..markcolor.." rg 0.4 w 0 %g m 0 %g l S Q",-
3,markheight)
502         head = node.insert_after(head,current,pdfstring)
503     return head
504 end
505 end
(End of definition for __tag_show_spacemark.)

```

`__tag_fakespace` This is used to define a lua version of `\pdf_fakespace`

```

ltx.__tag.func.fakespace 506 local function __tag_fakespace()
507     tex.setattribute(iwspaceattributeid,1)
508     tex.setattribute(iwfontattributeid,font.current())
509 end
510 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.

```

511 --[[ a function to mark up places where real space chars should be inserted
512     it only sets an attribute.
513 --]]
514
515 local function __tag_mark_spaces (head)
516     local inside_math = false
517     for n in nodetraverse(head) do
518         local id = n.id
519         if id == GLYPH then
520             local glyph = n
521             default_currfontid = glyph.font

```

```

522     if glyph.next and (glyph.next.id == GLUE)
523         and not inside_math and (glyph.next.width >0)
524     then
525         nodesetattribute(glyph.next,iwspaceattributeid,1)
526         nodesetattribute(glyph.next,iwfontattributeid,glyph.font)
527     -- for debugging
528     if ltx.__tag.trace.showspace then
529         __tag_show_spacemark (head,glyph)
530     end
531     elseif glyph.next and (glyph.next.id==KERN) and not inside_math then
532         local kern = glyph.next
533         if kern.next and (kern.next.id== GLUE) and (kern.next.width >0)
534             -- the attribute is also set on the kern in case the kern+glue is
535             -- discarded at a line break tagging issue #1102
536             -- TODO iterate back through all discardable nodes.
537         then
538             nodesetattribute(kern,iwspaceattributeid,1)
539             nodesetattribute(kern,iwfontattributeid,glyph.font)
540             nodesetattribute(kern.next,iwspaceattributeid,1)
541             nodesetattribute(kern.next,iwfontattributeid,glyph.font)
542         end
543     end
544     -- look also back
545     if glyph.prev and (glyph.prev.id == GLUE)
546         and not inside_math
547         and (glyph.prev.width >0)
548         and not nodehasattribute(glyph.prev,iwspaceattributeid)
549     then
550         nodesetattribute(glyph.prev,iwspaceattributeid,1)
551         nodesetattribute(glyph.prev,iwfontattributeid,glyph.font)
552     -- for debugging
553     if ltx.__tag.trace.showspace then
554         __tag_show_spacemark (head,glyph)
555     end
556     end
557     elseif id == PENALTY then
558         local glyph = n
559         -- __tag_log ("PENALTY ".. n.subtype.."VALUE"..n.penalty,3)
560         if glyph.next and (glyph.next.id == GLUE)
561             and not inside_math and (glyph.next.width >0) and n.subtype==0
562         then
563             nodesetattribute(glyph.next,iwspaceattributeid,1)
564             -- changed 2024-01-18, issue #72
565             nodesetattribute(glyph.next,iwfontattributeid,default_currfontid)
566         -- for debugging
567         if ltx.__tag.trace.showspace then
568             __tag_show_spacemark (head,glyph)
569         end
570     end
571     elseif id == MATH then
572         inside_math = (n.subtype == 0)
573     end
574 end
575 return head

```

```
576 end
(End of definition for __tag_mark_spaces.)
```

```
__tag_activate_mark_space These functions add/remove the function which marks the spaces to the callbacks
ltx.__tag.func.markspaceon pre_linebreak_filter and hpack_filter
ltx.__tag.func.markspaceoff

577 local function __tag_activate_mark_space ()
578   if not luatexbase.in_callback ("pre_linebreak_filter", "markspaces") then
579     luatexbase.add_to_callback("pre_linebreak_filter", __tag_mark_spaces, "markspaces")
580     luatexbase.add_to_callback("hpack_filter", __tag_mark_spaces, "markspaces")
581   end
582 end
583
584 ltx.__tag.func.markspaceon=__tag_activate_mark_space
585
586 local function __tag_deactivate_mark_space ()
587   if luatexbase.in_callback ("pre_linebreak_filter", "markspaces") then
588     luatexbase.remove_from_callback("pre_linebreak_filter", "markspaces")
589     luatexbase.remove_from_callback("hpack_filter", "markspaces")
590   end
591 end
592
593 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.

```
594 local default_space_char = nodenew(GLYPH)
595 local default_fontid      = fontid("TU/lmr/m/n/10")
596 local default_currfontid = fontid("TU/lmr/m/n/10")
597 default_space_char.char   = 32
598 default_space_char.font   = default_fontid
```

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

```
599 local function __tag_font_has_space (fontid)
600   t= fonts.hashes.identifiers[fontid]
```

test if t is nil, see <https://github.com/latex3/tagging-project/issues/1494>

```
601   if not t then return false end
602   if luaotfload.aux.slot_of_name(fontid, "space")
603     or t and t.characters and t.characters[32] and t.characters[32]["unicode"]==32
604   then
605     return true
606   else
607     return false
608   end
609 end
```

```
__tag_space_chars_shipout These is the main function to insert real space chars. It inserts a glyph before every glue
ltx.__tag.func.space_chars_shipout 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.
```

```
610 local function __tag_space_chars_shipout (box)
611   local head = box.head
612   if head then
```

```

613 for n in node.traverse(head) do
614     local spaceattr = -1
615     if not nodehasattribute(n,iwspaceOffattributeid) then
616         spaceattr = nodegetattribute(n,iwspaceattributeid) or -1
617     end
618     if n.id == HLIST then -- enter the hlist
619         __tag_space_chars_shipout (n)
620     elseif n.id == VLIST then -- enter the vlist
621         __tag_space_chars_shipout (n)
622     elseif n.id == GLUE then
623         if ltx.__tag.trace.showspace and spaceattr==1 then
624             __tag_show_spacemark (head,n,"0 1 0")
625         end
626         if spaceattr==1 then
627             local space
628             local space_char = node.copy(default_space_char)
629             local curfont = nodegetattribute(n,iwfontattributeid)
630             __tag_log ("INFO SPACE-FUNCTION-FONT: ".. tostring(curfont),3)
631             if curfont and
632                 -- luaotfload.aux.slot_of_name(curfont,"space")
633                 __tag_font_has_space (curfont)
634             then
635                 space_char.font=curfont
636             end
637             head, space = node.insert_before(head, n, space_char) --
638             n.width = n.width - space.width
639             space.attr = n.attr
640         end
641     end
642 end
643 box.head = head
644 end
645 end
646
647 function ltx.__tag.func.space_chars_shipout (box)
648     __tag_space_chars_shipout (box)
649 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.

```

650 function ltx.__tag.func.mc_insert_kids (mcnum,single)
651     if ltx.__tag.mc[mcnum] then
652         __tag_log("INFO TEX-MC-INSERT-KID-TEST: " .. mcnum,4)
653         if ltx.__tag.mc[mcnum]["kids"] then
654             if #ltx.__tag.mc[mcnum]["kids"] > 1 and single==1 then
655                 tex.sprint(catlatex,"")
656             end

```

```

657 for i,kidstable in ipairs( ltx.__tag.mc[mcnum]["kids"] ) do
658   local kidnum = kidstable["kid"]
659   local kidpage = kidstable["page"]
660   local kidpageobjnum = pdfpageref(kidpage)
661   __tag_log("INFO TEX-MC-INSERT-KID: " .. mcnum ..
662             " insert KID " .. i ..
663             " with num " .. kidnum ..
664             " on page " .. kidpage.."/"..kidpageobjnum,3)
665   tex.sprint(catlatex,"<</Type /MCR /Pg "..kidpageobjnum .. " 0 R /MCID "..kidnum.. ">> "
666   end
667   if #ltx.__tag.mc[mcnum]["kids"] > 1 and single==1 then
668     tex.sprint(catlatex,"")
669   end
670 else
671   -- this is typically not a problem, e.g. empty hbox in footer/header can
672   -- trigger this warning.
673   __tag_log("WARN TEX-MC-INSERT-NO-KIDS: "..mcnum.." has no kids",2)
674   if single==1 then
675     tex.sprint(catlatex,"null")
676   end
677 end
678 else
679   __tag_log("WARN TEX-MC-INSERT-MISSING: "..mcnum.." doesn't exist",0)
680 end
681 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.

```

682 function ltx.__tag.func.store_struct_mcabs (structnum,mcnum)
683   ltx.__tag.struct[structnum]=ltx.__tag.struct[structnum] or { }
684   ltx.__tag.struct[structnum]["mc"]=ltx.__tag.struct[structnum]["mc"] or { }
685   -- a structure can contain more than one mc chunk, the content should be ordered
686   tableinsert(ltx.__tag.struct[structnum]["mc"],mcnum)
687   __tag_log("INFO TEX-MC-INTO-STRUCT: "..
688             mcnum.." inserted in struct "..structnum,3)
689   -- but every mc can only be in one structure
690   ltx.__tag.mc[mcnum]= ltx.__tag.mc[mcnum] or { }
691   ltx.__tag.mc[mcnum]["parent"] = structnum
692 end
693

```

(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.

```

694 -- pay attention: lua counts arrays from 1, tex pages from one
695 -- mcid and arrays in pdf count from 0.
696 function ltx.__tag.func.store_mc_in_page (mcnum,mcpagencnt,page)
697   ltx.__tag.page[page] = ltx.__tag.page[page] or { }
698   ltx.__tag.page[page][mcpagencnt] = mcnum
699   __tag_log("INFO TAG-MC-INTO-PAGE: page " .. page ..
700             ": inserting MCID " .. mcpagencnt .. " => " .. mcnum,3)
701 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)

```

702 local function __tag_update_mc_attributes (head,mcnum,type)
703   for n in node.traverse(head) do
704     node.set_attribute(n,mccntattributeid,mcnum)
705     node.set_attribute(n,mctypeattributeid,type)
706     if n.id == HLIST or n.id == VLIST then
707       __tag_update_mc_attributes (n.list,mcnum,type)
708     end
709   end
710   return head
711 end
712 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.

```

713 --[[
714   Now follows the core function
715   It wades through the shipout box and checks the attributes
716   ARGUMENTS
717   box: is a box,
718   mcpagecnt: num, the current page cnt of mc (should start at -1 in shipout box), needed for
719   mccntprev: num, the attribute cnt of the previous node/whatever - if different we have a
720   mcpopen: num, records if some bdc/emc is open
721   These arguments are only needed for log messages, if not present are replaced by fix strings
722   name: string to describe the box
723   mctypeprev: num, the type attribute of the previous node/whatever
724
725   there are lots of logging messages currently. Should be cleaned up in due course.
726   One should also find ways to make the function shorter.
727 --]]
728
729 function ltx.__tag.func.mark_page_elements (box,mcpagecnt,mccntprev,mcpopen,name,mctypeprev)
730   local name = name or ("SOMEBBOX")
731   local mctypeprev = mctypeprev or -1
732   local abspage = status.total_pages + 1 -- the real counter is increased
733                                         -- inside the box so one off
734                                         -- if the callback is not used. (???)
735   __tag_log ("INFO TAG-ABSPAGE: " .. abspage,3)
736   __tag_log ("INFO TAG-ARGS: pagecnt".. mcpagecnt..
737             " prev"..mccntprev ..
738             " type prev"..mctypeprev,4)
739   __tag_log ("INFO TAG-TRAVERSING-BOX: ".. tostring(name)..
740             " TYPE ".. node.type(node.getid(box)),3)
741   local head = box.head -- ShipoutBox is a vlist?
742   if head then
743     mccnthead, mctypehead,taghead = __tag_get_mc_cnt_type_tag (head)
744     __tag_log ("INFO TAG-HEAD: " ..
745               node.type(node.getid(head))..
746               " MC"..tostring(mccnthead)..

```

```

747         " => TAG " .. tostring(mctypehead)..
748         " => ".. tostring(taghead),3)
749     else
750         __tag_log ("INFO TAG-NO-HEAD: head is "..
751                 tostring(head),3)
752     end
753     for n in node.traverse(head) do
754         local mccnt, mctype, tag = __tag_get_mc_cnt_type_tag (n)
755         local spaceattr = nodegetattribute(n,iwspaceattributeid) or -1
756         __tag_log ("INFO TAG-NODE: "..
757                 node.type(node.getid(n))..
758                 " MC".. tostring(mccnt)..
759                 " => TAG ".. tostring(mctype)..
760                 " => " .. tostring(tag),3)
761         if n.id == HLIST
762         then -- enter the hlist
763             mcopy,mcpagecnt,mccntprev,mctypeprev=
764             ltx.__tag.func.mark_page_elements (n,mcpagecnt,mccntprev,mcopy,"INTERNAL HLIST",mctype)
765         elseif n.id == VLIST then -- enter the vlist
766             mcopy,mcpagecnt,mccntprev,mctypeprev=
767             ltx.__tag.func.mark_page_elements (n,mcpagecnt,mccntprev,mcopy,"INTERNAL VLIST",mctype)
768         elseif n.id == GLUE and not n.leader then -- at glue real space chars are inserted, but t
769             -- been done if the previous shipout wandering, so here it
770         elseif n.id == LOCAL_PAR then -- local_par is ignored
771         elseif n.id == PENALTY then -- penalty is ignored
772         elseif n.id == KERN then -- kern is ignored
773             __tag_log ("INFO TAG-KERN-SUBTYPE: "..
774                     node.type(node.getid(n)).." ".n.subtype,4)
775         else
776             -- math is currently only logged.
777             -- we could mark the whole as math
778             -- for inner processing the mlist_to_hlist callback is probably needed.
779             if n.id == MATH then
780                 __tag_log("INFO TAG-MATH-SUBTYPE: "..
781                         node.type(node.getid(n)).." ".__tag_get_mathsubtype(n),4)
782             end
783             -- endmath
784             __tag_log("INFO TAG-MC-COMPARE: current "..
785                     mccnt.." prev "..mccntprev,4)
786             if mccnt~=mccntprev then -- a new mc chunk
787                 __tag_log ("INFO TAG-NEW-MC-NODE: "..
788                         node.type(node.getid(n))..
789                         " MC"..tostring(mccnt)..
790                         " <=> PREVIOUS "..tostring(mccntprev),4)
791             if mcopy~=0 then -- there is a chunk open, close it (hope there is only one ...
792                 box.list=__tag_insert_emc_node (box.list,n)
793                 mcopy = mcopy - 1
794                 __tag_log ("INFO TAG-INSERT-EMC: " ..
795                         mcpagecnt .. " MOPEN = " .. mcopy,3)
796             if mcopy ~=0 then
797                 __tag_log ("WARN TAG-OPEN-MC: " .. mcopy,1)
798             end
799         end
800         if ltx.__tag.mc[mccnt] then

```

```

801     if ltx.__tag.mc[mccnt]["artifact"] then
802         __tag_log("INFO TAG-INSERT-ARTIFACT: "..
803                 tostring(ltx.__tag.mc[mccnt]["artifact"]),3)
804         if ltx.__tag.mc[mccnt]["artifact"] == "" then
805             box.list = __tag_insert_bmc_node (box.list,n,"Artifact")
806         else
807             box.list = __tag_insert_bdc_node (box.list,n,"Artifact", "/Type /"..ltx.__tag.mc[mccnt]["artifact"])
808         end
809     else
810         __tag_log("INFO TAG-INSERT-TAG: "..
811                 tostring(tag),3)
812         mcpagecnt = mcpagecnt + 1
813         __tag_log ("INFO TAG-INSERT-BDC: "..mcpagecnt,3)
814         local dict= "/MCID "..mcpagecnt
815         if ltx.__tag.mc[mccnt]["raw"] then
816             __tag_log("INFO TAG-USE-RAW: "..
817                     tostring(ltx.__tag.mc[mccnt]["raw"]),3)
818             dict= dict .. " " .. ltx.__tag.mc[mccnt]["raw"]
819         end
820         if ltx.__tag.mc[mccnt]["alt"] then
821             __tag_log("INFO TAG-USE-ALT: "..
822                     tostring(ltx.__tag.mc[mccnt]["alt"]),3)
823             dict= dict .. " " .. ltx.__tag.mc[mccnt]["alt"]
824         end
825         if ltx.__tag.mc[mccnt]["lang"] then
826             __tag_log("INFO TAG-USE-LANG: "..
827                     tostring(ltx.__tag.mc[mccnt]["lang"]),3)
828             dict= dict .. " " .. ltx.__tag.mc[mccnt]["lang"]
829         end
830         if ltx.__tag.mc[mccnt]["actualtext"] then
831             __tag_log("INFO TAG-USE-ACTUALTEXT: "..
832                     tostring(ltx.__tag.mc[mccnt]["actualtext"]),3)
833             dict= dict .. " " .. ltx.__tag.mc[mccnt]["actualtext"]
834         end
835         box.list = __tag_insert_bdc_node (box.list,n,tag, dict)
836         ltx.__tag.func.store_mc_kid (mccnt,mcpagecnt,abspage)
837         ltx.__tag.func.store_mc_in_page(mccnt,mcpagecnt,abspage)
838         ltx.__tag.trace.show_mc_data (mccnt,3)
839     end
840     mcopen = mcopen + 1
841 else
842     if tagunmarkedbool.mode == truebool.mode then
843         __tag_log("INFO TAG-NOT-TAGGED: this has not been tagged, using artifact",2)
844         box.list = __tag_insert_bmc_node (box.list,n,"Artifact")
845         mcopen = mcopen + 1
846     else
847         __tag_log("WARN TAG-NOT-TAGGED: this has not been tagged",1)
848     end
849 end
850 mcntprev = mccnt
851 end
852 end -- end if
853 end -- end for
854 if head then

```

```

855     mccnthead, mctypehead, taghead = __tag_get_mc_cnt_type_tag (head)
856     __tag_log ("INFO TAG-ENDHEAD: " ..
857               node.type(node.getid(head))..
858               " MC"..tostring(mccnthead)..
859               " => TAG "..tostring(mctypehead)..
860               " => "..tostring(taghead),4)
861   else
862     __tag_log ("INFO TAG-ENDHEAD: ".. tostring(head),4)
863   end
864   __tag_log ("INFO TAG-QUITTING-BOX "..
865             tostring(name)..
866             " TYPE ".. node.type(node.getid(box)),4)
867   return mcopen, mcpagecnt, mccntprev, mctypeprev
868 end
869

```

(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.

```

870 function ltx.__tag.func.mark_shipout (box)
871   mcopen = ltx.__tag.func.mark_page_elements (box,-1,-100,0,"Shipout",-1)
872   if mcopen~=0 then -- there is a chunk open, close it (hope there is only one ...
873     local emcnode = __tag_backend_create_emc_node ()
874     local list = box.list
875     if list then
876       list = node.insert_after (list,node.tail(list),emcnode)
877       mcopen = mcopen - 1
878       __tag_log ("INFO SHIPOUT-INSERT-LAST-EMC: MCOPEN " .. mcopen,3)
879     else
880       __tag_log ("WARN SHIPOUT-UPS: this shouldn't happen",0)
881     end
882     if mcopen ~=0 then
883       __tag_log ("WARN SHIPOUT-MC-OPEN: " .. mcopen,1)
884     end
885   end
886 end

```

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

7 Parenttree

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

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

```

887 function ltx.__tag.func.fill_parent_tree_line (page)
888   -- we need to get page-> i=kid -> mcnum -> structnum
889   -- pay attention: the kid numbers and the page number in the parent tree start with 0!
890   local numsentry = ""
891   local pdfpage = page-1
892   if ltx.__tag.page[page] and ltx.__tag.page[page][0] then
893     mcchunks=#ltx.__tag.page[page]
894     __tag_log("INFO PARENTTREE-NUM:  page "..

```

```

895         page.." has "..mcchunks.."+1 Elements ",4)
896     for i=0,mcchunks do
897         -- what does this log??
898         __tag_log("INFO PARENTTREE-CHUNKS:  "..
899             ltx.__tag.page[page][i],4)
900     end
901     if mcchunks == 0 then
902         -- only one chunk so no need for an array
903         local mcnum = ltx.__tag.page[page][0]
904         local structnum = ltx.__tag.mc[mcnum]["parent"]
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         __tag_log("INFO PARENTTREE-STRUCT-OBJREF:  =====>"..
909             tostring(objref),5)
910         numsentry = pdfpage .. " [".. objref .. "]"
911         __tag_log("INFO PARENTTREE-NUMENTRY: page " ..
912             page.. " num entry = ".. numsentry,3)
913     else
914         numsentry = pdfpage .. " ["
915         for i=0,mcchunks do
916             local mcnum = ltx.__tag.page[page][i]
917             local structnum = ltx.__tag.mc[mcnum]["parent"] or 0
918             local propname = "g__tag_struct_"..structnum.."__prop"
919             --local objref = ltx.__tag.tables[propname]["objref"] or "XXXX"
920             local objref = __tag_pdf_object_ref('__tag/struct',structnum)
921             numsentry = numsentry .. " " .. objref
922         end
923         numsentry = numsentry .. "]"
924         __tag_log("INFO PARENTTREE-NUMENTRY: page " ..
925             page.. " num entry = ".. numsentry,3)
926     end
927     else
928         __tag_log ("INFO PARENTTREE-NO-DATA: page "..page,3)
929         numsentry = pdfpage.." []"
930     end
931     return numsentry
932 end
933
934 function ltx.__tag.func.output_parenttree (abspage)
935     for i=1,abspage do
936         line = ltx.__tag.func.fill_parent_tree_line (i) .. "^~J"
937         tex.sprint(catlatex,line)
938     end
939 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.

```

940 do
941     local properties = node.get_properties_table()
942     local is_soft_hyphen_prop = 'tagpdf.rewrite-softhyphen.is_soft_hyphen'
943     local hyphen_char = 0x2D
944     local soft_hyphen_char = 0xAD

```

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

```

945 local soffthyphen_fonts = setmetatable({}, {__index = function(t, fid)
946   local fdir = identifiers[fid]
947   local format = fdir and fdir.format
948   local result = (format == 'opentype' or format == 'truetype')
949   local characters = fdir and fdir.characters
950   result = result and (characters and characters[soft_hyphen_char]) ~= nil
951   t[fid] = result
952   return result
953 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.

```

954 local function process_soffthyphen_pre(head, _context, _dir)
955   if soffthyphenbool.mode ~= truebool.mode then return true end
956   for disc, sub in node.traverse_id(DISC, head) do
957     if sub == explicit_disc or sub == regular_disc then
958       for n, _ch, _f in node.traverse_char(disc.pre) do
959         local props = properties[n]
960         if not props then
961           props = {}
962           properties[n] = props
963         end
964         props[is_soft_hyphen_prop] = true
965       end
966     end
967   end
968   return true
969 end
970

```

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

```

971 local function process_soffthyphen_post(head, _context, _dir)
972   if soffthyphenbool.mode ~= truebool.mode then return true end
973   for disc, sub in node.traverse_id(DISC, head) do
974     for n, ch, fid in node.traverse_glyph(disc.pre) do
975       local props = properties[n]
976       if ch == hyphen_char and props and props[is_soft_hyphen_prop] then
977         if nodegetattribute(n,soffthyphenattribute) and soffthyphen_fonts[fid] then
978           n.char = soft_hyphen_char
979           props.glyph_info = nil
980         else
981           nodesetattribute(n,mctypeattributeid,-2147483647)
982           nodesetattribute(n,mccntattributeid,-2147483647)
983         end
984       end
985     end
986   end
987   return true
988 end
989
990 luatexbase.add_to_callback('pre_shaping_filter', process_soffthyphen_pre, 'tagpdf.rewrite-
soffthyphen')

```

```

991   luatexbase.add_to_callback('post_shaping_filter', process_softhyphen_post, 'tagpdf.rewrite-
softhyphen')
992 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

```

993 local function role_get_parent_child_rule (parent,child)
994   local state=
995   ltx.__tag.role.matrix[ltx.__tag.role.index[parent]]
996   and ltx.__tag.role.matrix[ltx.__tag.role.index[parent]][ltx.__tag.role.index[child]] or 0
997   return state
998 end
999 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.

```

1000 function check_update_stashed (struct,loglevel,loop)
1001   loop = (loop or 0) + 1
1002   if loop > 10 then
1003     __tag_log ('Warning: Too deeply nested stashed structures',0)
1004     return
1005   end
1006   __tag_log ('updating parentrole for stashed structure '..struct,loglevel)
1007   local parent = ltx.__tag.tables['g__tag_struct_ '..struct..'__prop']['parentnum']
1008   if parent then
1009     local ptag =
1010       string.match(ltx.__tag.tables['g__tag_struct_ '..parent..'__prop']['parentrole'], "{(.-)}{(.-)}")
1011     if ptag == 'STASHED' then
1012       -- look at the parent and update it first
1013       check_update_stashed (parent,loglevel,loop)
1014     end
1015     -- now copy the parent role from the parent
1016     ltx.__tag.tables['g__tag_struct_ '..struct..'__prop']['parentrole']
1017     =
1018     ltx.__tag.tables['g__tag_struct_ '..parent..'__prop']['parentrole']
1019     __tag_log
1020     ('new parentrole: ' .. ltx.__tag.tables['g__tag_struct_ '..struct..'__prop']['parentrole'],
1021     else
1022     __tag_log ('Warning: structure '..struct..' has no parent.',0)
1023   end
1024 end

```

```

1025
1026 function check_parent_child_rules (loglevel)
1027     texio.write_nl('\n')
1028     __tag_log ('checking parent-child rules ...' ,0)
1029     for i=2,ltx.tag.get_struct_counter() do
1030         local t,tNS=
1031             string.match(ltx.__tag.tables['g__tag_struct_'.i..'_prop']['tag'], "{(-)}{(-)
1032             }{(-)}")
1033         local r,rNS=
1034             string.match(ltx.__tag.tables['g__tag_struct_'.i..'_prop']['rolemap'], "{(-)
1035             }{(-)}{(-)}")
1036         local p,pNS=
1037             string.match(ltx.__tag.tables['g__tag_struct_'.i..'_prop']['parentrole'], "{(-)
1038             }{(-)}{(-)}")
1039         local parent=ltx.__tag.tables['g__tag_struct_'.i..'_prop']['parentnum']
1040         if parent then
1041             __tag_log (i..': '.. t..':'.tNS,loglevel)
1042             __tag_log (i..': '.. r..':'.rNS,loglevel)
1043             __tag_log (i..': '.. p..':'.pNS,loglevel)
1044             __tag_log ('parent of '..i..': '.. parent,loglevel )
1045             if p == 'STASHED' then
1046                 check_update_stashed (i,loglevel,0)
1047                 p,pNS=
1048                     string.match(ltx.__tag.tables['g__tag_struct_'.i..'_prop']['parentrole'], "{(-)
1049                     }{(-)}{(-)}")
1050             end
1051             local pt,ptNS=
1052                 string.match(ltx.__tag.tables['g__tag_struct_'.parent..'__prop']['tag'], "{(-)
1053                 }{(-)}{(-)}")
1054             local pr,prNS=
1055                 string.match(ltx.__tag.tables['g__tag_struct_'.parent..'__prop']['rolemap'], "{(-)
1056                 }{(-)}{(-)}")
1057             local pp,ppNS=
1058                 string.match(ltx.__tag.tables['g__tag_struct_'.parent..'__prop']['parentrole'], "{(-)
1059                 }{(-)}{(-)}")
1060             if pp == 'STASHED' then
1061                 check_update_stashed (parent,loglevel,0)
1062                 pp,ppNS=
1063                     string.match(ltx.__tag.tables['g__tag_struct_'.parent..'__prop']['parentrole'], "{(-)
1064                     }{(-)}{(-)}")
1065             end
1066             __tag_log (parent..'': '.. pt..':'.ptNS,loglevel)
1067             __tag_log (parent..'': '.. pr..':'.prNS,loglevel)
1068             __tag_log (parent..'': '.. pp..':'.ppNS,loglevel)
1069             -- now check the rule.
1070             -- at first rolemap of child against rolemap of parent.
1071             local state=ltx.__tag.func.role_get_parent_child_rule (pr,r)
1072             __tag_log ('rule of '..pr.."-">"..r..' is '..state,loglevel)
1073             -- if the state is 7 we check against parentrole of the parent
1074             if state == 7 then
1075                 state=ltx.__tag.func.role_get_parent_child_rule (pp,r)
1076                 __tag_log ('Parent-Child relation '..pp.."-">"..r..' is '..state,loglevel)
1077             end
1078             if state == 0 then

```

```

1071     __tag_log
1072     ('Warning: Parent-Child relation '
1073      ..ptNS..'':'..pt..' -> '..tNS..'':'..t..' is unknown',0)
1074     __tag_log
1075     ('Structure ' ..parent..' -> '..i,0)
1076 end
1077 if state == -1 then
1078     __tag_log
1079     ('Warning: Parent-Child relation '
1080      ..ptNS..'':'..pt..' -> '..tNS..'':'..t..' is not allowed',0)
1081     __tag_log
1082     ('Structure ' ..parent..' -> '..i,0)
1083 end
1084 -- check also for MC
1085 state =ltx.__tag.func.role_get_parent_child_rule ( r , 'MC')
1086 local curtag=r
1087 if state == 7 then
1088     state =ltx.__tag.func.role_get_parent_child_rule ( p , 'MC')
1089     local curtag=p
1090 end
1091 if state == -1 then
1092     if ltx.__tag.struct[i] and NEXT(ltx.__tag.struct[i]) then
1093         __tag_log
1094         ('Warning: Real content (MC) is not allowed in ' ..curtag,0)
1095     end
1096 end
1097 __tag_log('=====',loglevel)
1098 end
1099 end -- end for
1100 end
1101
1102 ltx.__tag.func.check_parent_child_rules=check_parent_child_rules
1103

```

(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.

```

1104 local linkbincnt = 0
1105
1106 function ltx.__tag.func.linkbin()
1107     return linkbincnt
1108 end
1109
1110 if luatexbase.callbacktypes['linksplit'] then
1111     luatexbase.add_to_callback('linksplit', function(start_link, position)
1112         if start_link == nil then return end
1113         local structnum =
1114             node.get_attribute(start_link,luatexbase.attributes.g__tag_structnum_attr)
1115         if structnum and structnum > -1 then

```

```

1116 local s = ltx.__tag.tables['g__tag_struct_'.structnum..'__prop']['rolemap']
1117 if s and (string.find(s,'Link') or string.find(s,'Reference')) then
1118     local struct_insert_annot_shipout = token.create__tag_struct_insert_annot_shipout:R
1119     local parentnum = tex.count['c@g__tag_parenttree_obj_int']
1120     start_link.link_attr =
1121         start_link.link_attr ..
1122         ' /LTEX_position /' .. position ..
1123         '/StructParent ' .. parentnum
1124     tex.sprint(catlatex,struct_insert_annot_shipout,'{'..
1125         structnum..'__tag_struct_'.structnum..'__prop'..'
1126         start_link.objnum..' 0 R'..'
1127         parentnum ..'}')
1128     -- the counter must be set explicitly as struct_insert_annot_shipout doesn't do it!
1129     tex.setcount('global','c@g__tag_parenttree_obj_int',parentnum +1)
1130     __tag_log(position .. " link part has object id " .. start_link.objnum .. " and str
1131 else
1132     local linkbin = token.get_macro("c__tag_struct_link_bin_tl")
1133     if linkbin then
1134         local struct_insert_annot_shipout = token.create__tag_struct_insert_annot_shipout:R
1135         local parentnum = tex.count['c@g__tag_parenttree_obj_int']
1136         start_link.link_attr = string.gsub (start_link.link_attr, "/Contents%s+<%w+>", "")
1137         start_link.link_attr =
1138             start_link.link_attr ..
1139             ' /LTEX_position /' .. position ..
1140             '/StructParent ' .. parentnum .. '/Contents (artifact)'
1141         tex.sprint(catlatex,struct_insert_annot_shipout,'{'..
1142             linkbin..'__tag_struct_'.structnum..'__prop'..'
1143             start_link.objnum..' 0 R'..'
1144             parentnum ..'}')
1145         tex.setcount('global','c@g__tag_parenttree_obj_int',parentnum +1)
1146         linkbincnt=linkbincnt+1
1147         if linkbincnt == 1 then
1148             __tag_log('Info: some Link annotation(s) are not in Link or Reference structure el
1149             __tag_log(' moved into a container at the end of the structure tree\n',0)
1150         end
1151     else
1152         __tag_log('Warning: Link not in Link or Reference structure element',0)
1153         __tag_log('OBJR not created',0)
1154         __tag_log('',0)
1155     end
1156 end
1157 end
1158 end, 'tagpdf')
1159 end
1160 </lua>

```

Ulrike Fischer
Version 1.0h, released 2026-10-08

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.

`\tag_if_tag_known:nnTF` `\tag_if_tag_known:nnTF` `{\tag}` `{\name space}` `{\true code}` `{\false code}`

Check if a specified structure tag `\tag` has been declared in the current document in name space `\name space`. In PDF 2.0 it is true if the name space exists and the tag in the name space, in PDF 1.7 it only tests if the tag exists at all, the name space argument is ignored.

<code>\tag_if_NS_known:nTF</code>	<code>\tag_if_NS_known:nTF {<NS>} {<true code>} {<false code>}</code>
-----------------------------------	---

Check if the name space has `<NS>` been declared in the current document.

<code>\tag_set_split:NNn</code>	<code>\tag_set_split:NNn {<tl var>} {<tl var>} {<tag/NS>}</code>
<code>\tag_set_split:NNe</code>	<code>\tag_gset_split:NNn {<tl var1>} {<tl var2>} {<tag/NS>}</code>
<code>\tag_gset_split:NNn</code>	This splits up pairs of names separated by a slash, e.g. <code>tag/role</code> , <code>tag/NS</code> or <code>tag</code> and stores them in the <code><tl var></code> s.
<code>\tag_gset_split:NNe</code>	

```

1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-roles-code} {2026-10-08} {1.0h}
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/dft>

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

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

9 \prop_new:N \g__tag_role_NS_prop

(End of definition for \g__tag_role_NS_prop.)

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

10 \prop_new:N \g__tag_role_index_prop

(End of definition for `\g__tag_role_index_prop`.)

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

11 `\prop_new:N \l__tag_role_debug_prop`

(End of definition for `\l__tag_role_debug_prop`.)

We need also a bunch of temporary variables.

```

\l__tag_role_tag_tmpa_tl
\l__tag_role_tag_namespace_tmpa_tl 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}{#2}{#3}
27   }
28 }
29 {
30   \cs_new_protected:Npn \__tag_role_NS_new:nnn #1 #2 #3
31   {

```

```

32     \_tag_prop_new:c { g__tag_role_NS_#1_prop }
33     \prop_new:c { g__tag_role_NS_#1_class_prop }
34     \pdf_object_new:n {tag/NS/#1}
35     \pdfdict_new:n      {g__tag_role/Namespace_#1_dict}
36     \pdf_object_new:n {__tag/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/dfl}{ }
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:nNtT {\l__tag_loglevel_int} > { 0 }
133         {
134             \msg_info:nnn { tag }{new-tag}{#1}
135         }
136     }

```

now the addition

```

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

```

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

```

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

```

(End of definition for __tag_role_add_tag:nn.)

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

_tag_role_get:nnNN

```

154 \pdf_version_compare:NnT < {2.0}
155 {
156   \cs_new_protected:Npn \_tag_role_get:nnNN #1#2#3#4 {%#1 tag, #2 NS, #3 tlv which hold th
157   {
158     \prop_get:NnNF \g__tag_role_rolemap_prop {#1}#3
159     {
160       \tl_set:Nn #3 {#1}
161     }
162     \tl_set:Nn #4 {}
163   }
164   \cs_generate_variant:Nn \_tag_role_get:nnNN {ooNN}
165 }
166
(End of definition for \_tag_role_get:nnNN.)

```

1.3.2 The pdf 2.0 version

_tag_role_add_tag:nnnn

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

```

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

```

Do not remap standard tags. TODO add warning?

```

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

```

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

```

196   \tl_if_empty:nF { #2 }

```

```

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

```

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

```

208     \bool_if:NT \l__tag_role_update_bool
209     {
210       \tl_if_empty:nF { #3 }
211       {
212         \tl_if_eq:nnF{#1}{#3}
213         {
214           \prop_get:NnN \g__tag_role_rolemap_prop {#3}\l__tag_tmpa_tl
215           \quark_if_no_value:NTF \l__tag_tmpa_tl
216           {
217             \__tag_prop_gput:Nne \g__tag_role_rolemap_prop {#1}{\tl_to_str:n{#3}}
218           }
219           {
220             \__tag_prop_gput:Nno \g__tag_role_rolemap_prop {#1}{\l__tag_tmpa_tl}
221           }
222         }
223       }
224     }
225   }
226   \cs_generate_variant:Nn \__tag_role_add_tag:nnnn {oooo}

```

(End of definition for __tag_role_add_tag:nnnn.)

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

__tag_role_get:nnNN

```

227   \pdf_version_compare:NnF < {2.0}
228   {
229     \cs_new_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.

__tag_role_read_namespace_line:nw

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

```

255 \bool_new:N\l__tag_role_update_bool
256 \bool_set_true:N \l__tag_role_update_bool
257 \pdf_version_compare:NnTF < {2.0}
258 {
259     \cs_new_protected:Npn \__tag_role_read_namespace_line:nw #1#2,#3,#4,#5,#6\q_stop %
260     % #1 NS, #2 tag, #3 rolemap, #4 NS rolemap #5 type
261     {
262         \tl_if_empty:nF { #2 }
263         {
264             \bool_if:NTF \l__tag_role_update_bool
265             {
266                 \tl_if_empty:nTF {#5}
267                 {
268                     \prop_get:NnN \g__tag_role_tags_class_prop {#3}\l__tag_tmpa_tl
269                     \quark_if_no_value:NT \l__tag_tmpa_tl
270                     {
271                         \tl_set:Nn\l__tag_tmpa_tl{--UNKNOWN--}
272                     }
273                 }
274                 {
275                     \tl_set:Nn \l__tag_tmpa_tl {#5}
276                 }
277             }
278             \__tag_role_alloctag:nno {#2} {#1} { \l__tag_tmpa_tl }
279             \tl_if_eq:nnF {#2}{#3}
280             {
281                 \__tag_role_add_tag:nn {#2}{#3}
282             }
283         }
284     }
285 }

```

```

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 namespaces 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 \IfFormatAtLeastTF{2026-11-01}
346 {
347   \file_if_exist:nTF { tagpdf-ns-latex-lab-book.def }
348   { \__tag_role_read_namespace:nn {latex-book}{latex-lab-book}}
349   { \__tag_role_read_namespace:n {latex-book}}
350 }
351 {
352   \__tag_role_read_namespace:n {latex-book}
353 }
354 \bool_set_true:N\l__tag_role_update_bool
355 %\__tag_role_read_namespace:n {latex}
356 \__tag_role_read_namespace:nn {latex} {latex-lab}
357 \__tag_role_read_namespace:n {pdf}
358 \__tag_role_read_namespace:n {pdf2}

```

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

```

359 \pdf_version_compare:NnTF < {2.0}
360 {
361   \hook_gput_code:nnn {begindocument}{tagpdf}
362   {
363     \bool_lazy_and:nnT
364     {
365       \cs_if_exist_p:N \chapter
366     }
367     {
368       \cs_if_exist_p:N \c@chapter
369     }
370   }

```

```

371         \prop_map_inline:cn{g__tag_role_NS_latex-book_prop}
372         {
373             \__tag_role_add_tag:ne {#1}{\use_i:nn #2\c_empty_tl\c_empty_tl}
374         }
375     }
376 }
377 }
378 {
379     \hook_gput_code:nnn {begindocument}{tagpdf}
380     {
381         \bool_lazy_and:nnT
382         {
383             \cs_if_exist_p:N \chapter
384         }
385         {
386             \cs_if_exist_p:N \c@chapter
387         }
388         {
389             \prop_map_inline:cn{g__tag_role_NS_latex-book_prop}
390             {
391                 \prop_gput:Nnn \g__tag_role_tags_NS_prop { #1 }{ latex-book }
392                 \__tag_prop_gput:Nne
393                 \g__tag_role_rolemap_prop {#1}{\use_i:nn #2\c_empty_tl\c_empty_tl}
394             }
395         }
396     }
397 }

```

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.

```

398 \intarray_new:Nn \g__tag_role_parent_child_intarray {6000}

```

(End of definition for `\g__tag_role_parent_child_intarray`.)

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

(End of definition for `\c__tag_role_rules_prop` and `\c__tag_role_rules_num_prop`.)

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

```

399 \sys_if_engine luatex:TF
400 {
401     \cs_new_protected:Npn \__tag_store_parent_child_rule:nnn #1 #2 #3 % num parent, num child,
402     {
403         \prop_get:NeNTF \c__tag_role_rules_prop{#3} \l__tag_tmp_unused_tl
404         {
405             \intarray_gset:Nnn \g__tag_role_parent_child_intarray
406             { #1#2 }{0\l__tag_tmp_unused_tl}
407             \lua_now:e

```

```

408         {
409             ltx.__tag.role.matrix[#1] = ltx.__tag.role.matrix[#1] or {}
410             ltx.__tag.role.matrix[#1][#2] = 0\l__tag_tmp_unused_tl
411         }
412     }
413     {
414         \intarray_gset:Nnn \g__tag_role_parent_child_intarray
415         { #1#2 }{0}
416         \lua_now:e
417         {
418             ltx.__tag.role.matrix[#1] = ltx.__tag.role.matrix[#1] or {}
419             ltx.__tag.role.matrix[#1][#2] = 0
420         }
421     }
422 }
423 }
424 {
425     \cs_new_protected:Npn \__tag_store_parent_child_rule:nnn #1 #2 #3 % num parent, num child,
426     {
427         \prop_get:NeNTF \c__tag_role_rules_prop{#3} \l__tag_tmp_unused_tl
428         {
429             \intarray_gset:Nnn \g__tag_role_parent_child_intarray
430             { #1#2 }{0\l__tag_tmp_unused_tl}
431         }
432         {
433             \intarray_gset:Nnn \g__tag_role_parent_child_intarray
434             { #1#2 }{0}
435         }
436     }
437 }

```

(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

```

438 \int_zero:N \l__tag_tmpa_int

```

Open the file depending on the PDF version

```

439 \pdf_version_compare:NnTF < {2.0}
440 {
441     \ior_open:Nn \g_tmpa_ior {tagpdf-parent-child.csv}
442 }
443 {
444     \ior_open:Nn \g_tmpa_ior {tagpdf-parent-child-2.csv}
445 }

```

Now the main loop over the file

```

446 \ior_map_inline:Nn \g_tmpa_ior
447 {

```

ignore lines containing only comments

```

448     \tl_if_empty:nF{#1}
449     {

```

count the lines ...

```
450         \int_incr:N\l__tag_tmpa_int
```

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

```
451         \seq_set_from_clist:Nn\l__tag_tmpa_seq { #1 }
452         \int_compare:nNtF {\l__tag_tmpa_int}=1
```

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

```
453     {
454         \seq_map_indexed_inline:Nn \l__tag_tmpa_seq
455         {
456             \prop_gput:Nne\g__tag_role_index_prop
457             {##2}
458             {\int_compare:nNtF{##1}<{10}{0}##1}
459         }
460     }
```

now the data lines.

```
461     {
462         \seq_set_from_clist:Nn\l__tag_tmpa_seq { #1 }
```

get the name of the child tag from the first column

```
463         \seq_pop_left:NN\l__tag_tmpa_seq\l__tag_tmpa_tl
```

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

```
464         \prop_get:NoN \g__tag_role_index_prop { \l__tag_tmpa_tl } \l__tag_tmptb_tl
```

remove column 2+3

```
465         \seq_pop_left:NN\l__tag_tmpa_seq\l__tag_tmpa_tl
466         \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.

```
467         \seq_map_indexed_inline:Nn \l__tag_tmpa_seq
468         {
469             \exp_args:Nne
470             \__tag_store_parent_child_rule:nnn {##1}{\l__tag_tmptb_tl}{ ##2 }
471         }
472     }
473 }
474 }
```

close the read handle.

```
475 \ior_close:N\g_tmpa_ior
```

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

```
476 \prop_get:NnN\g__tag_role_index_prop{StructTreeRoot}\l__tag_tmpa_tl
477 \prop_gput:Nne\g__tag_role_index_prop{Root}{\l__tag_tmpa_tl}
478 \prop_get:NnN\g__tag_role_index_prop{Hn}\l__tag_tmpa_tl
479 \pdf_version_compare:NnTF < {2.0}
480 {
481     \int_step_inline:nn{6}
482     {
```

```

483     \prop_gput:Nne\g__tag_role_index_prop{H#1}{\l__tag_tmpa_tl}
484   }
485 }
486 {
487   \int_step_inline:nn{10}
488   {
489     \prop_gput:Nne\g__tag_role_index_prop{H#1}{\l__tag_tmpa_tl}
490   }

```

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

```

491   \prop_get:NnN\g__tag_role_index_prop {mathml}\l__tag_tmpa_tl
492   \prop_get:NnN\g__tag_role_index_prop {math}\l__tag_tmpb_tl
493   \prop_get:NnN\g__tag_role_index_prop {mtext}\l__tag_tmpc_tl
494   \prop_map_inline:Nn \g__tag_role_NS_mathml_prop
495   {
496     \prop_gput:Nno\g__tag_role_index_prop {#1} {\l__tag_tmpa_tl}
497   }
498   \prop_gput:Nno\g__tag_role_index_prop{math}{\l__tag_tmpb_tl}
499   \prop_gput:Nno\g__tag_role_index_prop{mtext}{\l__tag_tmpc_tl}
500 }
501 \sys_if_engine luatex:T
502 {
503   \prop_map_inline:Nn\g__tag_role_index_prop
504   {
505     \lua_now:e { ltx.__tag.role.index['#1']=#2 }
506   }
507 }

```

1.6.2 Retrieving the parent-child rule

`\__tag_role_get_parent_child_rule:nnN`

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

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

```

508 \tl_new:N \l__tag_parent_child_check_tl
509 \sys_if_engine luatex:TF
510 {
511   \cs_new_protected:Npn \__tag_role_get_parent_child_rule:nnN #1 #2 #3
512     % #1 parent (string, standard tag after rolemapping!)
513     % #2 child (string, standard tag after rolemapping!)
514     % #3 tl for state
515   {
516     \tl_set:Nc#3
517     {
518       \lua_now:etex.print(\int_use:N\c_document_cctab,ltx.__tag.func.role_get_parent_ch
519     }

```

Debugging messages, this can perhaps go into debug mode.

```

520   \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
521   {
522     \prop_get:NoNF\c__tag_role_rules_num_prop {#3} \l__tag_tmpa_tl
523     {

```

```

524         \tl_set:Nn \l__tag_tmpa_tl {unknown}
525     }
526     \tl_set:Nn \l__tag_tmpb_tl {#1}
527     \msg_note:nneee
528     { tag }
529     { role-parent-child-result }
530     { #1 }
531     { #2 }
532     {
533         #3~(=\l__tag_tmpa_tl')
534     }
535 }
536 \int_compare:nNnT {#3} = { 0 }
537 {
538     \msg_warning:nneee
539     { tag }
540     {role-parent-child-result}
541     { #1 }
542     { #2 }
543     { unknown! }
544 }
545
546 }
547 }
548 {
549     \cs_new_protected:Npn \__tag_role_get_parent_child_rule:nnN #1 #2 #3
550     % #1 parent (string, standard tag after rolemapping)
551     % #2 child (string, standard tag after rolemapping)
552     % #3 tl for state
553     {
554
555         \prop_get:NnN \g__tag_role_index_prop{#1}\l__tag_tmpa_tl
556         \prop_get:NnN \g__tag_role_index_prop{#2}\l__tag_tmpb_tl
557         \bool_lazy_and:nnTF
558         { ! \quark_if_no_value_p:N \l__tag_tmpa_tl }
559         { ! \quark_if_no_value_p:N \l__tag_tmpb_tl }
560         {

```

Get the rule from the intarray

```

560         \tl_set:Nn#3
561         {
562             \intarray_item:Nn
563             \g__tag_role_parent_child_intarray
564             {\l__tag_tmpa_tl\l__tag_tmpb_tl}
565         }
566     }
567     {
568         \tl_set:Nn#3 {0}
569     }

```

Debugging messages, this can perhaps go into debug mode.

```

570     \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
571     {
572         \prop_get:NoNF\c__tag_role_rules_num_prop {#3} \l__tag_tmpa_tl

```

```

573         {
574             \tl_set:Nn \l__tag_tmpa_tl {unknown}
575         }
576         \tl_set:Nn \l__tag_tmpb_tl {#1}
577         \msg_note:nneee
578             { tag }
579             { role-parent-child-result }
580             { #1 }
581             { #2 }
582             {
583                 #3~(=\l__tag_tmpa_tl')
584             }
585     }
586     \int_compare:nNtT {#3} = { 0 }
587     {
588         \msg_warning:nneee
589             { tag }
590             {role-parent-child-result}
591             { #1 }
592             { #2 }
593             { unknown! }
594     }
595 }
596 }
597 \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.

```

598 \pdf_version_compare:NnTF < {2.0}
599 {
600     \cs_new_protected:Npn \__tag_role_check_parent_child:nnnnN #1 #2 #3 #4 #5
601         % #1 parent tag,% not necessarily rolemapped, but often the case
602         % #2 NS (empty in pdf 1.x)
603         % #3 child tag, % not necessarily rolemapped, but often the case
604         % #4 NS (empty in pdf 1.x)
605         % #5 tl var: to give the result back.
606     {

```

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

```

607     \prop_get:NnNTF \g__tag_role_index_prop {#1}\l__tag_tmpa_tl
608     {
609         \tl_set:Nn \l__tag_tmpa_tl {#1}
610     }
611     {
612         \prop_get:NnNF \g__tag_role_rolemap_prop {#1}\l__tag_tmpa_tl
613         {
614             \tl_set:Nn \l__tag_tmpa_tl {\q_no_value}
615         }
616     }

```

now the child

```

617     \prop_get:NnNTF \g__tag_role_index_prop {#3}\l__tag_tmpb_tl
618     {
619         \tl_set:Nn \l__tag_tmpb_tl {#3}
620     }
621     {
622         \prop_get:NnNF \g__tag_role_rolemap_prop {#3}\l__tag_tmpb_tl
623         {
624             \tl_set:Nn \l__tag_tmpb_tl {\q_no_value}
625         }
626     }

```

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

```

627     \bool_lazy_and:nnTF
628     { ! \quark_if_no_value_p:N \l__tag_tmpa_tl }
629     { ! \quark_if_no_value_p:N \l__tag_tmpb_tl }
630     {
631         \__tag_role_get_parent_child_rule:ooN
632         { \l__tag_tmpa_tl }
633         { \l__tag_tmpb_tl }
634         #5
635     }
636     {
637         \tl_set:Nn #5 {0}
638         \msg_warning:nneee
639         { tag }
640         {role-parent-child-result}
641         { #1 }
642         { #3 }
643         { unknown! }
644     }
645 }
646 }

```

and now the pdf 2.0 version

```

647 {
648     \cs_new_protected:Npn \__tag_role_check_parent_child:nnnnN #1 #2 #3 #4 #5 %tag,NS,tag,NS,
649     {
650

```

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

```

651     \tl_if_empty:nTF {#2}
652     {
653         \tl_set:Nn \l__tag_tmpa_tl {#1}
654     }
655     {
656         \prop_if_exist:cTF { g__tag_role_NS_#2_prop }
657         {
658             \prop_get:cnNTF
659             { g__tag_role_NS_#2_prop }
660             {#1}
661             \l__tag_tmpa_tl
662             {

```

```

663         \tl_set:Nn \l__tag_tmpa_tl {\tl_head:N\l__tag_tmpa_tl}
664         \tl_if_empty:NT\l__tag_tmpa_tl
665         {
666             \tl_set:Nn \l__tag_tmpa_tl {\#1}
667         }
668     }
669     {
670         \tl_set:Nn \l__tag_tmpa_tl {\q_no_value}
671     }
672 }
673 {
674     \msg_warning:nnn { tag } {role-unknown-NS} { #2}
675     \tl_set:Nn \l__tag_tmpa_tl {\q_no_value}
676 }
677 }

```

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

```

678     \tl_if_empty:nTF {#4}
679     {
680         \tl_set:Nn \l__tag_tmpb_tl {\#3}
681     }
682     {
683         \prop_if_exist:cTF { g__tag_role_NS_#4_prop }
684         {
685             \prop_get:cnNTF
686             { g__tag_role_NS_#4_prop }
687             {\#3}
688             \l__tag_tmpb_tl
689             {
690                 \tl_set:Nn \l__tag_tmpb_tl { \tl_head:N\l__tag_tmpb_tl }
691                 \tl_if_empty:NT\l__tag_tmpb_tl
692                 {
693                     \tl_set:Nn \l__tag_tmpb_tl {\#3}
694                 }
695             }
696             {
697                 \tl_set:Nn \l__tag_tmpb_tl {\q_no_value}
698             }
699         }
700         {
701             \msg_warning:nnn { tag } {role-unknown-NS} { #4}
702             \tl_set:Nn \l__tag_tmpb_tl {\q_no_value}
703         }
704     }

```

and now get the relation

```

705     \bool_lazy_and:nnTF
706     { ! \quark_if_no_value_p:N \l__tag_tmpa_tl }
707     { ! \quark_if_no_value_p:N \l__tag_tmpb_tl }
708     {
709         \__tag_role_get_parent_child_rule:ooN
710         { \l__tag_tmpa_tl }
711         { \l__tag_tmpb_tl }

```

```

712         #5
713     }
714     {
715         \tl_set:Nn #5 {0}
716         \msg_warning:nneee
717         { tag }
718         {role-parent-child-result}
719         { #2 : #1 }
720         { #4 : #3 }
721         { unknown! }
722     }
723 }
724 }
725 \cs_generate_variant:Nn\__tag_role_check_parent_child:nnnnN {oonnN,ooooN}
726 \endpackage

```

(End of definition for __tag_role_check_parent_child:nnnnN.)

\tag_check_child:nnTF

```

727 <base>\prg_new_protected_conditional:Npnn \tag_check_child:nn #1 #2 {T,F,TF}{\prg_return_true}
728 <*package>
729 \prg_set_protected_conditional:Npnn \tag_check_child:nn #1 #2 {T,F,TF} {%#1 tag, #2 NS
730 {
731     \seq_get:NN\g__tag_struct_stack_seq\l__tag_tmpa_tl
732     \__tag_struct_get_role:enNN
733     {\l__tag_tmpa_tl}
734     {rolemap}
735     \l__tag_get_parent_tmpa_tl
736     \l__tag_get_parent_tmpb_tl
737     \__tag_role_check_parent_child:oonnN
738     { \l__tag_get_parent_tmpa_tl }
739     { \l__tag_get_parent_tmpb_tl }
740     {#1}{#2}
741     \l__tag_parent_child_check_tl
742     \int_compare:nNnT {\l__tag_parent_child_check_tl} = { \c__tag_role_rule_checkparent_tl }
743     {
744         \seq_get:NN\g__tag_struct_stack_seq\l__tag_tmpa_tl
745         \__tag_struct_get_role:enNN
746         {\l__tag_tmpa_tl}
747         {parentrole}
748         \l__tag_get_parent_tmpa_tl
749         \l__tag_get_parent_tmpb_tl
750         \__tag_role_check_parent_child:oonnN
751         { \l__tag_get_parent_tmpa_tl }
752         { \l__tag_get_parent_tmpb_tl }
753         {#1}{#2}
754         \l__tag_parent_child_check_tl
755     }
756     \int_compare:nNnTF { \l__tag_parent_child_check_tl } < {0}
757     {\prg_return_false:}
758     {\prg_return_true:}
759 }

```

(End of definition for \tag_check_child:nnTF. This function is documented on page 188.)

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.

Both for the syntax tag/role and tag/NS we need a helper command to split up such a pair:

```

\tag_set_split:NNn
\tag_gset_split:NNn
760 \cs_new_protected:Npn \__tag_set_split:NNw #1#2#3/#4/#5!
761 {
762   \tl_set:Nx #1{#3}
763   \tl_set:Nx #2{#4}
764 }
765
766 \cs_new_protected:Npn \__tag_gset_split:NNw #1#2#3/#4/#5!
767 {
768   \tl_gset:Nx #1{#3}
769   \tl_gset:Nx #2{#4}
770 }
771
772 \cs_new_protected:Npn \tag_set_split:NNn #1#2#3
773 { \__tag_set_split:NNw #1#2#3//! }
774
775 \cs_new_protected:Npn \tag_gset_split:NNn #1#2#3
776 { \__tag_gset_split:NNw #1#2#3//! }
777
778 \cs_generate_variant:Nn \tag_set_split:NNn {NNe}
779 \cs_generate_variant:Nn \tag_gset_split:NNn {NNe}

```

(End of definition for `\tag_set_split:NNn` and `\tag_gset_split:NNn`. These functions are documented on page 189.)

```

tag (rolemap-key)
tag-namespace (rolemap-key)
role (rolemap-key)
role-namespace (rolemap-key)
role/new-tag (setup-key)
role/new-NS (setup-key)
add-new-tag (deprecated)
780 \keys_define:nn { __tag / tag-role }
781 {
782   ,tag .tl_set:N = \l__tag_role_tag_tmpa_tl
783   ,tag-namespace .tl_set:N = \l__tag_role_tag_namespace_tmpa_tl
784   ,role .tl_set:N = \l__tag_role_role_tmpa_tl
785   ,role-namespace .tl_set:N = \l__tag_role_role_namespace_tmpa_tl
786 }
787
788 \keys_define:nn { __tag / setup }
789 {
790   role/mathml-tags .bool_gset:N = \g__tag_role_add_mathml_bool
791   ,role/new-NS .code:n = { \__tag_role_NS_new:nnn #1 }
792   ,role/new-tag .code:n =
793   {
794     \keys_set_known:nnnN
795     {__tag/tag-role}
796     {
797       tag-namespace=user,
798       role-namespace=, %so that we can test for it.
799       #1
800     }{__tag/tag-role}\l__tag_tmpa_tl

```

```

801 \tl_if_empty:NF \l__tag_tmpa_tl
802 {
803   \tag_set_split:NNe
804     \l__tag_role_tag_tmpa_tl
805     \l__tag_role_role_tmpa_tl
806     {\l__tag_tmpa_tl}
807   %\exp_args:NNno \seq_set_split:Nnn \l__tag_tmpa_seq { / } {\l__tag_tmpa_tl}/
808   % \tl_set:Ne \l__tag_role_tag_tmpa_tl { \seq_item:Nn \l__tag_tmpa_seq {1} }
809   % \tl_set:Ne \l__tag_role_role_tmpa_tl { \seq_item:Nn \l__tag_tmpa_seq {2} }
810 }
811 \tl_if_empty:NT \l__tag_role_role_namespace_tmpa_tl
812 {
813   \prop_get:NoNTF
814     \g__tag_role_tags_NS_prop
815     { \l__tag_role_role_tmpa_tl }
816     \l__tag_role_role_namespace_tmpa_tl
817     {
818       \prop_get:NoNF
819         \g__tag_role_NS_prop
820         { \l__tag_role_role_namespace_tmpa_tl }
821         \l__tag_tmp_unused_tl
822         {
823           \tl_set:Nn \l__tag_role_role_namespace_tmpa_tl {user}
824         }
825       }
826     {
827       \tl_set:Nn \l__tag_role_role_namespace_tmpa_tl {user}
828     }
829   }
830 \pdf_version_compare:NnTF < {2.0}
831 {
832   %TODO add check for emptiness?
833   \__tag_role_add_tag:oo
834     { \l__tag_role_tag_tmpa_tl }
835     { \l__tag_role_role_tmpa_tl }
836 }
837 {
838   \__tag_role_add_tag:oooo
839     { \l__tag_role_tag_tmpa_tl }
840     { \l__tag_role_tag_namespace_tmpa_tl }
841     { \l__tag_role_role_tmpa_tl }
842     { \l__tag_role_role_namespace_tmpa_tl }
843 }
844 }
845 ,role/map-tags .choice:
846 ,role/map-tags/false .code:n = { \socket_assign_plug:nn { tag/struct/tag } {latex-
tags} }
847 ,role/map-tags/pdf .code:n = { \socket_assign_plug:nn { tag/struct/tag } {pdf-
tags} }

848 ,role/user-NS .code:n =
849 {
850   \pdf_version_compare:NnF < {2.0}
851   {

```

```

852 \pdf_string_from_unicode:nnN{utf8/string}{https://www.latex-project.org/ns/local/#1}
853 \tl_if_empty:NF \l__tag_tmpa_str
854 {
855     \pdfdict_gput:nne
856     {g__tag_role/Namespace_user_dict}
857     {NS}
858     {\l__tag_tmpa_str}
859 }
860 }
861 }

```

deprecated names

```

862 , mathml-tags .bool_gset:N = \g__tag_role_add_mathml_bool
863 , add-new-tag .meta:n = {role/new-tag={#1}}
864 }

```

(End of definition for tag (rolemap-key) and others. These functions are documented on page 188.)

\tag_if_tag_known:nTF

```

865 \prg_new_protected_conditional:Nnn \tag_if_tag_known:n { T, F, TF }
866 {
867     \prop_get:NnNTF\g__tag_role_tags_NS_prop{#1}\l__tag_tmp_unused_tl { \prg_return_true: }
868 }
869 \prg_generate_conditional_variant:Nnn \tag_if_tag_known:n {e}{T, F, TF}

```

(End of definition for \tag_if_tag_known:nTF. This function is documented on page 188.)

\tag_if_tag_known:nnTF

This tests if a tag exists in a names space. This only makes sense for PDF 2.0, in 1.7 it falls back to \tag_if_tag_known:n.

```

870 \pdf_version_compare:NnTF < {2.0}
871 {
872     \prg_new_protected_conditional:Nnn \tag_if_tag_known:nn { T, F, TF }
873     {
874         \tag_if_tag_known:nTF {#1}{ \prg_return_true: } { \prg_return_false: }
875     }
876 }
877 {
878     \prg_new_protected_conditional:Nnn \tag_if_tag_known:nn { T, F, TF }
879     {
880         \prop_if_exist:cTF { g__tag_role_NS_#2_prop }
881         {
882             \prop_get:cnNTF { g__tag_role_NS_#2_prop } {#1}\l__tag_tmp_unused_tl
883             { \prg_return_true: }
884             { \prg_return_false: }
885         }
886         {
887             \prg_return_false:
888         }
889     }
890 }
891 \prg_generate_conditional_variant:Nnn \tag_if_tag_known:nn {ee}{T, F, TF}

```

(End of definition for \tag_if_tag_known:nnTF. This function is documented on page 188.)

`\tag_if_NS_known:nTF`

892 `\prg_new_protected_conditional:Nnn \tag_if_NS_known:n { T, F, TF }`

893 `{`

894 `\prop_if_exist:cTF {g__tag_role_NS_#1_prop } { \prg_return_true: } { \prg_return_false: }`

895 `}`

(End of definition for \tag_if_NS_known:nTF. This function is documented on page 189.)

896 `\</package>`

Ulrike Fischer

Version 1.0h, released 2026-10-08

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} {2026-10-08} {1.0h}
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 `\pdfinterwordspaceoff` and `\pdfinterwordspaceon`. These commands insert what-sits and this mean they act globally. In luatex a attribute is used to this effect, for consistency this is also set globally.

The off command sets the attributes in luatex.

```

\_tag_spacechar_on: 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_interwordspaceOff_attr,
95         1
96       )
97     }
98   }
99   \cs_set_protected:Npn \tag_spacechar_on:
100   {
101     \lua_now:e
102     {
103       tex.setattribute
104       (
105         "global",
106         luatexbase.attributes.g\_tag_interwordspaceOff_attr,
107         -2147483647
108       )
109     }
110   }
111 }
112 \sys_if_engine_pdftex:T
113 {
114   \sys_if_output_pdf:T
115   {
116     \cs_set_protected:Npn \tag_spacechar_off:

```

```

117         {
118             \pdfinterwordspaceoff
119         }
120     \cs_set_protected:Npn \tag_spacechar_on:
121     {
122         \pdfinterwordspaceon
123     }
124 }
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>242</u>
activate/collect-link-artifacts (key)	1, <u>340</u>
activate/mc (key)	1, <u>242</u>
activate/socket_ (setup-key)	274
activate/softhyphen (key)	1, <u>276</u>
activate/space_ (setup-key)	213
activate/spaces (key)	1
activate/spaces_ (setup-key)	6
activate/struct (key)	1, <u>242</u>
activate/struct-dest (key)	1, <u>242</u>
activate/tagunmarked (key)	1, <u>273</u>
activate/tree (key)	1, <u>242</u>
actualtext (key)	1, <u>764</u>
actualtext_ (mc-key)	83, <u>238</u> , <u>382</u>
add-new-tag_ (deprecated)	780
add-new-tag_ (setup-key)	188
\AddToHook	13, 16, 31, 34, 44, 57, 58, 228, 240, 242, 292, 439, 441, 442, 444
\AddToHookNext	323, 637
AF (key)	1, <u>977</u>
AInline (key)	1, <u>977</u>
AInline-o (key)	1, <u>977</u>
AFref (key)	1, <u>977</u>
alt (key)	1, <u>764</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>1651</u>
attribute-class (key)	1, <u>1617</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, 277, 278, 370, 509, 536, 609
\bool_gset_true:N	36, 57, 87, 95, 175, 281, 296, 311, 311, 326, 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, 361, 370, 389, 421, 432, 478, 503, 518, 530, 548, 603, 617, 1228, 1260, 1291
\bool_if:nTF	529
\bool_lazy_all:nTF	262
\bool_lazy_and:nnTF	294, 309, 314, 324, 363, 381, 556, 627, 670, 705, 777, 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, 165, 166, 167, 176, 189, 190, 191, 222, 344, 384, 473, 502, 529, 602
\bool_set_true:N	16, 89, 91, 175, 176, 177, 200, 201, 202, 256, 354, 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_eq:NN	39
\c@g__tag_struct_abs_int	6,	\cs_new_protected:Nn	
20, 40, 58, 91, 114, 115, 118, 124,		74, 83, 127, 167, 294, 424, 428	
127, 149, 166, 226, 248, 378, 586,		\cs_new_protected:Npn	
769, 782, 827, 839, 853, 869, 884,		13, 15, 20, 21, 22, 27,	
892, 961, 972, 991, 994, 999, 1035,		30, 31, 36, 42, 43, 43, 60, 61, 61, 65,	
1037, 1042, 1054, 1056, 1061, 1152,		67, 76, 79, 80, 81, 82, 82, 83, 85, 86,	
1163, 1164, 1165, 1166, 1167, 1169,		91, 93, 95, 106, 106, 107, 108, 117,	
1171, 1177, 1182, 1189, 1192, 1202,		120, 122, 126, 137, 142, 147, 151,	
1212, 1216, 1231, 1244, 1254, 1267,		156, 163, 165, 169, 171, 172, 173,	
1270, 1285, 1286, 1288, 1299, 1644,		187, 187, 207, 211, 212, 213, 214,	
1647, 1665, 1714, 1724, 1730, 1736		215, 216, 221, 229, 233, 236, 238,	
catalog-supplemental-file (key)	1132	239, 246, 253, 258, 259, 260, 263,	
cctab commands:		264, 275, 276, 279, 283, 286, 292,	
\c_document_cctab	55, 60, 75, 144, 518	292, 293, 317, 318, 322, 327, 332,	
\chapter	199, 365, 383	334, 337, 338, 342, 342, 346, 349,	
check commands:		349, 350, 358, 362, 369, 372, 376,	
check_parent_child_rules	1000	391, 401, 407, 410, 418, 418, 425,	
check_update_stashed	1000	425, 429, 437, 440, 451, 456, 474,	
clist commands:		475, 480, 481, 482, 483, 500, 511,	
\clist_const:Nn	79, 80	514, 515, 527, 543, 548, 549, 549,	
\clist_count:n	1552	553, 556, 557, 563, 570, 590, 590,	
\clist_if_empty:NTF	1657	598, 598, 600, 605, 613, 614, 614,	
\clist_map_inline:nn		620, 648, 655, 656, 657, 670, 760,	
106, 431, 958, 1721		766, 772, 775, 897, 905, 918, 931,	
\clist_new:N	75	946, 979, 1007, 1152, 1153, 1154,	
\clist_set:Nn	1621, 1656	1353, 1394, 1447, 1458, 1467, 1487,	
\clist_use:nn	1558	1491, 1495, 1499, 1503, 1509, 1528,	
color commands:		1532, 1541, 1550, 1561, 1585, 1590	
\color_select:n	424, 435	\cs_set:Nn	568, 569, 630, 631
cs commands:		\cs_set:Npn	53, 58, 78, 98
\cs:w	1476, 1480	\cs_set_eq:NN	14, 20, 66, 78,
\cs_end:	1476, 1480	79, 80, 139, 140, 141, 142, 143, 144,	
\cs_generate_variant:Nn		145, 146, 147, 148, 149, 182, 183,	
42, 79, 98, 99, 100, 101,		194, 208, 217, 218, 223, 230, 233,	
102, 103, 104, 105, 105, 106, 107,		234, 235, 236, 371, 372, 373, 374,	
107, 113, 115, 116, 117, 126, 134,		561, 562, 563, 564, 570, 571, 575,	
151, 152, 153, 153, 154, 155, 156,		576, 577, 578, 632, 633, 661, 662, 1598	
157, 163, 164, 168, 181, 206, 226,		\cs_set_protected:Nn	
226, 253, 259, 263, 274, 275, 291,		169, 216, 265, 357, 363, 1309, 1310	
316, 326, 357, 597, 669, 717, 725,		\cs_set_protected:Npn	9, 15, 16, 22,
778, 779, 978, 1006, 1027, 1438,		29, 35, 38, 41, 46, 48, 49, 52, 62, 63,	
1445, 1457, 1508, 1570, 1599, 1600		64, 67, 71, 71, 72, 78, 83, 83, 87, 99,	
\cs_gset_eq:NN	454	102, 107, 116, 120, 143, 161, 170,	
\cs_if_exist:NTF	253, 639, 659	184, 195, 220, 228, 238, 240, 261,	
\cs_if_exist_p:N	365, 368, 383, 386	277, 295, 297, 303, 352, 356, 360,	
\cs_if_exist_use:NTF		364, 1156, 1157, 1347, 1355, 1396, 1449	
237, 239, 412, 1451		\cs_to_str:N	12,
\cs_if_free:NTF	48	18, 25, 32, 44, 49, 67, 68, 74, 75, 81, 82	
\cs_new:Nn		\cs_undefine:N	57
109, 131, 136, 291, 421, 422, 423			
\cs_new:Npn			
9, 15, 25, 117, 118, 161, 227,			
235, 237, 540, 572, 578, 584, 1441, 1571			

D

debug/log (key)	1, 260
debug/show (key)	259

`debug/structures␣(show-key)` . . . [45](#), [243](#)
`debug/uncompress (key)` [260](#)
`\DebugSocketsOn` [47](#)
`\DeclareOption` [37](#), [38](#)
dim commands:
 `\c_max_dim` [167](#), [192](#)
 `\c_zero_dim` [175](#), [176](#), [177](#)
`\directlua` [231](#)
`\documentclass` [16](#)
`\DocumentMetadata` [15](#)

E

`E (key)` [1](#), [764](#), [954](#)
`\endinput` [22](#)
`\ERRORusetaggingsocket` [95](#), [110](#)
`exclude-header-footer␣(deprecated)` [556](#)
exp commands:
 `\exp_args:Ne` [120](#), [542](#)
 `\exp_args:NNe` [84](#), [87](#), [193](#), [213](#)
 `\exp_args:Nne` [79](#), [337](#), [341](#), [425](#), [469](#)
 `\exp_args:NNno` [807](#)
 `\exp_args:No` [289](#), [324](#)
 `\exp_last_unbraced:Ne` [99](#), [102](#), [109](#)
 `\exp_last_unbraced:No` [135](#), [138](#),
 [152](#), [154](#), [157](#), [159](#), [218](#), [219](#), [238](#),
 [239](#), [628](#), [630](#), [632](#), [633](#), [641](#), [644](#), [1336](#)
 `\exp_last_unbraced:NV` [551](#)
 `\exp_not:n` [187](#), [206](#)

F

file commands:
 `\file_if_exist:nTF` [322](#), [347](#)
 `\file_input:n` [357](#)
`firstkid (key)` [1](#), [764](#)
flag commands:
 `\flag_clear:n` [237](#)
 `\flag_height:n` [137](#), [249](#)
 `\flag_new:n` [135](#)
 `\flag_raise:n` [250](#)
`\fontencoding` [6](#)
`\fontfamily` [6](#)
`\fontseries` [6](#)
`\fontshape` [6](#)
`\fontsize` [6](#)

G

group commands:
 `\group_begin:` [67](#), [76](#), [173](#),
 [309](#), [984](#), [1076](#), [1084](#), [1119](#), [1136](#), [1162](#)
 `\group_end:` [74](#), [79](#), [213](#),
 [348](#), [1002](#), [1080](#), [1090](#), [1129](#), [1147](#), [1305](#)

H

`\halign` [47](#)

hbox commands:
 `\hbox_set:Nn` [169](#), [170](#)
hook commands:
 `\hook_gput_code:nnn` [7](#), [11](#), [33](#), [57](#),
 [66](#), [80](#), [156](#), [237](#), [277](#), [278](#), [361](#), [379](#),
 [387](#), [391](#), [682](#), [689](#), [696](#), [703](#), [710](#),
 [717](#), [723](#), [730](#), [736](#), [743](#), [751](#), [764](#),
 [775](#), [788](#), [799](#), [812](#), [823](#), [836](#), [846](#), [859](#)
 `\hook_new:n` [348](#)
 `\hook_use:n` [353](#)

I

`\IfFormatAtLeastTF` [345](#)
`\IfPDFManagementActiveF` [6](#)
`\ignorespaces` [44](#)
int commands:
 `\int_abs:n` [166](#)
 `\int_case:nnTF` [88](#), [103](#), [341](#), [1552](#)
 `\int_compare:nNnTF` [22](#), [58](#),
 [70](#), [98](#), [114](#), [124](#), [125](#), [132](#), [137](#), [142](#),
 [157](#), [170](#), [171](#), [173](#), [231](#), [293](#), [351](#),
 [381](#), [400](#), [427](#), [430](#), [450](#), [452](#), [458](#),
 [458](#), [459](#), [464](#), [520](#), [536](#), [551](#), [558](#),
 [565](#), [570](#), [572](#), [586](#), [592](#), [600](#), [607](#),
 [615](#), [619](#), [622](#), [655](#), [742](#), [756](#), [1193](#), [1678](#)
 `\int_compare:nTF`
 [180](#), [498](#), [1637](#), [1639](#), [1641](#)
 `\int_compare_p:nNn` [782](#)
 `\int_decr:N` [172](#), [197](#)
 `\int_eval:n` [118](#), [136](#), [166](#),
 [197](#), [396](#), [643](#), [651](#), [779](#), [784](#), [787](#),
 [999](#), [1042](#), [1061](#), [1164](#), [1165](#), [1166](#),
 [1167](#), [1285](#), [1286](#), [1288](#), [1299](#), [1647](#)
 `\int_gincr:N` [178](#), [240](#), [286](#), [293](#),
 [336](#), [340](#), [344](#), [348](#), [354](#), [358](#), [362](#),
 [366](#), [512](#), [985](#), [1121](#), [1138](#), [1152](#), [1163](#)
 `\int_gset:Nn` [7](#), [82](#), [158](#)
 `\int_if_zero:nTF`
 [172](#), [173](#), [197](#), [198](#), [639](#), [647](#)
 `\int_incr:N` [93](#), [164](#), [188](#), [450](#)
 `\int_new:N` [6](#), [76](#), [78](#),
 [81](#), [96](#), [155](#), [160](#), [315](#), [316](#), [317](#), [318](#), [977](#)
 `\int_rand:n` [61](#), [62](#), [64](#), [66](#), [68](#), [70](#), [71](#)
 `\int_set:Nn` [261](#), [264](#), [267](#), [268](#), [269](#)
 `\int_step_inline:nn` [481](#), [487](#)
 `\int_step_inline:nnn` [25](#), [91](#), [248](#)
 `\int_step_inline:nnnn`
 [149](#), [174](#), [177](#), [200](#), [483](#), [489](#)
 `\int_to_arabic:n` [166](#), [168](#)
 `\int_to_Hex:n` [61](#), [62](#), [64](#), [66](#), [68](#), [70](#), [71](#)
 `\int_use:N` [11](#), [15](#), [20](#),
 [26](#), [35](#), [40](#), [48](#), [55](#), [55](#), [58](#), [60](#), [60](#), [66](#),
 [72](#), [75](#), [89](#), [104](#), [124](#), [131](#), [133](#), [144](#),
 [163](#), [180](#), [187](#), [206](#), [226](#), [234](#), [241](#),

243, 291, 293, 344, 378, 424, 435, 455, 456, 464, 465, 518, 527, 586, 769, 827, 839, 853, 869, 884, 892, 961, 972, 988, 991, 994, 1035, 1037, 1054, 1056, 1125, 1128, 1142, 1146, 1171, 1177, 1182, 1189, 1192, 1216, 1231, 1244, 1254, 1267, 1270, 1571, 1644, 1665, 1714, 1724, 1730, 1736	ltx.__tag.func.check_parent_- child_rules 1000
<code>\int_zero:N</code> 90, 105, 438	<code>ltx.__tag.func.fakespace</code> 506
intarray commands:	<code>ltx.__tag.func.fill_parent_tree_- line</code> 887
<code>\intarray_gset:Nnn</code> 405, 414, 429, 433, 461	<code>ltx.__tag.func.get_num_from</code> 336
<code>\intarray_item:Nn</code> 463, 466, 562	<code>ltx.__tag.func.get_tag_from</code> 355
<code>\intarray_new:Nn</code> 398, 453	<code>ltx.__tag.func.mark_page_- elements</code> 713
interwordspace _□ (deprecated) 213 , 6	<code>ltx.__tag.func.mark_shipout</code> 870
ior commands:	<code>ltx.__tag.func.markspaceoff</code> 577
<code>\ior_close:N</code> 330, 475	<code>ltx.__tag.func.markspaceon</code> 577
<code>\ior_map_inline:Nn</code> 326, 446	<code>ltx.__tag.func.mc_insert_kids</code> .. 650
<code>\ior_open:Nn</code> 324, 441, 444	<code>ltx.__tag.func.mc_num_of_kids</code> .. 385
<code>\g_tmpa_ior</code> 324, 326, 330, 441, 444, 446, 475	<code>ltx.__tag.func.output_num_from</code> . 336
iow commands:	<code>ltx.__tag.func.output_parenttree</code> 887
<code>\iow_newline:</code> 203, 303	<code>ltx.__tag.func.output_tag_from</code> . 355
<code>\iow_term:n</code> 196, 200, 203, 209, 213, 265, 355, 359, 363, 367, 371, 375, 379	<code>ltx.__tag.func.role_get_parent_- child_rule</code> 993
K	<code>ltx.__tag.func.space_chars_- shipout</code> 610
kernel internal commands:	<code>ltx.__tag.func.store_mc_data</code> ... 370
<code>__kernel_pdffdict_name:n</code> 45	<code>ltx.__tag.func.store_mc_in_page</code> 694
<code>\g__kernel_pdfmanagement_end_- run_code_tl</code> 880	<code>ltx.__tag.func.store_mc_kid</code> 379
keys commands:	<code>ltx.__tag.func.store_mc_label</code> .. 375
<code>\keys_define:nn</code> 8, 32, 34, 40, 52, 120, 132, 182, 192, 195, 235, 238, 243, 244, 280, 380, 383, 397, 412, 476, 556, 625, 764, 780, 788, 891, 954, 1028, 1093, 1115, 1132, 1602, 1617, 1652	<code>ltx.__tag.func.store_struct_- mcabs</code> 682
<code>\keys_set:nn</code> 10, 18, 18, 19, 117, 187, 190, 285, 316, 319, 338, 342, 426, 886, 1126, 1187, 1731, 1737	<code>ltx.__tag.func.update_mc_- attributes</code> 702
<code>\keys_set_known:nnnN</code> 794	<code>ltx.__tag.tables.role_tag_- attribute</code> 325
L	<code>ltx.__tag.trace.log</code> 239
<code>label (key)</code> 1 , 764	<code>ltx.__tag.trace.show_all_mc_data</code> 296
<code>\label</code> 12	<code>ltx.__tag.trace.show_mc_data</code> ... 281
<code>label_□(mc-key)</code> 84 , 238 , 382	<code>ltx.__tag.trace.show_prop</code> 256
<code>lang (key)</code> 1 , 764	<code>ltx.__tag.trace.show_seq</code> 247
<code>lang_□(mc-key=</code> 238	<code>ltx.__tag.trace.show_struct_data</code> 302
<code>\llap</code> 424	lua commands:
<code>log (deprecated) (key)</code> 260	<code>\lua_escape:n</code> 32
ltx. internal commands:	<code>\lua_now:n</code> 8, 12, 15, 18, 25, 26, 27, 32, 35, 38, 42, 44, 49, 50, 55, 58, 59, 60, 63, 67, 67, 68, 69, 73, 74, 75, 77, 81, 82, 86, 87, 87, 89, 96, 101, 109, 111, 120, 126, 133, 138, 141, 150, 158, 162, 181, 189, 230, 237, 244, 252, 268, 282, 284, 299, 303, 314, 317, 327, 329, 407, 416, 505, 518, 675
<code>ltx.__tag.func.alloctag</code> 327	M
	<code>\MakeLinkTarget</code> 114 , 153
	<code>mathml (key)</code> 1 , 977
	<code>\maxdimen</code> 190
	<code>mc-current</code> 23 , 16
	<code>mc-current_□(show-key)</code> 45 , 132

`\pdf_object_ref:n` 116, 56, 98, 131, 135,
153, 192, 318, 335, 1035, 1099, 1146
`\pdf_object_ref_indexed:nn`
57, 74, 96, 127,
211, 267, 283, 426, 447, 508, 536, 1443
`\pdf_object_ref_last:` 116,
104, 118, 124, 306, 1519, 1525, 1701
`\pdf_object_unnamed_write:nn` ...
.. 100, 111, 120, 248, 298, 1511, 1696
`\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_purify:nN` 152
`\pdf_string_from_unicode:nnN` 42, 852
`\pdf_uncompress:` 270, 272
`\pdf_version:` 241, 244, 246
`\pdf_version_compare:NnTF`
.. 20, 81, 148, 154, 171, 227, 257,
324, 359, 439, 479, 598, 830, 850, 870
`\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, 855
`\pdfdict_if_empty:nTF` 328
`\pdfdict_new:n` 18, 35, 37
`\pdfdict_put:nnn` 1077,
1078, 1085, 1086, 1087, 1120, 1137
`\pdfdict_use:n` 283, 332, 339
`\pdffakespace` 45, 303
pdffile commands:
`\pdffile_embed_file:nnn`
..... 107, 1122, 1139
`\pdffile_embed_stream:nnN` . 978, 986
`\pdffile_embed_stream:nnn` 100
`\pdfglyphtounicode` 30
`\pdfinterwordsoff` 215, 118
`\pdfinterwordsoff` 215, 34, 122
pdfmanagement commands:
`\pdfmanagement_add:nnn`
.. 52, 70, 71, 345, 347, 349, 393, 1143
`\pdfmanagement_remove:nn` 351
phoneme (key) 764
prg commands:
`\prg_do_nothing:` 39, 80, 91, 106, 371,
372, 373, 374, 454, 575, 576, 577, 578
`\prg_generate_conditional_-`
variant:Nnn 97, 869, 891
`\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`
..... 865, 872, 878, 892
`\prg_new_protected_conditional:Npnn`
..... 304, 727
`\prg_replicate:nn` 165
`\prg_return_false:` 78, 228, 256, 274,
285, 295, 298, 308, 319, 329, 524,
536, 542, 757, 867, 874, 884, 887, 894
`\prg_return_true:` 79, 227,
271, 282, 294, 307, 316, 326, 525,
535, 541, 727, 758, 867, 874, 883, 894
`\prg_set_conditional:Npnn` . 260, 278
`\prg_set_protected_conditional:Npnn`
..... 729
process commands:
`process_softhyphen_preprocess_-`
softhyphen_post 940
`\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, 464, 476, 478,
491, 492, 493, 554, 555, 621, 622, 935
`\prop_get:NnTF` 44, 94,
130, 137, 144, 158, 183, 193, 205,
213, 236, 295, 334, 344, 364, 379,
398, 403, 427, 444, 445, 522, 559,
572, 607, 612, 617, 622, 658, 685,
721, 739, 794, 813, 818, 867, 882,
907, 920, 933, 1238, 1337, 1403,
1470, 1512, 1629, 1664, 1690, 1694
`\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, 391, 446,
448, 449, 456, 477, 483, 489, 496,
498, 499, 768, 898, 900, 1287, 1298,
1374, 1419, 1587, 1592, 1633, 1701
`\prop_gremove:Nn` 37, 137, 149, 159, 901
`\prop_gset_eq:NN` 158, 1284
`\prop_gset_from_keyval:Nn` 871
`\prop_if_exist:NnTF`
..... 174, 209, 234, 280,
320, 442, 656, 683, 880, 894, 1359, 1400
`\prop_if_exist_p:N` 779

`\prop_item:Nn` 41, 99, 102, 103, 109, 115, 146, 256, 545, 1295, 1699, 1706
`\prop_log:N` 80
`\prop_map_function:NN` 254
`\prop_map_inline:Nn` 267, 272, 293, 326, 371, 389, 410, 494, 503, 884
`\prop_map_tokens:Nn` 344
`\prop_new:N` 8, 9, 10, 11, 11, 25, 33, 72, 139, 156, 870, 1165, 1580, 1583
`\prop_new_linked:N` 7, 17, 86, 91, 103, 140, 1581
`\prop_put:Nnn` 102, 188
`\prop_show:N` 73, 93, 148, 1281, 1302, 1647, 1695
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, 1360, 1364
`\providecommand` 310
`\ProvidesExplFile` 3
`\ProvidesExplPackage` 3, 3, 3, 3, 3, 3, 3, 3, 7, 20, 31, 1576

Q

`\quad` 222, 223
quark commands:
`\q_no_value` 614, 624, 670, 675, 697, 702
`\quark_if_no_value:NTF` 132, 178, 199, 215, 269, 300, 627, 638
`\quark_if_no_value_p:N` 557, 558, 628, 629, 706, 707
`\q_stop` 259, 292, 328

R

`raw_(mc-key)` 83, 238, 382
`ref (key)` 1, 764, 954
`\RemoveFromHook` 35, 342
`\renewcommand` 472, 473
`\RenewDocumentCommand` 8
`\RequirePackage` ... 40, 363, 366, 372, 375
`\rlap` 435
`role_(rolemap-key)` 188, 780
role commands:
`role_get_parent_child_rule` 993
`role-MC-child-forbidden` 104
`role-missing` 24, 86
`role-namespace_(rolemap-key)` . 188, 780
`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, 1585
`role/new-attribute*_(setup-key)` ... 117, 1585
`role/new-NS_(setup-key)` 780
`role/new-tag_(setup-key)` 780
`root-AF (key)` 1, 1093
`root-supplemental-file (key)` 1115

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, 1637, 1639, 1641, 1678
`\seq_get:NN` 550, 555, 731, 744
`\seq_get:NNTF` 481, 624, 1195, 1325, 1333
`\seq_gpop:NN` 1317, 1318
`\seq_gpop:NNTF` 106, 1319
`\seq_gpop_left:NN` 319
`\seq_gpush:Nn` 13, 16, 17, 89, 96, 1202, 1208, 1210
`\seq_gput_left:Nn` 48, 144, 285, 323, 1594
`\seq_gput_right:Nn` 43, 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, 752, 753, 803, 804, 808, 809
`\seq_log:N` 173, 197, 238, 434, 595, 610
`\seq_map_function:NN` 263
`\seq_map_indexed_inline:Nn` 454, 467
`\seq_map_inline:Nn` 289, 332, 1627, 1688
`\seq_map_pairwise_function:NNN` 1460
`\seq_new:N` 12, 12, 13, 14, 14, 15, 17, 17, 18, 19, 20, 24, 73, 74, 141, 157, 1167, 1584
`\seq_pop_left:NN` 463, 465, 466
`\seq_put_right:Nn` 333
`\seq_remove_all:Nn` 336, 566
`\seq_set_eq:NN` 205, 206
`\seq_set_from_clist:NN` ... 1622, 1659
`\seq_set_from_clist:Nn` 85, 88, 194, 214, 451, 462

\seq_set_map:NNn	567	\socket_use:nn	70, 205, 339, 817, 1263, 1381, 1426
\seq_set_map_e:NNn	1623, 1660	\socket_use:nnn	75
\seq_set_split:Nnn	51, 104, 565, 796, 800, 807	\socket_use:nw	86
\seq_show:N	66, 147, 205, 206, 239, 334, 335, 337, 518, 1214, 1282, 1303, 1313	\socket_use_expandable:n	81
\seq_use:Nn	108, 109, 203, 222, 223, 395, 568, 1638	\socket_use_expandable:nw	101
Setup keys:		stash (key)	1, 764
activate-all (deprecated)	1	stash_(mc-key)	84, 182
activate-mc (deprecated)	1	str commands:	
activate-struct (deprecated)	1	\str_case:nnTF	46, 679, 1221
activate-tree (deprecated)	1	\str_const:Nn	59
activate/all	1, 242	\str_if_eq:nnTF	117, 125, 540, 626, 691
activate/collect-link-artifacts	1, 340	\str_if_eq_p:nn	310, 531, 533
activate/mc	1, 242	\str_new:N	71
activate/softhyphen	1, 276	\str_remove_once:Nn	563, 564
activate/spaces	1	\str_set_convert:Nnnn	105, 261, 296, 396, 413, 821, 833, 847, 863, 878, 966, 1534, 1543
activate/struct	1, 242	\str_use:N	272, 309
activate/struct-dest	1, 242	\c_tilde_str	57, 59
activate/tagunmarked	1, 273	\string	15, 16
activate/tree	1, 242	struct-faulty-nesting	24, 32
catalog-supplemental-file	1132	struct-label-unknown	24, 38
debug/log	1, 260	struct-missing-tag	24, 35
debug/show	259	struct-no-objnum	24, 24
debug/uncompress	260	struct-orphan	24, 25
log (deprecated)	260	struct-Ref-unknown	42
no-struct-dest (deprecated)	1	struct-show-closing	24, 40
page/tabsorder	1, 343	struct-stack_(show-key)	45, 235
root-AF	1, 1093	struct-unknown	24, 22
root-supplemental-file	1115	struct-used-twice	24, 36
tabsorder (deprecated)	1, 343	Structure keys:	
tagunmarked (deprecated)	1, 273	actualtext	1, 764
uncompress (deprecated)	260	AF	1, 977
shipout commands:		AFinline	1, 977
\g_shipout_readonly_int	131, 131, 241, 396, 527	AFinline-o	1, 977
show-kids	24, 64	AFref	1, 977
show-spaces_(deprecated)	213, 6	alt	1, 764
show-struct	24, 64	attribute	1, 1651
\ShowTagging	21, 45, 114	attribute-class	1, 1617
skip commands:		E	1, 764, 954
\skip_horizontal:n	78	firstkid	1, 764
\c_zero_skip	78	label	1, 764
socket commands:		lang	1, 764
\socket_assign_plug:nn	200, 204, 205, 209, 210, 763, 846, 847	mathml	1, 977
\socket_if_exist:nTF	645	parent	1, 764
\socket_new:nn	183, 184, 718	phoneme	764
\socket_new_plug:nnn	185, 719, 737	ref	1, 764, 954
\socket_use:n	28, 65	stash	1, 764
		tag	1, 764
		texsource	1, 977
		title	1, 764
		title-o	1, 764
		\SuspendTagging	47

sys commands:
 \c_sys_backend_str 46
 \c_sys_engine_str 12, 14
 \sys_if_engine luatex:TF . 23, 36,
 50, 72, 83, 85, 105, 187, 223, 282,
 297, 312, 327, 355, 355, 399, 501, 509
 \sys_if_engine luatex_p: 671
 \sys_if_engine pdftex:TF ... 26, 112
 \sys_if_output_pdf:TF 11, 28, 114
 sys-no-interwordspace 24, 228

T

taborder (deprecated) (key) 1, 343
 tag (key) 1, 764
 tag_(mc-key) 83, 238, 382
 tag_(rolemap-key) 188, 780

tag commands:

\tag_attr_new:nn 117, 1585, 1598, 1600
 \tag_check_benchmark_on: 657
 \tag_check_child:nn 727, 729
 \tag_check_child:nnTF 188, 727
 \tag_get:n 20, 21, 84, 113,
 115, 132, 133, 89, 92, 235, 235, 579, 611
 \tag_get:nN 21, 132, 236, 236
 \tag_get:nnN 21, 133, 238, 238
 \tag_gset_split:NNn
 189, 723, 729, 741, 747, 760, 775, 779
 \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_NS_known:n 892
 \tag_if_NS_known:nTF 189, 892
 \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 211, 865, 869
 \tag_if_tag_known:nn ... 872, 878, 891
 \tag_if_tag_known:nnTF 188, 870
 \tag_if_tag_known:nTF .. 188, 865, 874
 \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 83, 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, 159, 195, 208, 218, 516, 545
 \tag_set_split:NNn
 189, 760, 772, 778, 803
 \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, 159, 170, 183, 212
 \tag_start:n 7, 159, 208, 216, 218
 \tag_stop: ... 7, 56, 159, 161, 182, 211
 \tag_stop:n 7, 159, 194, 215, 217
 \tag_struct_begin:n . 113, 48, 538,
 588, 611, 654, 1152, 1152, 1156, 1157
 \tag_struct_end:
 113, 26, 53, 547, 595,
 616, 664, 1152, 1153, 1309, 1310, 1350
 \tag_struct_end:n ... 113, 1154, 1347
 \tag_struct_gput:nnn
 114, 960, 1447, 1447, 1449, 1457
 \tag_struct_gput_ref:nnn 114
 \tag_struct_insert_annot:nn
 113, 156, 770,
 794, 818, 842, 865, 1561, 1561, 1570
 \tag_struct_map_function:N
 114, 1458, 1458
 \tag_struct_object_ref:n ... 113,
 911, 924, 939, 950, 1440, 1441, 1445
 \tag_struct_parent_int:
 113, 156, 760, 771, 784, 795,
 808, 819, 832, 843, 855, 866, 1561, 1571
 \tag_struct_recordtarget: ... 93, 93
 \tag_struct_use:n
 113, 115, 58, 1353, 1353, 1355

```

\tag_struct_use_num:n .....
.... 113, 234, 1394, 1394, 1396, 1438
\tag_suspend:n .....
... 7, 68, 159, 184, 194, 217, 512, 540
\tag_tool:n ..... 13, 13, 14, 16, 20
tag internal commands:
\_tag_activate_linkbin: .. 221, 240
\_tag_activate_mark_space ..... 577
\g_tag_active_mc_bool .....
..... 40, 82, 245, 252, 265, 314
\l_tag_active_mc_bool .....
..... 88, 166, 176, 190, 201, 268, 314
\l_tag_active_socket_bool ....
..... 64, 69, 74,
80, 85, 88, 100, 167, 177, 191, 202, 282
\g_tag_active_space_bool .....
..... 13, 57, 62, 82
\g_tag_active_struct_bool ....
..... 82, 247, 254, 264, 296, 324, 478
\l_tag_active_struct_bool ....
..... 88, 165, 175, 189, 200, 267, 324
\g_tag_active_struct_dest_bool
..... 82, 251, 258, 295
\g_tag_active_tree_bool .....
.... 9, 68, 82, 246, 253, 266, 351, 389
\_tag_add_missing_mcs:Nn .....
..... 97, 165, 165, 217
\_tag_add_missing_mcs_to_-
stream:Nn . 65, 65, 66, 187, 187, 223
\l_tag_attr_array_end_tl .....
..... 1651, 1674, 1681, 1685, 1711
\g_tag_attr_class_used_prop ...
..... 291, 293, 1579, 1633
\g_tag_attr_class_used_seq ....
..... 289, 1584, 1594
\g_tag_attr_entries_prop .....
..... 295, 1579,
1587, 1592, 1629, 1690, 1695, 1699
\_tag_attr_new_entry:nn .....
..... 526, 1585, 1585, 1598, 1599, 1606, 1614
\_tag_attr_new_used_entry:nn ..
..... 1590, 1610
\g_tag_attr_objref_prop .....
..... 1579, 1694, 1701, 1706
\l_tag_attr_value_tl 1579, 1672,
1673, 1680, 1684, 1703, 1709, 1716
\_tag_backend_create_bdc_node .. 451
\_tag_backend_create_bmc_node .. 422
\_tag_backend_create_emc_node .. 393
\_tag_check_add_tag_role:nn ...
..... 129, 372, 372
\_tag_check_add_tag_role:nnn ..
..... 169, 391
\_tag_check_benchmark_tic: . 356,
360, 364, 368, 372, 376, 380, 655, 661
\_tag_check_benchmark_toc: . 358,
362, 366, 370, 374, 378, 382, 656, 662
\_tag_check_forbidden_parent_-
child:nnnn ..... 120, 120, 134, 171
\_tag_check_if_active_mc: .... 312
\_tag_check_if_active_mc:TF ...
..... 85, 104,
171, 189, 218, 299, 305, 311, 359, 365
\_tag_check_if_active_struct: . 322
\_tag_check_if_active_struct:TF
..... 40, 311, 1159, 1160,
1314, 1315, 1349, 1357, 1398, 1564
\_tag_check_if_mc_in_galley: .. 521
\_tag_check_if_mc_in_galley:TF
..... 198, 219
\_tag_check_if_mc_tmb_missing: 527
\_tag_check_if_mc_tmb_missing:TF
..... 110, 207, 224, 527
\_tag_check_if_mc_tmb_missing_-
p: ..... 527
\_tag_check_if_mc_tme_missing: 538
\_tag_check_if_mc_tme_missing:TF
..... 153, 211, 228, 538
\_tag_check_if_mc_tme_missing_-
p: ..... 538
\_tag_check_info_closing_-
struct:n ..... 349, 349, 357, 1321
\_tag_check_init_mc_used: ....
..... 451, 451, 454, 460
\_tag_check_mc_if_nested: ....
..... 174, 310, 410, 410
\_tag_check_mc_if_open: .....
..... 220, 369, 410, 418
\_tag_check_mc_in_galley:TF ... 521
\_tag_check_mc_in_galley:p: ... 521
\_tag_check_mc_pushed_popped:nn
..... 90, 97, 110, 113, 118, 425, 425
\_tag_check_mc_tag:N .....
..... 193, 328, 437, 437
\_tag_check_mc_used:n .....
..... 145, 266, 456, 456
\g_tag_check_mc_used_intarray .
..... 451, 461, 463, 466
\_tag_check_no_open_struct: ...
..... 358, 358, 1323, 1331
\_tag_check_para_begin_show:nn 418
\_tag_check_para_end_show:nn .. 429
\c_tag_check_pdfversion_tl ....
..... 241, 244, 246, 247
\_tag_check_show_MCID_by_page:
..... 475, 475

```

__tag_check_struct_forbidden_- parent_child:nnn ...	137, 163, 664	__tag_get_data_struct_C:nN	557, 557
__tag_check_struct_used:n	...	__tag_get_data_struct_counter:	...
.....	362, 362, 1362		583, 584
__tag_check_structure_has_tag:n		__tag_get_data_struct_id:	572, 572
.....	332, 332, 1192	__tag_get_data_struct_num:	577, 578
__tag_check_structure_tag:N	...	__tag_get_data_struct_num:N	553, 553
.....	342, 342, 734, 761, 812	__tag_get_data_struct_tag:	540, 540
__tag_check_typeout_v:n		__tag_get_data_struct_tag:N	548, 548
.....	108, 109, 112,	__tag_get_mathsubtype	317
147, 155, 162, 200, 209, 230, 230, 265		__tag_get_mc_abs_cnt:	
__tag_check_unresolved_parent_- child:nnnn	169, 169		14, 15, 19, 20, 102,
\g__tag_css_bool ..	878, 879, 882, 893	126, 155, 166, 183, 210, 246, 254,	
\g__tag_css_prop		272, 286, 307, 321, 331, 414, 422, 442	
....	870, 871, 884, 897, 898, 900, 901	__tag_get_mc_cnt_type_tag	311
__tag_debug_mc_begin_ignore:n		__tag_get_num_from	336
.....	352, 556	\l__tag_get_parent_tmpa_tl	
__tag_debug_mc_begin_insert:n		..	60, 127, 132, 136, 139, 149, 152,
.....	307, 549	162, 165, 175, 600, 608, 621, 627,	
__tag_debug_mc_end_ignore:	377, 570	631, 634, 701, 703, 735, 738, 748, 751	
__tag_debug_mc_end_insert:	367, 563	\l__tag_get_parent_tmpb_tl	
__tag_debug_struct_begin_- ignore:n	598, 1307		60, 150,
__tag_debug_struct_begin_- insert:n	590, 1304	153, 163, 166, 175, 601, 609, 622,	
__tag_debug_struct_end_check:n		638, 642, 645, 702, 736, 739, 749, 752	
.....	620, 1349	\l__tag_get_parent_tmpc_tl	
__tag_debug_struct_end_ignore:			60, 144, 152, 154
.....	613, 1344	__tag_get_tag_from	355
__tag_debug_struct_end_insert:		\l__tag_get_tmpc_tl	64,
.....	605, 1342	193, 198, 216, 218, 219, 236, 238,	
__tag_exclude_headfoot_begin:		239, 1241, 1247, 1473, 1475, 1479, 1485	
.....	500, 561, 562	__tag_gincr_para_begin_int:	...
__tag_exclude_headfoot_end:			334, 338, 356, 372
.....	514, 563, 564	__tag_gincr_para_end_int:	
__tag_exclude_struct_headfoot_- begin:n	527, 568, 569		334, 346, 364, 374
__tag_exclude_struct_headfoot_- end:	543, 570, 571	__tag_gincr_para_main_begin_- int:	334, 334, 352, 371
__tag_fakespace	506	__tag_gincr_para_main_end_int:	334, 342, 360, 373
__tag_fakespace:	72, 74, 307	__tag_gset_split:NNw	766, 776
__tag_finish_structure:		__tag_headfoot_tagged_begin:n	
.....	13, 16, 348, 349		598, 630, 631
\l__tag_get_child_tmpa_tl		__tag_headfoot_tagged_end:	
.....	60, 605, 610, 677, 679, 689,		614, 632, 633
692, 703, 1363, 1367, 1369, 1375, 1385		__tag_hook_kernel_after_foot:	
\l__tag_get_child_tmpb_tl			483, 496, 564, 571, 578, 633
.....	60, 606, 611, 678, 690	__tag_hook_kernel_after_head:	
\l__tag_get_child_tmpc_tl			481, 488, 563, 570, 577, 632
.....	60, 145, 157, 159	__tag_hook_kernel_before_foot:	
__tag_get_data_mc_counter:	9, 9		482, 494, 562, 569, 576, 631
__tag_get_data_mc_tag:		__tag_hook_kernel_before_head:	
.....	237, 237, 291, 291		480, 486, 561, 568, 575, 630
		\g__tag_in_mc_bool	16,
		18, 175, 221, 226, 311, 370, 508,	
		509, 523, 535, 536, 553, 608, 609, 622	
		__tag_insert_bdc_node	451

__tag_insert_bmc_node	422	__tag_mc_insert_mcid_kids:n ...	131 , 131 , 150 , 321
__tag_insert_emc_node	393	__tag_mc_insert_mcid_single_	
__tag_log	239	kids:n	131 , 136 , 322
\\l__tag_loglevel_int		\\l__tag_mc_key_label_tl	
.....	81 , 125 , 132 , 170 , 171 , 261 , 264 , 267 , 268 , 269 , 351 , 381 , 400 , 428 , 431 , 458 , 520 , 551 , 558 , 565 , 570 , 572 , 592 , 600 , 607 , 615 , 619 , 622		23 , 198 , 200 , 316 , 332 , 333 , 335 , 422
__tag_mark_spaces	511	\\l__tag_mc_key_properties_tl ...	
__tag_mc_artifact_begin_marks:n			23 , 177 , 251 , 266 , 267 , 281 , 301 , 302 , 331 , 392 , 401 , 402 , 407 , 418 , 419
.....	21 , 43 , 79 , 325	\\l__tag_mc_key_stash_bool	
\\l__tag_mc_artifact_bool			20 , 29 , 38 , 184 , 203 , 337
.....	20 , 176 , 185 , 196 , 222 , 321	\\g__tag_mc_key_tag_tl	19 , 23 , 180 , 225 , 237 , 243 , 291 , 313 , 371 , 388
\\l__tag_mc_artifact_type_tl		\\l__tag_mc_key_tag_tl	23 , 179 , 193 , 195 , 224 , 242 , 312 , 328 , 330 , 332 , 387
.....	19 , 189 , 193 , 197 , 201 , 205 , 209 , 213 , 217 , 323 , 325 , 344	\\l__tag_mc_lang_tl	
__tag_mc_bdc:nn	232 , 235 , 281		22 , 185 , 190 , 314 , 319
__tag_mc_bdc_mcid:n ...	121 , 237 , 253	__tag_mc_lua_set_mc_type_attr:n	
__tag_mc_bdc_mcid:nn			83 , 83 , 107 , 195
.....	237 , 238 , 255 , 260	__tag_mc_lua_unset_mc_type_	
__tag_mc_bdc_shipout:nn ..	236 , 246	attr:	83 , 109 , 223
__tag_mc_begin_marks:nn		\\g__tag_mc_main_marks_seq	12
.....	21 , 21 , 42 , 78 , 332	\\g__tag_mc_marks	11 , 23 , 32 , 45 , 52 , 63 , 69 , 86 , 89 , 195 , 215
__tag_mc_bmc:n	232 , 233 , 277	\\g__tag_mc_multicol_marks_seq ...	12
__tag_mc_bmc_artifact:	275 , 275 , 288	\\g__tag_mc_parenttree_prop	
__tag_mc_bmc_artifact:n	275 , 279 , 289		17 , 18 , 101 , 184 , 270
\\l__tag_mc_botmarks_seq		__tag_mc_set_label_used:n	31 , 31 , 51
.....	97 , 19 , 88 , 109 , 159 , 206 , 206 , 214 , 219 , 223 , 523 , 540	\\g__tag_mc_stack_seq	
__tag_mc_check_parent_child:n .			18 , 89 , 96 , 106 , 434
.....	122 , 122 , 181 , 207 , 341	__tag_mc_store:nnn ..	91 , 91 , 105 , 132
__tag_mc_disable_marks:	76 , 76	g__tag_MCID_abs_int	7
__tag_mc_emc:	156 , 232 , 234 , 372	\\g__tag_mode_lua_bool	
__tag_mc_end_marks: ..	21 , 61 , 80 , 373		35 , 36 , 124 , 135 , 248 , 303 , 329 , 361 , 370 , 503 , 518 , 530 , 548 , 603 , 617
\\l__tag_mc_firstmarks_seq		\\l__tag_name_link_tl ...	639 , 641 , 642
.....	96 , 19 , 85 , 108 , 194 , 197 , 198 , 205 , 205 , 206 , 222 , 523 , 531 , 533	__tag_pairs_prop	256
\\g__tag_mc_footnote_marks_seq ...	12	\\l__tag_para_attr_class_tl	311
__tag_mc_get_marks: .	82 , 82 , 197 , 218	\\g__tag_para_begin_int	
__tag_mc_handle_artifact:N			311 , 340 , 358 , 424 , 459 , 464
.....	117 , 275 , 283 , 323	\\l__tag_para_bool	311 , 382 , 399 , 414 , 472 , 473 , 502 , 529 , 602
__tag_mc_handle_mc_label:n		\\g__tag_para_end_int	
.....	27 , 27 , 200 , 335		311 , 348 , 366 , 435 , 459 , 465
__tag_mc_handle_mcid:nn		\\l__tag_para_flattened_bool	
.....	237 , 258 , 263 , 329		311 , 395 , 410
__tag_mc_handle_stash:n ..	50 , 140 , 142 , 143 , 168 , 210 , 264 , 264 , 274 , 344	\\l__tag_para_main_attr_class_tl ..	311
__tag_mc_if_in:	68 , 82 , 224 , 231	\\g__tag_para_main_begin_int	
__tag_mc_if_in:TF	68 , 87 , 224 , 412 , 420		311 , 336 , 354 , 450 , 455
__tag_mc_if_in_p:	68 , 224	\\g__tag_para_main_end_int	
__tag_mc_insert_extra_tmb:n ...			311 , 344 , 362 , 450 , 456
.....	106 , 106 , 169	__tag_para_main_store_struct: .	
__tag_mc_insert_extra_tme:n ...			376 , 376
.....	106 , 151 , 170	\\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](#)
 \l__tag_parent_child_check_tl ..
 [156](#), [157](#), [169](#), [172](#), [508](#),
 [654](#), [655](#), [662](#), [665](#), [741](#), [742](#), [754](#), [756](#)
 __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 [481](#)
 __tag_prop_gput:Nnn
 [9](#), [29](#), [89](#), [98](#), [109](#), [111](#),
 [120](#), [121](#), [128](#), [132](#), [139](#), [142](#), [146](#),
 [149](#), [151](#), [201](#), [205](#), [217](#), [220](#), [282](#),
 [285](#), [314](#), [315](#), [392](#), [1368](#), [1516](#), [1523](#)
 __tag_prop_gremove:Nn
 [35](#), [149](#), [1729](#), [1735](#)
 __tag_prop_item:Nn ... [9](#), [58](#), [139](#), [146](#)
 __tag_prop_log:N [78](#)
 __tag_prop_new:N [9](#), [9](#),
 [11](#), [19](#), [24](#), [32](#), [120](#), [139](#), [139](#), [154](#), [1164](#)
 __tag_prop_new_linked:N
 [15](#), [17](#), [139](#), [140](#)
 __tag_prop_remove:Nn [139](#)
 __tag_prop_show:N [9](#), [71](#), [139](#), [148](#), [157](#)
 \c__tag_property_mc_clist .. [79](#), [245](#)
 __tag_property_record:nn
 [29](#), [108](#), [108](#), [117](#), [241](#), [489](#), [770](#)
 __tag_property_ref_lastpage:nn
 . [83](#), [118](#), [118](#), [160](#), [174](#), [177](#), [479](#), [493](#)
 \c__tag_property_struct_clist [79](#), [772](#)
 g__tag_role/RoleMap_dict [18](#)
 \g__tag_role_add_mathml_bool ...
 [73](#), [265](#), [790](#), [862](#)
 __tag_role_add_tag:nn
 [127](#), [127](#), [153](#), [280](#), [373](#), [833](#)
 __tag_role_add_tag:nnnn
 [167](#), [167](#), [226](#), [312](#), [838](#)
 __tag_role_alloctag:nnn [81](#),
 [85](#), [95](#), [107](#), [117](#), [126](#), [141](#), [184](#), [277](#), [308](#)
 __tag_role_check_parent_-
 child:nnnnN [151](#),
 [164](#), [598](#), [600](#), [607](#), [648](#), [725](#), [737](#), [750](#)
 \l__tag_role_debug_prop [11](#)
 __tag_role_get:nnNN [154](#),
 [156](#), [164](#), [227](#), [229](#), [253](#), [754](#), [805](#), [1203](#)
 __tag_role_get_parent_child_-
 rule:nnN
 [205](#), [508](#), [511](#), [549](#), [597](#), [631](#), [709](#)
 \g__tag_role_index_prop
 [190](#), [10](#), [456](#), [464](#), [476](#),
 [477](#), [478](#), [483](#), [489](#), [491](#), [492](#), [493](#),
 [496](#), [498](#), [499](#), [503](#), [554](#), [555](#), [607](#), [617](#)
 \g__tag_role_NS_<ns>_class_prop [190](#)
 \g__tag_role_NS_<ns>_prop [190](#)
 \g__tag_role_NS_mathml_prop [267](#), [494](#)
 __tag_role_NS_new:nnn [191](#),
 [20](#), [22](#), [30](#), [74](#), [75](#), [76](#), [77](#), [78](#), [80](#), [791](#)
 \g__tag_role_NS_prop
 [189](#), [9](#), [26](#), [56](#), [193](#), [326](#), [344](#), [819](#)
 \g__tag_role_parent_child_-
 intarray [398](#), [405](#), [414](#), [429](#), [433](#), [563](#)
 __tag_role_read_namespace:n ...
 [337](#), [337](#),
 [341](#), [342](#), [343](#), [349](#), [352](#), [355](#), [357](#), [358](#)
 __tag_role_read_namespace:nn ..
 [318](#), [318](#), [339](#), [348](#), [356](#)
 __tag_role_read_namespace_-
 line:nw [255](#), [259](#), [292](#), [328](#)
 \l__tag_role_role_namespace_-
 tmpa_tl [12](#),
 [785](#), [811](#), [816](#), [820](#), [823](#), [827](#), [842](#)
 \l__tag_role_role_tmpa_tl
 [12](#), [784](#), [805](#), [809](#), [815](#), [835](#), [841](#)
 \g__tag_role_rolemap_prop
 [189](#), [18](#), [144](#), [146](#), [149](#), [158](#),
 [214](#), [217](#), [220](#), [269](#), [272](#), [393](#), [612](#), [622](#)
 \c__tag_role_rule_checkparent_tl
 [157](#), [173](#), [655](#), [742](#)
 \c__tag_role_rules_num_prop
 [399](#), [522](#), [572](#)
 \c__tag_role_rules_prop [399](#), [403](#), [427](#)
 \l__tag_role_tag_namespace_tmpa_-
 tl [12](#), [783](#), [840](#)
 \l__tag_role_tag_namespace_tmpb_-
 tl [14](#)
 \l__tag_role_tag_namespace_tmpb_-
 tl\UUUUUU% [12](#)
 \l__tag_role_tag_tmpa_tl
 [12](#), [782](#), [804](#), [808](#), [834](#), [839](#)
 \g__tag_role_tags_class_prop ...
 [189](#), [8](#), [90](#), [99](#), [112](#), [121](#), [137](#), [268](#)
 \g__tag_role_tags_NS_prop .. [189](#),
 [7](#), [88](#), [97](#), [110](#), [119](#), [130](#), [344](#), [379](#),
 [391](#), [445](#), [721](#), [739](#), [794](#), [814](#), [867](#), [1337](#)
 \l__tag_role_tmpa_seq [12](#)
 \l__tag_role_update_bool
 [208](#), [255](#), [256](#), [264](#), [344](#), [354](#)
 \c__tag_role_userNS_id_str
 [190](#), [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](#), [46](#), [144](#), [153](#), [280](#)
 __tag_seq_gput_right:Nn [9](#),
 [41](#), [139](#), [143](#), [152](#), [243](#), [253](#), [264](#), [303](#)
 __tag_seq_item:Nn ... [9](#), [53](#), [139](#), [145](#)
 __tag_seq_new:N
 [9](#), [9](#), [22](#), [121](#), [139](#), [141](#), [155](#), [1166](#)
 __tag_seq_show:N [9](#), [64](#), [139](#), [147](#), [156](#)
 __tag_set_split:NNw [760](#), [773](#)
 __tag_show_spacemark [492](#)
 \l__tag_showspace_bool ... [7](#), [16](#), [17](#)
 \g__tag_softthyphen_bool
 [94](#), [277](#), [278](#), [281](#), [296](#), [311](#), [326](#)
 __tag_space_chars_shipout [610](#)
 __tag_start_para_ints:
 [178](#), [203](#), [350](#), [350](#)
 __tag_stop_para_ints:
 [168](#), [192](#), [350](#), [369](#)
 __tag_store_parent_child_-
 rule:nnn [399](#), [401](#), [425](#), [470](#)
 g__tag_struct_1_prop [119](#)
 __tag_struct_add_AF:nn
 ... [990](#), [1007](#), [1027](#), [1034](#), [1054](#), [1099](#)
 __tag_struct_add_inline_AF:nn .
 ... [979](#), [1006](#), [1068](#), [1072](#), [1079](#), [1089](#)
 \l__tag_struct_addkid_tl [88](#), [814](#), [1278](#)
 \g__tag_struct_AFobj_int [977](#), [985](#), [988](#)
 __tag_struct_check_parent_-
 child:nn [614](#), [614](#), [669](#), [705](#), [714](#), [1265](#)
 __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](#), [920](#), [933](#)
 \l__tag_struct_elem_stash_bool .
 [87](#), [774](#), [1228](#), [1261](#), [1291](#)
 __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](#), [732](#), [745](#)
 __tag_struct_gput_data_alt:nn .
 [1532](#), [1532](#)
 __tag_struct_gput_data_attribute:nn
 [1509](#), [1509](#), [1723](#)
 __tag_struct_gput_data_C:nn ...
 [1550](#), [1550](#)
 __tag_struct_gput_data_ref:nn .
 [1487](#), [1508](#)
 __tag_struct_gput_data_ref_-
 aux:nnn [1466](#),
 [1467](#), [1489](#), [1493](#), [1497](#), [1501](#), [1505](#)
 __tag_struct_gput_data_ref_-
 dest:nn [1495](#)
 __tag_struct_gput_data_ref_-
 dest_parent:nn [1499](#)
 __tag_struct_gput_data_ref_-
 label:nn [1491](#)
 __tag_struct_gput_data_ref_-
 num:nn [1503](#)
 __tag_struct_gput_data_tag:nn .
 [1528](#), [1528](#)
 __tag_struct_gput_data_title:nn
 [1541](#), [1541](#)
 __tag_struct_insert_annot:nn ..
 [474](#), [474](#), [1566](#)
 __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](#), [1365](#), [1410](#)
 g__tag_struct_kids_1_seq [119](#)
 \g__tag_struct_label_num_prop ..
 [86](#), [768](#), [907](#)
 \l__tag_struct_lang_tl
 [478](#), [1150](#), [1174](#), [1179](#)
 \c__tag_struct_link_bin_tl [226](#), [234](#)
 __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 .	\g__tag_struct_tag_tl
. 78, 804, 807, 811, 1234	 78, 179, 180, 183,
\l__tag_struct_parenttag_tl	312, 313, 441, 442, 724, 730, 734,
. 78, 803, 806, 810, 812, 1234	742, 748, 752, 755, 759, 761, 1190,
__tag_struct_prop_gput:nnn	1204, 1209, 1335, 1337, 1379, 1424
. 105, 106, 107, 113, 123, 128, 133,	__tag_struct_use_check_parent_-
138, 143, 150, 176, 180, 189, 195,	child:nn . 670, 670, 717, 1383, 1428
200, 363, 376, 390, 826, 838, 852,	__tag_struct_write_obj 140
868, 883, 891, 971, 993, 1036, 1055,	__tag_struct_write_obj:n
1100, 1170, 1176, 1181, 1215, 1230,	 151, 440, 440
1243, 1253, 1269, 1413, 1482, 1530,	\l__tag_tag_stop_int 159, 163, 164,
1539, 1548, 1555, 1558, 1643, 1713	172, 173, 180, 187, 188, 197, 198, 206
\g__tag_struct_ref_by_dest_prop 103	\g__tag_tagunmarked_bool 93, 273, 275
__tag_struct_Ref_dest:nN . 897, 918	\l__tag_tmp_unused_tl 63, 130, 337,
__tag_struct_Ref_dest_parent:nN	344, 403, 406, 410, 427, 430, 445,
. 897, 931	561, 563, 564, 565, 721, 726, 739,
__tag_struct_Ref_label:nN 897, 905	744, 750, 794, 797, 821, 867, 882, 1690
__tag_struct_Ref_num:nN . . 897, 946	\l__tag_tmp_unused_tl\l__-
__tag_struct_Ref_obj:nN . . 897, 897	tag_get_tmppc_tl 60
\g__tag_struct_roletag_NS_tl 78	\l__tag_tmppa_box
\l__tag_struct_roletag_NS_tl	 60, 169, 175, 176, 180, 191, 192
. 81, 1207, 1219, 1257	\l__tag_tmppa_clist
\g__tag_struct_roletag_stack_seq	 60, 1621, 1622, 1656, 1657, 1659
. 15, 17, 306, 1210, 1317	\l__tag_tmppa_int 60,
\l__tag_struct_roletag_tl	90, 93, 98, 101, 105, 114, 438, 450, 452
78, 1206, 1209, 1211, 1219, 1221, 1257	\l__tag_tmppa_prop 60, 176, 189, 203, 205
__tag_struct_set_attribute:	\l__tag_tmppa_seq 51,
. 25, 39, 1213, 1328	58, 59, 60, 331, 333, 335, 336, 337,
__tag_struct_set_tag_info:nnn .	338, 451, 454, 462, 463, 465, 466,
. 171, 173, 187, 206, 1188	467, 488, 508, 518, 565, 566, 567,
\g__tag_struct_stack_current_tl	568, 752, 753, 796, 800, 803, 804,
. 18,	807, 808, 809, 1623, 1627, 1637,
27, 33, 36, 67, 73, 100, 116, 148,	1638, 1639, 1641, 1660, 1678, 1688
154, 162, 208, 268, 272, 299, 342,	\l__tag_tmppa_str 42, 43, 48, 60, 262,
545, 574, 580, 1212, 1276, 1280,	267, 272, 297, 302, 309, 397, 402,
1281, 1302, 1321, 1327, 1366, 1372,	414, 419, 822, 829, 834, 841, 848,
1378, 1384, 1411, 1417, 1423, 1429	852, 853, 855, 858, 864, 871, 879,
\l__tag_struct_stack_parent_-	886, 967, 974, 1535, 1539, 1544, 1548
tmppa_tl . . 18, 483, 492, 509, 784,	\l__tag_tmppa_tl 42, 43,
1186, 1193, 1197, 1239, 1266, 1273,	47, 49, 50, 51, 56, 60, 86, 88, 93, 94,
1277, 1279, 1282, 1294, 1295, 1303	94, 96, 102, 106, 106, 108, 109, 113,
\g__tag_struct_stack_seq 12,	114, 116, 116, 117, 137, 138, 139,
22, 25, 482, 555, 731, 744, 1196,	141, 143, 144, 146, 177, 178, 180,
1202, 1214, 1313, 1319, 1325, 1462	183, 184, 186, 191, 198, 199, 205,
\c__tag_struct_StructElem_-	205, 206, 209, 211, 214, 215, 220,
entries_seq 41	268, 269, 271, 275, 277, 288, 297,
\c__tag_struct_StructTreeRoot_-	299, 300, 302, 306, 308, 308, 314,
entries_seq 41	319, 320, 323, 359, 361, 367, 379,
\g__tag_struct_tag_NS_tl	398, 460, 463, 464, 465, 465, 466,
. 78, 725, 731, 743, 749,	476, 477, 477, 478, 483, 485, 489,
753, 756, 760, 1191, 1205, 1301, 1339	491, 496, 522, 524, 533, 554, 557,
\g__tag_struct_tag_stack_seq	564, 572, 574, 583, 607, 609, 612,
. 14, 50, 238, 239, 550,	614, 624, 628, 628, 632, 632, 653,
595, 610, 624, 1208, 1318, 1333, 1461	661, 663, 664, 666, 670, 675, 706,

710, 731, 733, 744, 746, 757, 759, 800, 801, 806, 807, 808, 810, 934, 989, 992, 1317, 1318, 1319, 1325, 1327, 1333, 1336, 1337, 1339, 1406, 1512, 1514, 1515, 1519, 1629, 1635, 1646, 1667, 1669, 1670, 1673, 1694	
\l__tag_tmpb_box	60, 170, 177, 178, 182, 184
\l__tag_tmpb_seq	60, 1622, 1623, 1659, 1660
\l__tag_tmpb_tl	202, 60, 89, 104, 118, 120, 295, 301, 444, 464, 470, 492, 498, 526, 550, 551, 555, 558, 564, 576, 617, 619, 622, 624, 629, 633, 680, 688, 690, 691, 693, 697, 702, 707, 711, 758, 760, 809, 811, 907, 911, 920, 924, 933, 936, 939
\l__tag_tmpc_tl	60, 493, 499
__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
__tag_tree_write_namespaces: ..	322, 322, 373
__tag_tree_write_parenttree: ..	246, 246, 357
__tag_tree_write_rolemap:	263, 263, 365
__tag_tree_write_structelements:	147, 147, 377
__tag_tree_write_structtreeroot:	126, 126, 381
\g__tag_unique_cnt_int	96, 1121, 1125, 1128, 1138, 1142, 1146
__tag_whatsits:	36, 43, 48, 49, 52, 293, 294
tag-namespace_(rolemap-key)	780
tag/check/parent-child	183
tag/check/parent-child-end	183
tag/struct/1 internal commands:	
__tag/struct/1	31
tag/tree/namespaces internal commands:	
__tag/tree/namespaces	321
tag/tree/parenttree internal commands:	
__tag/tree/parenttree	154
tag/tree/rolemap internal commands:	
__tag/tree/rolemap	262
tagabspace	8, 122
tagmcabs	8, 122
\tagmcbegin	44, 189, 22
\tagmcend	44, 22
tagmcid	8, 122
\tagmcifinTF	44, 39
\tagmcuse	44, 22
\tagpdfparaOff	46, 469
\tagpdfparaOn	46, 469
\tagpdfsetup ..	8, 44, 69, 116, 117, 188, 6
\tagpdfsuppressmarks	46, 474
\tagstart	7, 183, 214
\tagstop	7, 182, 213
tagstruct	8, 122
\tagstructbegin	44, 114, 153, 188, 189, 45, 225, 233, 277
\tagstructend	44, 45, 227, 235, 278
tagstructobj	8, 122
\tagstructuse	44, 45
\tagtool	14, 20
\tagtool_(deprecated)	13
tagunmarked (deprecated) (key) ...	1, 273
test/lang_(setup-key)	476
TeX and L ^A T _E X 2 _ε commands:	
\OM	166
\@bsphack	110
\@currentHref	121, 95, 99
\@esphack	112
\@gobble	31, 55
\@ifpackageloaded	22
\@maxdepth	179
\@secondoftwo	31, 55
\c@chapter	368, 386
\r@tag@LastPage	57
tex commands:	
\tex_botmarks:D	89
\tex_firstmarks:D	86
\tex_kern:D	182
\tex_marks:D	23, 32, 45, 52, 63, 69
\tex_special:D	52
\tex_splitbotmarks:D	215
\tex_splitfirstmarks:D	195
texsource (key)	1, 977
\tiny	424, 435
title (key)	1, 764
title-o (key)	1, 764
tl commands:	
\c_empty_tl	373, 393
\c_space_tl	55, 56, 58, 60, 100, 104, 116, 167, 191, 197, 198, 216, 230,

