

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

Ulrike Fischer[†]

Released 2026-01-29

Contents

I	7
1 Initialization and test if pdfmanagement is active.	8
2 base package	9
3 Package options	9
4 Packages	9
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	12
9 Commands to fill seq and prop	13
10 General tagging commands	13
11 Keys for tagpdfsetup	15
12 loading of engine/more dependent code	17
II	19
1 Commands	19

^{*}This file describes v0.99y, last revised 2026-01-29.

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

2	Description of log messages	19
2.1	\ShowTagging command	19
2.2	Messages in checks and commands	19
2.3	Messages from the ptagging code	20
2.4	Warning messages from the lua-code	20
2.5	Info messages from the lua-code	20
2.6	Debug mode messages and code	21
2.7	Messages	21
3	Messages	22
3.1	Messages related to mc-chunks	22
3.2	Messages related to structures	23
3.3	Attributes	25
3.4	Roles	25
3.5	Miscellaneous	29
4	Retrieving data	30
5	PDF version check	30
6	User conditionals	30
7	Internal checks	31
7.1	checks for active tagging	31
7.2	Checks related to structures	32
7.3	Checks related to roles	33
7.4	Check related to mc-chunks	34
7.5	Checks related to the state of MC on a page or in a split stream	36
7.6	Benchmarks	39
III		41
1	Setup commands	41
2	Commands related to mc-chunks	41
3	Commands related to structures	41
4	Debugging	42
5	Extension commands	42
5.1	Fake space	42
5.2	Tagging of paragraphs	43
5.3	Header and footer	43
5.4	Link tagging	44
6	Socket support	44
7	User commands and extensions of document commands	45
8	Setup and preamble commands	45

9	Commands for the mc-chunks	45
10	Commands for the structure	46
11	Socket support	47
12	Debugging	48
13	Commands to extend document commands	52
13.1	Document structure	52
13.2	Structure destinations	52
13.3	Fake space	53
13.4	Paratagging	53
13.5	Language support	59
13.6	Header and footer	59
13.7	Links	63
13.8	Attaching css-files for derivation	68
IV		70
1	Trees, pdfmanagement and finalization code	70
1.1	Check structure	70
1.2	Catalog: MarkInfo and StructTreeRoot and OpenAction	70
1.3	Writing the IDtree	72
1.4	Writing structure elements	73
1.5	ParentTree	74
1.6	Rolemap dictionary	76
1.7	Classmap dictionary	77
1.8	Namespaces	78
1.9	Finishing the structure	79
1.10	StructParents entry for Page	80
V		81
1	Public Commands	81
2	Public keys	82
3	Marked content code – shared	83
3.1	Variables and counters	83
3.2	Functions	84
3.3	Keys	88
VI		90

1	Marked content code – generic mode	90
1.1	Variables	90
1.2	Functions	91
1.3	Looking at MC marks in boxes	94
1.4	Keys	100
VII		103
1	Marked content code – luamode code	103
1.1	Commands	104
1.2	Key definitions	109
VIII		112
1	Public Commands	112
2	Public keys	113
2.1	Keys for the structure commands	113
2.2	Setup keys	115
3	Variables	115
3.1	Variables used by the keys	118
3.2	Variables used by tagging code of basic elements	118
4	Commands	119
4.1	Initialization of the StructTreeRoot	119
4.2	Adding the /ID key	120
4.3	Filling in the tag info	120
4.4	Handlings kids	122
4.5	Output of the object	126
4.6	Commands for the parent-child checks	130
5	Keys	133
6	User commands	142
7	Attributes and attribute classes	150
7.1	Variables	151
7.2	Commands and keys	151
IX		154
1	Loading the lua	154
2	User commands to access data	158
3	Logging functions	159

4	Helper functions	161
4.1	Retrieve data functions	161
4.2	Functions to insert the pdf literals	163
5	Function for the real space chars	165
6	Function for the tagging	169
7	Parenttree	174
8	parent-child rules	176
9	Link annotations	179
X		180
1	Code related to roles and structure names	180
1.1	Variables	180
1.2	Namespaces	183
1.3	Adding a new tag	184
1.3.1	pdf 1.7 and earlier	185
1.3.2	The pdf 2.0 version	187
1.4	Helper command to read the data from files	189
1.5	Reading the default data	190
1.6	Parent-child rules	192
1.6.1	Reading in the csv-files	193
1.6.2	Retrieving the parent-child rule	194
1.7	Key-val user interface	200
XI		203
1	Code for interword spaces	203
Index		207

The **tagpdf** main module
Part of the tagpdf package
Ulrike Fischer
Version 0.99y, released 2026-01-29

Part I

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

We need commands to stop tagging in some places. They switches three local booleans and also stop the counting of paragraphs. If they are nested an inner `\tag_resume:n` will not restart tagging. `⟨label⟩` is only used in debugging messages to allow to follow the nesting and to identify which code is disabling the tagging. The label is not expanded so can be a single token, e.g. `\caption`. `\tag_suspend:n` and `\tag_resume:n` are the l3-layer variants of `\SuspendTagging` and `\ResumeTagging` and will be provided by the kernel in the next release.

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

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

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

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

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

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

`activate-struct-dest` (*deprecated*) (*setup key*)

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

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

`activate/softhyphen` (*setup key*) This key allows to activates automatic handling of hyphens inserted by hyphenation. It 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 led 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 activate 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.

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} {2026-01-29} {0.99y}
4 { LaTeX kernel code for PDF tagging }
5
6 \IfPDFManagementActiveF
7 {
8   \PackageError{tagpdf}
9   {
10     PDF~resource~management~is~no~active!\MessageBreak
11     tagpdf~will~no~work.
12   }
13   {
14     Activate~it~with \MessageBreak
15     \string\DocumentMetadata{<options>}\MessageBreak
16     before~\string\documentclass
17   }
18 }
19 </package>
<*debug>
20 \ProvidesExplPackage {tagpdf-debug} {2026-01-29} {0.99y}
21 { debug code for tagpdf }
22 \@ifpackageloaded{tagpdf}{\PackageWarning{tagpdf-debug}{tagpdf~not~loaded,~quitting}}\ending

```


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

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

Debug mode has its special mapping:

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

2 base package

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

```
30 <*base>
31 \ProvidesExplPackage {tagpdf-base} {2026-01-29} {0.99y}
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 </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 <*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 \langle /base \rangle

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  $\langle$ *package $\rangle$ 
57 \AddToHook{begindocument/before}{\cs_undefine:N\r__tagtag@LastPage}
58 \AddToHook{enddocument/afterlastpage}
59 {\property_record:nn{@tag@LastPage}{abspage,tagmcabs,tagstruct}}
```

5 Variables

<pre>\l__tag_tmpa_tl \l__tag_tmpb_tl \l__tag_tmpc_tl \l__tag_tmp_unused_tl \l__tag_Ref_tmpa_tl \l__tag_get_tmpc_tl \l__tag_get_parent_tmpa_tl \l__tag_get_parent_tmpb_tl \l__tag_get_parent_tmpc_tl \l__tag_get_child_tmpa_tl \l__tag_get_child_tmpb_tl \l__tag_get_child_tmpc_tl \l__tag_tmpa_str \l__tag_tmpa_prop \l__tag_tmpa_seq \l__tag_tmpb_seq \l__tag_tmpa_clist \l__tag_tmpa_int \l__tag_tmpa_box \l__tag_tmpb_box</pre>	<p>A few temporary variables</p> <pre>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_Ref_tmpa_tl 65 \tl_new:N \l__tag_get_tmpc_tl 66 \tl_new:N \l__tag_get_parent_tmpa_tl 67 \tl_new:N \l__tag_get_parent_tmpb_tl 68 \tl_new:N \l__tag_get_parent_tmpc_tl 69 \tl_new:N \l__tag_get_child_tmpa_tl 70 \tl_new:N \l__tag_get_child_tmpb_tl 71 \tl_new:N \l__tag_get_child_tmpc_tl 72 \str_new:N \l__tag_tmpa_str 73 \prop_new:N \l__tag_tmpa_prop 74 \seq_new:N \l__tag_tmpa_seq 75 \seq_new:N \l__tag_tmpb_seq 76 \clist_new:N \l__tag_tmpa_clist 77 \int_new:N \l__tag_tmpa_int 78 \box_new:N \l__tag_tmpa_box 79 \box_new:N \l__tag_tmpb_box</pre>
--	--

(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
80 \clist_const:Nn \c__tag_property_mc_clist {tagabspage,tagmcabs,tagmcid}
81 \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.

```
82 \int_new:N \l__tag_loglevel_int
(End of definition for \l__tag_loglevel_int.)
```

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

```

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

```

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

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

```

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

```

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

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

```

94 \bool_new:N \g__tag_tagunmarked_bool

```

(End of definition for `\g__tag_tagunmarked_bool`.)

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

```

95 \bool_new:N \g__tag_softyphen_bool
96 \bool_gset_true:N \g__tag_softyphen_bool

```

(End of definition for `\g__tag_softyphen_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

```

97 \int_new:N \g__tag_unique_cnt_int

```

(End of definition for `\g__tag_unique_cnt_int`.)

6 Variants of l3 commands

```

98 \prg_generate_conditional_variant:Nnn \pdf_object_if_exist:n {e}{T,F,TF}
99 \cs_generate_variant:Nn \pdf_object_ref:n {e}
100 \cs_generate_variant:Nn \pdfannot_dict_put:nnn {nne}
101 \cs_generate_variant:Nn \pdffile_embed_stream:nnn {nee,oe}
102 \cs_generate_variant:Nn \prop_gput:Nnn {Nee,Nen} %** unneeded
103 \cs_generate_variant:Nn \prop_put:Nnn {Nee} %** unneeded
104 \cs_generate_variant:Nn \prop_item:Nn {No,Ne} %** unneeded
105 \cs_generate_variant:Nn \seq_set_split:Nnn{Nno}
106 \cs_generate_variant:Nn \str_set_convert:Nnnn {Nonn, Noon, Nnon }
107 \cs_generate_variant:Nn \clist_map_inline:nn {on}
108 \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`.

```

109 \cs_new_protected:Npn \__tag_property_record:nn #1#2
110 {
111     \@bspack
112     \property_record:nn{#1}{#2}
113     \@espack
114 }
115

```

And a few variants

```

116 \cs_generate_variant:Nn \property_ref:nnn {enn}
117 \cs_generate_variant:Nn \property_ref:nn {en}
118 \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.

```

119 \cs_new:Npn \__tag_property_ref_lastpage:nn #1 #2
120 {
121     \property_ref:nnn {@tag@LastPage}{#1}{#2}
122 }

```

(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 `tagabspace`, the absolute id `tagmcabc`, and the id on the page `tagmcid`.

```

123 \property_new:nnnn

```

```

124 { tagstruct } { now }
125 {1} { \int_use:N \c@g__tag_struct_abs_int }
126 \property_new:nnnn { tagstructobj } { now } {}
127 {
128   \pdf_object_ref_indexed:nn { __tag/struct } { \c@g__tag_struct_abs_int }
129 }
130 \property_new:nnnn
131 { tagabspage } { shipout }
132 {0} { \int_use:N \g_shipout_readonly_int }
133 \property_new:nnnn { tagmcabs } { now }
134 {0} { \int_use:N \c@g__tag_MCID_abs_int }
135
136 \flag_new:n { __tag/mcid }
137 \property_new:nnnn {tagmcid } { shipout }
138 {0} { \flag_height:n { __tag/mcid } }
139

```

(End of definition for tagstruct and others. These functions are documented on page 8.)

9 Commands to fill seq and prop

With most engines these are simply copies of the expl3 commands, but luatex will overwrite them, to store the data also in lua tables.

```

    \__tag_prop_new:N
\__tag_prop_new_linked:N 140 \cs_set_eq:NN \__tag_prop_new:N \prop_new:N
    \__tag_seq_new:N      141 \cs_set_eq:NN \__tag_prop_new_linked:N \prop_new_linked:N
    \__tag_prop_gput:Nnn  142 \cs_set_eq:NN \__tag_seq_new:N \seq_new:N
\__tag_seq_gput_right:Nn  143 \cs_set_eq:NN \__tag_prop_gput:Nnn \prop_gput:Nnn
    \__tag_seq_item:cn    144 \cs_set_eq:NN \__tag_seq_gput_right:Nn \seq_gput_right:Nn
    \__tag_prop_item:cn   145 \cs_set_eq:NN \__tag_seq_gput_left:Nn \seq_gput_left:Nn
    \__tag_seq_show:N     146 \cs_set_eq:NN \__tag_seq_item:cn \seq_item:cn
    \__tag_prop_show:N    147 \cs_set_eq:NN \__tag_prop_item:cn \prop_item:cn
    \__tag_prop_show:N    148 \cs_set_eq:NN \__tag_seq_show:N \seq_show:N
    \__tag_prop_show:N    149 \cs_set_eq:NN \__tag_prop_show:N \prop_show:N
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, c }
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 stop the counting of paragraphs. The commands keep track of the nesting with a local counter. Tagging only is only restarted at the outer level, if the current level is 1. The

\tag_resume:n commands with argument allow to give a label. This is only used in debugging messages

\tag_stop:

\tag_start:

\tag_stop:n

\tag_start:n

to allow to follow the nesting. The label is not expand so can e.g. be a single command token.

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

```

159 <*package | debug>
160 <package>\int_new:N \l__tag_tag_stop_int
\l__tag_tag_stop_int
161 \cs_set_protected:Npn \tag_stop:
162 {
163 <debug> \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 <debug> \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 Keys for tagpdfsetup

TODO: the log-levels must be sorted

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

activate/all (*setup key*)

activate/struct-dest (*setup key*)

```

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

```

old, deprecated names

```

230     activate-mc      .bool_gset:N = \g__tag_active_mc_bool,
231     activate-tree    .bool_gset:N = \g__tag_active_tree_bool,
232     activate-struct .bool_gset:N = \g__tag_active_struct_bool,
233     activate-all     .meta:n =
234       {activate/mc={#1},activate/tree={#1},activate/struct={#1}},
235     activate-all     .default:n = true,
236     no-struct-dest .bool_gset:N = \g__tag_active_struct_dest_bool,

```

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

```

237     debug/show      .choice:,

```

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

debug/uncompress (*setup key*)

log (deprecated) (*setup key*)

uncompress (deprecated) (*setup key*)

```

238     debug/log      .choice:,
239     debug/log / none .code:n = {\int_set:Nn \l__tag_loglevel_int { 0 }},
240     debug/log / v   .code:n =

```

```

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

```

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

```

249 log      .meta:n = {debug/log={#1}},
250 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.

```

251 activate/tagunmarked .bool_gset:N = \g__tag_tagunmarked_bool,
252 activate/tagunmarked .initial:n = true,

```

deprecated name

```

253 tagunmarked .bool_gset:N = \g__tag_tagunmarked_bool,

```

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

```

254 activate/softhyphen .choice:,
255 activate/softhyphen/false .code:n = {\bool_gset_false:N \g__tag_softhyphen_bool},
256 activate/softhyphen/off .code:n = {\bool_gset_false:N \g__tag_softhyphen_bool},
257 activate/softhyphen/char .code:n =
258 {
259   \bool_gset_true:N \g__tag_softhyphen_bool
260   \sys_if_engine luatex:T
261   {
262     \lua_now:e
263     {
264       tex.setattribute
265       (
266         luatexbase.attributes.l__tag_softhyphen_attr,
267         1
268       )
269     }
270   }
271 },
272 activate/softhyphen/artifact .code:n =
273 {
274   \bool_gset_true:N \g__tag_softhyphen_bool
275   \sys_if_engine luatex:T
276   {
277     \lua_now:e
278     {
279       tex.setattribute
280       (
281         luatexbase.attributes.l__tag_softhyphen_attr,
282         -2147483647
283       )
284     }
285   }

```



```

286 },
287 activate/softhyphen/true .code:n =
288 {
289   \bool_gset_true:N \g__tag_softhyphen_bool
290   \sys_if_engine luatex:T
291   {
292     \lua_now:e
293     {
294       tex.setattribute
295       (
296         luatexbase.attributes.l__tag_softhyphen_attr,
297         -2147483647
298       )
299     }
300   }
301 },
302 activate/softhyphen/on .code:n =
303 {
304   \bool_gset_true:N \g__tag_softhyphen_bool
305   \sys_if_engine luatex:T
306   {
307     \lua_now:e
308     {
309       tex.setattribute
310       (
311         luatexbase.attributes.l__tag_softhyphen_attr,
312         -2147483647
313       )
314     }
315   }
316 },
317 activate/softhyphen .default:n = artifact,

```

`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.

```

318 page/tabsorder .choice:,
319 page/tabsorder / row .code:n =
320   \pdfmanagement_add:nnn { Page } {Tabs}{/R},
321 page/tabsorder / column .code:n =
322   \pdfmanagement_add:nnn { Page } {Tabs}{/C},
323 page/tabsorder / structure .code:n =
324   \pdfmanagement_add:nnn { Page } {Tabs}{/S},
325 page/tabsorder / none .code:n =
326   \pdfmanagement_remove:nn {Page} {Tabs},
327 page/tabsorder .initial:n = structure,

```

deprecated name

```

328 tabsorder .meta:n = {page/tabsorder={#1}},
329 }

```

12 loading of engine/more dependent code

```

330 \sys_if_engine luatex:T

```

```

331 {
332   \file_input:n {tagpdf-luatex.def}
333 }
334 \endpackage
335 \begin{mcloding}
336 \bool_if:NTF \g__tag_mode_lua_bool
337 {
338   \RequirePackage {tagpdf-mc-code-lua}
339 }
340 {
341   \RequirePackage {tagpdf-mc-code-generic} %
342 }
343 \end{mcloding}
344 \begin{debug}
345 \bool_if:NTF \g__tag_mode_lua_bool
346 {
347   \RequirePackage {tagpdf-debug-lua}
348 }
349 {
350   \RequirePackage {tagpdf-debug-generic} %
351 }
352 \end{debug}

```

The tagpdf-checks module

Messages and check code

Part of the tagpdf package

Ulrike Fischer

Version 0.99y, released 2026-01-29

Part II

1 Commands

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

`\tag_get:n *` `\tag_get:n {<keyword>}`

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

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

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

2 Description of log messages

2.1 \ShowTagging command

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

2.2 Messages in checks and commands

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

2.3 Messages from the ptagging code

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

2.4 Warning messages from the lua-code

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

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

2.5 Info messages from the lua-code

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

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

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

2.6 Debug mode messages and code

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

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

2.7 Messages

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

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

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

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

tree-struct-still-open

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

tree-statistic

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

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

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

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

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

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

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

```

1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-checks-code} {2026-01-29} {0.99y}
4 {part of tagpdf - code related to checks, conditionals, debugging and messages}
5 </header>
```

3 Messages

3.1 Messages related to mc-chunks

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

```

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

(End of definition for mc-nested. This function is documented on page 21.)

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 21.)

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

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

(End of definition for mc-label-unknown. This function is documented on page 21.)

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

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

(End of definition for mc-used-twice. This function is documented on page 21.)

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 21.)

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 21.)

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 21.)

3.2 Messages related to structures

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

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

(End of definition for struct-unknown. This function is documented on page 21.)

struct-no-objnum Should not happen ...

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

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

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

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

(End of definition for struct-orphan. This function is documented on page 21.)

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

```
32 \msg_new:nnn { tag }
33   {struct-faulty-nesting}
34   { there~is~no~open~structure~on~the~stack }
(End of definition for struct-faulty-nesting. This function is documented on page 21.)
```

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

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

(End of definition for struct-missing-tag. This function is documented on page 21.)

struct-used-twice

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

(End of definition for struct-used-twice. This function is documented on page 21.)

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

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

(End of definition for struct-label-unknown. This function is documented on page 21.)

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

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

(End of definition for struct-show-closing. This function is documented on page 21.)

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

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

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

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

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

(End of definition for tree-struct-still-open. This function is documented on page 21.)

tree-statistic Message issued at the end showing the estimated number of structures and MC-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 \end{package}

```

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

The following messages are only needed in debug mode.

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

show-kids

```

64 \begin{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 \end{debug}

```

(End of definition for `show-struct` and `show-kids`. These functions are documented on page 22.)

3.3 Attributes

Not much yet, as attributes aren't used so much.

attr-unknown

```

84 \begin{package}
85 \msg_new:nnn { tag } { attr-unknown } { attribute~#1~is~unknown}

```

(End of definition for `attr-unknown`. This function is documented on page 22.)

3.4 Roles

role-missing

role-unknown

role-unknown-tag

role-unknown-NS

Warning message if either the tag or the role is missing

```

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

```

(End of definition for `role-missing` and others. These functions are documented on page 22.)

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 22.)

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:nNtT {#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:nNtT {#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:nNtT { #1 } = {\c\_tag\_role\_rule\_checkparent\_tl}
174   {
175     \msg_warning:nneee
176       { tag }
177       {role-parent-child-unresolved}
178       { #2 }
179       { #3 }
180       { #4 }
181   }
182 }

```

(End of definition for _tag_check_unresolved_parent_child:nnnn.)

tag/check/parent-child
tag/check/parent-child-end

Sockets used around the parent-child checks so that we can disable them.

```

183 \socket_new:nn{tag/check/parent-child}{1}
184 \socket_new:nn{tag/check/parent-child-end}{0}
185 \socket_new_plug:nnn {tag/check/parent-child-end}{check}
186 {
187   \sys_if_engine luatex:T
188   {
189     \lua_now:e
190     {
191       ltx\_tag.func.check\_parent\_child\_rules ( 2 )
192     }
193   }
194 }

```

And a key to disable the check

```

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

```

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

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

```
213 \msg_new:nnn { tag } {role-remapping}
214   { remapping~tag~to~#1 }
(End of definition for role-remapping. This function is documented on page 22.)
```

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 22.)
```

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 22.)
```

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

```
228 \msg_new:nnn { tag } {sys-no-interwordspace}
229   {engine/output~mode~#1~doesn't~support~the~interword~spaces}
(End of definition for sys-no-interwordspace. This function is documented on page 22.)
```

__tag_check_typeout_v:n A simple logging function. By default it 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~quite~probably~a~coding~error~and~the~structure~will~be~wrong!}
234 </package>
(End of definition for para-hook-count-wrong. This function is documented on page 22.)
```

4 Retrieving data

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

```
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 19.)
```

5 PDF version check

```
236 <*package>  
237 \tl_const:Nc \c__tag_check_pdfversion_tl {\pdf_version:}  
238 \AddToHook{begindocument/before}  
239 {  
240   \tl_if_eq:eeF{\c__tag_check_pdfversion_tl}{\pdf_version:}  
241   {  
242     \msg_error:nnee {tag}{wrong-pdfversion}{\c__tag_check_pdfversion_tl}{\pdf_version:}  
243     \pdf_version_gset:e {\c__tag_check_pdfversion_tl}  
244     \pdf_object_unnamed_write:nn{dict}{}  
245   }  
246 }  
247 </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

```
248 <*base>  
249 \cs_if_exist:Nf \tag_if_active:T  
250 {  
251   \prg_new_conditional:Npnn \tag_if_active: { p , T , TF, F }  
252   { \prg_return_false: }  
253 }  
254 </base>  
255 <*package>  
256 \prg_set_conditional:Npnn \tag_if_active: { p , T , TF, F }  
257 {  
258   \bool_lazy_all:nTF  
259   {  
260     {\g__tag_active_struct_bool}  
261     {\g__tag_active_mc_bool}  
262     {\g__tag_active_tree_bool}  
263     {\l__tag_active_struct_bool}  
264     {\l__tag_active_mc_bool}  
265   }  
266   {  
267     \prg_return_true:  
268   }  
269   {  
270     \prg_return_false:  
271   }  
272 }  
273 </package>
```

(End of definition for `\tag_if_active:TF`. This function is documented on page 19.)

`\tag_if_box_tagged_p:N` This tests if a box contains tagging commands. It relies on that the code that saved the box correctly set `\l_tag_box_<box number>_tl` to a positive value. The LaTeX commands will do that automatically at some time but it is in the responsibility of the user to ensure that when using low-level code. If the internal command doesn't exist the box is assumed to be untagged.

```

274 <*base>
275 \prg_new_conditional:Npnn \tag_if_box_tagged:N #1 {p,T,F,TF}
276 {
277   \tl_if_exist:cTF {l_tag_box_\int_use:N #1_tl}
278   {
279     \int_compare:nNnTF {0\tl_use:c{l_tag_box_\int_use:N #1_tl}}>{0}
280     { \prg_return_true: }
281     { \prg_return_false: }
282   }
283   {
284     \prg_return_false:
285     % warning??
286   }
287 }
288 </base>

```

(End of definition for `\tag_if_box_tagged:NTF`. This function is documented on page 19.)

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`

```

289 <*package>
290 \prg_new_conditional:Npnn \__tag_check_if_active_mc: {T,F,TF}
291 {
292   \bool_lazy_and:nnTF { \g__tag_active_mc_bool } { \l__tag_active_mc_bool }
293   {
294     \prg_return_true:
295   }
296   {
297     \prg_return_false:
298   }
299 }
300 \prg_new_conditional:Npnn \__tag_check_if_active_struct: {T,F,TF}
301 {
302   \bool_lazy_and:nnTF { \g__tag_active_struct_bool } { \l__tag_active_struct_bool }
303   {
304     \prg_return_true:
305   }
306   {
307     \prg_return_false:
308   }
309 }

```

(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.

```

310 \cs_new_protected:Npn __tag_check_structure_has_tag:n #1 %#1 struct num
311 {
312   \prop_get:cnNF
313     { g__tag_struct_#1_prop }
314     {S}
315     \l__tag_tmp_unused_tl
316     {
317       \msg_error:nn { tag } {struct-missing-tag}
318     }
319 }
```

(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.

```

320 \cs_new_protected:Npn __tag_check_structure_tag:N #1
321 {
322   \prop_get:NoNF \g__tag_role_tags_NS_prop {#1}\l__tag_tmp_unused_tl
323   {
324     \msg_warning:nne { tag } {role-unknown-tag} {#1}
325   }
326 }
```

(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.

```

327 \cs_new_protected:Npn __tag_check_info_closing_struct:n #1 %#1 struct num
328 {
329   \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
330   {
331     \msg_info:nnn { tag } {struct-show-closing} {#1}
332   }
333 }
```

```

335 \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.

```

336 \cs_new_protected:Npn __tag_check_no_open_struct:
337 {
338   \msg_error:nn { tag } {struct-faulty-nesting}
339 }
```

(End of definition for `__tag_check_no_open_struct:`.)

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

```

340 \cs_new_protected:Npn \__tag_check_struct_used:n #1 %#1 label
341 {
342   \prop_get:cnNT
343     {g__tag_struct_\property_ref:enn{tagpdfstruct-#1}{tagstruct}{unknown}_prop}
344     {parentnum}
345     \l__tag_tmpa_tl
346     {
347       \msg_warning:nnn { tag } {struct-used-twice} {#1}
348     }
349 }

```

(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.

```

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

```

Similar with a namespace

```

369 \cs_new_protected:Npn \__tag_check_add_tag_role:nnn #1 #2 #3 %#1 tag/NS, #2 role #3 namespace
370 {
371   \tl_if_empty:nTF {#2}
372   {
373     \msg_error:nnn { tag } {role-missing} {#1}
374   }
375   {
376     \prop_get:cnNTF { g__tag_role_NS_#3_prop } {#2} \l__tag_tmpa_tl
377     {
378       \int_compare:nNtT {\l__tag_loglevel_int} > { 0 }
379       {
380         \msg_info:nnnn { tag } {role-tag} {#1} {#2/#3}
381       }
382     }
383     {
384       \msg_error:nnn { tag } {role-unknown} {#2/#3}
385     }
386   }

```

```

386     }
387   }

```

(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).

```

388 \cs_new_protected:Npn \__tag_check_mc_if_nested:
389 {
390   \__tag_mc_if_in:T
391   {
392     \msg_warning:nne { tag } {mc-nested} { \__tag_get_mc_abs_cnt: }
393   }
394 }
395
396 \cs_new_protected:Npn \__tag_check_mc_if_open:
397 {
398   \__tag_mc_if_in:F
399   {
400     \msg_warning:nne { tag } {mc-not-open} { \__tag_get_mc_abs_cnt: }
401   }
402 }

```

(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.

```

403 \cs_new_protected:Npn \__tag_check_mc_pushed_popped:nn #1 #2
404 {
405   \int_compare:nNnT
406     { \l__tag_loglevel_int } = { 2 }
407     { \msg_info:nne {tag}{mc-#1}{#2} }
408   \int_compare:nNnT
409     { \l__tag_loglevel_int } > { 2 }
410     {
411       \msg_info:nne {tag}{mc-#1}{#2}
412       \seq_log:N \g__tag_mc_stack_seq
413     }
414 }

```

(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.

```

415 \cs_new_protected:Npn \__tag_check_mc_tag:N #1 % #1 is var with a tag name in it
416 {
417   \tl_if_empty:NTF #1
418   {
419     \tl_set:No #1 { \g__tag_struct_tag_tl }
420     \msg_info:nnee { tag } {mc-tag-missing} { \g__tag_struct_tag_tl } { \__tag_get_mc_abs_
421   }

```

```

422     {
423       \prop_get:NoNF \g__tag_role_tags_NS_prop {#1}\l__tag_tmp_unused_tl
424       {
425         \msg_warning:nne { tag } {role-unknown-tag} {#1}
426       }
427     }
428   }

```

(End of definition for __tag_check_mc_tag:N.)

\g__tag_check_mc_used_intarray
__tag_check_init_mc_used:

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

```

429 \cs_new_protected:Npn \__tag_check_init_mc_used:
430 {
431   \intarray_new:Nn \g__tag_check_mc_used_intarray { 65536 }
432   \cs_gset_eq:NN \__tag_check_init_mc_used: \prg_do_nothing:
433 }

```

(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.

```

434 \cs_new_protected:Npn \__tag_check_mc_used:n #1 %#1 mcid absnt
435 {
436   \int_compare:nNnT {\l__tag_loglevel_int} > { 2 }
437   {
438     \__tag_check_init_mc_used:
439     \intarray_gset:Nnn \g__tag_check_mc_used_intarray
440       {#1}
441       { \intarray_item:Nn \g__tag_check_mc_used_intarray {#1} + 1 }
442     \int_compare:nNnT
443       {
444         \intarray_item:Nn \g__tag_check_mc_used_intarray {#1}
445       }
446       >
447       { 1 }
448       {
449         \msg_warning:nnn { tag } {mc-used-twice} {#1}
450       }
451   }
452 }

```

(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.

```

453 \cs_new_protected:Npn \__tag_check_show_MCID_by_page:
454 {
455   \tl_set:Ne \l__tag_tmpa_tl
456   {
457     \__tag_property_ref_lastpage:nn
458       {abspage}
459       {-1}

```

```

460     }
461     \int_step_inline:nnnn {1}{1}
462     {
463         \l__tag_tmpa_tl
464     }
465     {
466         \seq_clear:N \l__tag_tmpa_seq
467         \int_step_inline:nnnn
468             {1}
469             {1}
470             {
471                 \__tag_property_ref_lastpage:nn
472                 {tagmcabs}
473                 {-1}
474             }
475             {
476                 \int_compare:nT
477                 {
478                     \property_ref:enn
479                     {mcid-####1}
480                     {tagabspage}
481                     {-1}
482                     =
483                     ##1
484                 }
485                 {
486                     \seq_gput_right:Ne \l__tag_tmpa_seq
487                     {
488                         Page##1-####1-
489                         \property_ref:enn
490                         {mcid-####1}
491                         {tagmcid}
492                         {-1}
493                     }
494                 }
495             }
496         \seq_show:N \l__tag_tmpa_seq
497     }
498 }

```

(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`.

```

499 \prg_new_conditional:Npnn \__tag_check_if_mc_in_galley: { T,F,TF }
500 {
501   \tl_if_eq:NNTF \l__tag_mc_firstmarks_seq \l__tag_mc_botmarks_seq
502   { \prg_return_false: }
503   { \prg_return_true: }
504 }
(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
`__tag_check_if_mc_tmb_missing:TF` the case if the firstmarks start with `e-` or `b+`. Like above we assume that the marks content is already in the seq's.

```

505 \prg_new_conditional:Npnn \__tag_check_if_mc_tmb_missing: { T,F,TF }
506 {
507   \bool_if:nTF
508   {
509     \str_if_eq_p:ee {\seq_item:Nn \l__tag_mc_firstmarks_seq {1}}{e-}
510     ||
511     \str_if_eq_p:ee {\seq_item:Nn \l__tag_mc_firstmarks_seq {1}}{b+}
512   }
513   { \prg_return_true: }
514   { \prg_return_false: }
515 }
(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.

```

516 \prg_new_conditional:Npnn \__tag_check_if_mc_tme_missing: { T,F,TF }
517 {
518   \str_if_eq:eeTF {\seq_item:Nn \l__tag_mc_botmarks_seq {1}}{b+}
519   { \prg_return_true: }
520   { \prg_return_false: }
521 }
(End of definition for \__tag_check_if_mc_tme_missing:TF.)

```

```

522 \</package>

```

```

523 \<debug>

```

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

```

524 \msg_new:nnn { tag / debug } {mc-begin} { MC~begin~#1~with~options:~\tl_to_str:n{#2}~[\msg_line_context:] }
525 \msg_new:nnn { tag / debug } {mc-end} { MC~end~#1~[\msg_line_context:] }
526
527 \cs_new_protected:Npn \__tag_debug_mc_begin_insert:n #1
528 {
529   \int_compare:nNtT { \l__tag_loglevel_int } > {0}
530   {
531     \msg_note:nnnn { tag / debug } {mc-begin} {inserted} { #1 }
532   }
533 }
534 \cs_new_protected:Npn \__tag_debug_mc_begin_ignore:n #1

```

```

535 {
536     \int_compare:nNnT { \l__tag_loglevel_int } > {0}
537     {
538         \msg_note:nnnn { tag / debug } {mc-begin } {ignored} { #1 }
539     }
540 }
541 \cs_new_protected:Npn \__tag_debug_mc_end_insert:
542 {
543     \int_compare:nNnT { \l__tag_loglevel_int } > {0}
544     {
545         \msg_note:nnn { tag / debug } {mc-end} {inserted}
546     }
547 }
548 \cs_new_protected:Npn \__tag_debug_mc_end_ignore:
549 {
550     \int_compare:nNnT { \l__tag_loglevel_int } > {0}
551     {
552         \msg_note:nnn { tag / debug } {mc-end } {ignored}
553     }
554 }

```

And now something for the structures

```

555 \msg_new:nnn { tag / debug } {struct-begin}
556 {
557     Struct~\tag_get:n{struct_num}~begin~#1~with~options:~\tl_to_str:n{#2}~\[\msg_line_context
558 }
559 \msg_new:nnn { tag / debug } {struct-end}
560 {
561     Struct~end~#1~[\msg_line_context:]
562 }
563 \msg_new:nnn { tag / debug } {struct-end-wrong}
564 {
565     Struct~end~'#1'~doesn't~fit~start~'#2'~[\msg_line_context:]
566 }
567
568 \cs_new_protected:Npn \__tag_debug_struct_begin_insert:n #1
569 {
570     \int_compare:nNnT { \l__tag_loglevel_int } > {0}
571     {
572         \msg_note:nnnn { tag / debug } {struct-begin} {inserted} { #1 }
573         \seq_log:N \g__tag_struct_tag_stack_seq
574     }
575 }
576 \cs_new_protected:Npn \__tag_debug_struct_begin_ignore:n #1
577 {
578     \int_compare:nNnT { \l__tag_loglevel_int } > {0}
579     {
580         \msg_note:nnnn { tag / debug } {struct-begin } {ignored} { #1 }
581     }
582 }
583 \cs_new_protected:Npn \__tag_debug_struct_end_insert:
584 {
585     \int_compare:nNnT { \l__tag_loglevel_int } > {0}
586     {
587         \msg_note:nnn { tag / debug } {struct-end} {inserted}

```

```

588     \seq_log:N \g__tag_struct_tag_stack_seq
589   }
590 }
591 \cs_new_protected:Npn \__tag_debug_struct_end_ignore:
592 {
593   \int_compare:nNtT { \l__tag_loglevel_int } > {0}
594   {
595     \msg_note:nnn { tag / debug } {struct-end } {ignored}
596   }
597 }
598 \cs_new_protected:Npn \__tag_debug_struct_end_check:n #1
599 {
600   \int_compare:nNtT { \l__tag_loglevel_int } > {0}
601   {
602     \seq_get:NNT \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
603     {
604       \str_if_eq:eeF
605       {#1}
606       {\exp_last_unbraced:No \use_i:nn { \l__tag_tmpa_tl }}
607       {
608         \msg_warning:nnee { tag/debug }{ struct-end-wrong }
609         {#1}
610         {\exp_last_unbraced:No \use_i:nn { \l__tag_tmpa_tl }}
611       }
612     }
613   }
614 }

```

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.

```

615 \msg_new:nnn { tag / debug } {tag-suspend}
616 {
617   \int_if_zero:nTF
618   {#1}
619   {Tagging~suspended}
620   {Tagging~(not)~suspended~(already~inactive)}\
621   level:~#1~==>~\int_eval:n{#1+1}\tl_if_empty:nF{#2}{,~label:~#2}~[\msg_line_context:]
622 }
623 \msg_new:nnn { tag / debug } {tag-resume}
624 {
625   \int_if_zero:nTF
626   {#1}
627   {Tagging~resumed}
628   {Tagging~(not)~resumed}\
629   level:~\int_eval:n{#1+1}~==>~#1\tl_if_empty:nF{#2}{,~label:~#2}~[\msg_line_context:]
630 }
631 </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?

```

632 <*package>

```

```

633 \cs_new_protected:Npn \__tag_check_benchmark_tic:{}
634 \cs_new_protected:Npn \__tag_check_benchmark_toc:{}
635 \cs_new_protected:Npn \tag_check_benchmark_on:
636 {
637   \cs_if_exist:NT \benchmark_tic:
638   {
639     \cs_set_eq:NN \__tag_check_benchmark_tic: \benchmark_tic:
640     \cs_set_eq:NN \__tag_check_benchmark_toc: \benchmark_toc:
641   }
642 }
643 \end{package}

```

The tagpdf-user module

Code related to L^AT_EX2e user commands and document commands

Part of the tagpdf package

Ulrike Fischer

Version 0.99y, released 2026-01-29

Part III

1 Setup commands

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

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

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

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

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

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

2 Commands related to mc-chunks

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

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

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

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

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

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

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

3 Commands related to structures

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

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

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

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

4 Debugging

`\ShowTagging` `\ShowTagging{⟨key-val⟩}`

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

`mc-data (show-key)` `mc-data = ⟨number⟩`

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

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

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

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

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

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

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

`debug/structures (show-key)` `debug/structures = ⟨structure number⟩`

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

5 Extension commands

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

The commands and keys should be view as experimental!

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

5.1 Fake space

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

5.2 Tagging of paragraphs

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

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

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

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

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

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

5.3 Header and footer

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

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

5.4 Link tagging

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

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

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

6 Socket support

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

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

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

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} {2026-01-29} {0.99y}
4   {tagpdf - user commands}
5 </header>

```

8 Setup and preamble commands

`\tagpdfsetup`

```

6 <base>\NewDocumentCommand \tagpdfsetup { m }{}
7 <*package>
8 \RenewDocumentCommand \tagpdfsetup { m }
9   {
10     \keys_set:nn { __tag / setup } { #1 }
11   }
12 </package>

```

(End of definition for `\tagpdfsetup`. This function is documented on page 41.)

`\tag_tool:n`
`\tagtool`

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

```

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

```

(End of definition for `\tag_tool:n` and `\tagtool`. These functions are documented on page 41.)

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 41.)

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

```

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

```

(End of definition for `\tagmcifinTF`. This function is documented on page 41.)

10 Commands for the structure

`\tagstructbegin`
`\tagstructend`
`\tagstructuse`

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

```

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

```

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

11 Socket support

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

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

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

72 <*package>
73 \cs_set_protected:Npn \tag_socket_use:n #1
74 {
75   \bool_if:NT \l__tag_active_socket_bool
76   { \socket_use:n {tagsupport/#1} }
77 }
78 \cs_set_protected:Npn \tag_socket_use:nn #1#2
79 {
80   \bool_if:NT \l__tag_active_socket_bool
81   { \socket_use:nn {tagsupport/#1} {#2} }
82 }
83 \cs_set_protected:Npn \tag_socket_use:nnn #1#2#3
84 {
85   \bool_if:NTF \l__tag_active_socket_bool
86   { \socket_use:nnn {tagsupport/#1} {#2} {#3} }
87   { #3 }
88 }
89 \cs_set:Npn \tag_socket_use_expandable:n #1
90 {
91   \bool_if:NT \l__tag_active_socket_bool
92   { \socket_use_expandable:n {tagsupport/#1} }
93 }
94 \cs_set_protected:Npn \UseTaggingSocket #1
95 {
96   \bool_if:NTF \l__tag_active_socket_bool
97   { \socket_use:nw {tagsupport/#1} }
98   {
99     \int_case:nnF
100     { \int_use:c { c__socket_tagsupport/#1_args_int } }
101     {
102       0 \prg_do_nothing:
103       1 \use_none:n
104       2 \use_ii:nn
```

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

```

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

```

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

```

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

```

(End of definition for `\tag_socket_use:n` and others. These functions are documented on page 44.)

12 Debugging

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

```

125 <*package>
126 \NewDocumentCommand\ShowTagging { m }
127 {
128     \keys_set:nn { __tag / show }{ #1}
129 }
130 }

```

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

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

```

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

```



```

140     ,mc-data .default:n = 1
141   }
142

```

(End of definition for mc-data (show-key). This function is documented on page 42.)

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

```

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

```

```

191             (luatexbase.attributes.g__tag_mc_type_attr)
192             ,0
193         )
194         texio.write_nl("")
195     }
196 }
197 }
198 {
199     \msg_note:nn{ tag }{ mc-current }
200 }
201 }
202 }

```

(End of definition for mc-current (show-key). This function is documented on page 42.)

mc-marks (show-key) It maps the mc-marks into the sequences and then shows them. This allows to inspect the first and last mc-Mark on a page. It should only be used in the shipout (header/footer).

```

203 \keys_define:nn { __tag / show }
204 {
205     mc-marks .choice: ,
206     mc-marks / show .code:n =
207     {
208         \__tag_mc_get_marks:
209         \__tag_check_if_mc_in_galley:TF
210         {
211             \iow_term:n {Marks~from~this~page:~}
212         }
213         {
214             \iow_term:n {Marks~from~a~previous~page:~}
215         }
216         \seq_show:N \l__tag_mc_firstmarks_seq
217         \seq_show:N \l__tag_mc_botmarks_seq
218         \__tag_check_if_mc_tmb_missing:T
219         {
220             \iow_term:n {BDC~missing~on~this~page!}
221         }
222         \__tag_check_if_mc_tme_missing:T
223         {
224             \iow_term:n {EMC~missing~on~this~page!}
225         }
226     },
227     mc-marks / use .code:n =
228     {
229         \__tag_mc_get_marks:
230         \__tag_check_if_mc_in_galley:TF
231         { Marks~from~this~page:~}
232         { Marks~from~a~previous~page:~}
233         \seq_use:Nn \l__tag_mc_firstmarks_seq {,~}\quad
234         \seq_use:Nn \l__tag_mc_botmarks_seq {,~}\quad
235         \__tag_check_if_mc_tmb_missing:T
236         {
237             BDC~missing~
238         }
239         \__tag_check_if_mc_tme_missing:T
240         {

```

```

241         EMC~missing
242     }
243 },
244 mc-marks .default:n = show
245 }

```

(End of definition for mc-marks (show-key). This function is documented on page 42.)

struct-stack (show-key)

```

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

```

(End of definition for struct-stack (show-key). This function is documented on page 42.)

debug/structures (show-key) The following key is available only if the tagpdf-debug package is loaded and shows all structures starting with the one with the number given by the key.

```

254 <*debug>
255 \keys_define:nn { __tag / show }
256 {
257     ,debug/structures .code:n =
258     {
259         \int_step_inline:nnn{#1}{\c@g__tag_struct_abs_int}
260         {
261             \msg_term:nneeee
262             { tag/debug } { show-struct }
263             { ##1 }
264             {
265                 \prop_map_function:cN
266                 {g__tag_struct_debug_##1_prop}
267                 \msg_show_item_unbraced:nn
268             }
269             { } { }
270             \msg_term:nneeee
271             { tag/debug } { show-kids }
272             { ##1 }
273             {
274                 \seq_map_function:cN
275                 {g__tag_struct_debug_kids_##1_seq}
276                 \msg_show_item_unbraced:n
277             }
278             { } { }
279         }
280     }
281     ,debug/structures .default:n = 1
282 }
283 </debug>

```

(End of definition for debug/structures (show-key). This function is documented on page 42.)

13 Commands to extend document commands

The following commands and code parts are not core commands of tagpdf. They either provide work-arounds for missing functionality elsewhere, or do a first step to apply tagpdf commands to document commands. This part should be regularly revisited to check if the code should go to a better place or can be improved.

284 `<*package>`

13.1 Document structure

```

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

```

(End of definition for `\g__tag_root_default_tl`, `activate (setup-key)`, and `activate/socket (setup-key)`. These functions are documented on page 41.)

13.2 Structure destinations

Since TeXlive 2022 pdfTeX and LuaTeX offer support for structure destinations and the pdfmanagement has backend support for. We activate them if structures are actually created. Structure destinations are actually PDF 2.0 only but they don't harm in older PDF and can improve html export.

```

303 \AddToHook{begindocument/before}
304 {
305   \bool_lazy_and:nnT
306   { \g__tag_active_struct_dest_bool }
307   { \g__tag_active_struct_bool }
308   {
309     \tl_set:Nn \l_pdf_current_structure_destination_tl
310     { {__tag/struct}{\g__tag_struct_stack_current_tl } }
311     \pdf_activate_indexed_structure_destination:
312   }
313 }

```

13.3 Fake space

`\pdffakespace` We need a luatex variant for `\pdffakespace`. This should probably go into the kernel at some time. We also provide a no-op version for dvi mode

```

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

```

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

13.4 Paratagging

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

```

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

```

At first some variables.

```

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

```

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

```

330 \tl_new:N \g__tag_para_main_struct_tl
331 \tl_gset:Nn \g__tag_para_main_struct_tl {1}
332 \tl_new:N \l__tag_para_tag_default_tl
333 \tl_new:N \l__tag_para_tag_tl
334 \tl_set:Nn \l__tag_para_tag_tl { \l__tag_para_tag_default_tl }
335 \tl_new:N \l__tag_para_main_tag_tl
336 \IfFormatAtLeastTF{2025-11-01}
337 {
338   \tl_set:Nn \l__tag_para_tag_default_tl { \UseStructureName {para/textblock} }
339   \tl_set:Nn \l__tag_para_main_tag_tl { \UseStructureName {para/semantic} }
340 }
341 {
342   \tl_set:Nn \l__tag_para_tag_default_tl { text }
343   \tl_set:Nn \l__tag_para_main_tag_tl {text-unit}
344 }
345

```

this is perhaps already defined by the block code

```

346 \tl_if_exist:NF \l__tag_para_attr_class_tl
347 {\tl_new:N \l__tag_para_attr_class_tl }
348 \tl_new:N \l__tag_para_main_attr_class_tl

```

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

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

```
\_tag_gincr_para_begin_int: 349 \cs_new_protected:Npn \__tag_gincr_para_main_begin_int:
\_tag_gincr_para_end_int: 350 {
351   \int_gincr:N \g__tag_para_main_begin_int
352 }
353 \cs_new_protected:Npn \__tag_gincr_para_begin_int:
354 {
355   \int_gincr:N \g__tag_para_begin_int
356 }
357 \cs_new_protected:Npn \__tag_gincr_para_main_end_int:
358 {
359   \int_gincr:N \g__tag_para_main_end_int
360 }
361 \cs_new_protected:Npn \__tag_gincr_para_end_int:
362 {
363   \int_gincr:N \g__tag_para_end_int
364 }
```

(End of definition for __tag_gincr_para_main_begin_int: and others.)

```
\_tag_start_para_ints:
\_tag_stop_para_ints: 365 \cs_new_protected:Npn \__tag_start_para_ints:
366 {
367   \cs_set_protected:Npn \__tag_gincr_para_main_begin_int:
368   {
369     \int_gincr:N \g__tag_para_main_begin_int
370   }
371   \cs_set_protected:Npn \__tag_gincr_para_begin_int:
372   {
373     \int_gincr:N \g__tag_para_begin_int
374   }
375   \cs_set_protected:Npn \__tag_gincr_para_main_end_int:
376   {
377     \int_gincr:N \g__tag_para_main_end_int
378   }
379   \cs_set_protected:Npn \__tag_gincr_para_end_int:
380   {
381     \int_gincr:N \g__tag_para_end_int
382   }
383 }
384 \cs_new_protected:Npn \__tag_stop_para_ints:
385 {
386   \cs_set_eq:NN \__tag_gincr_para_main_begin_int:\prg_do_nothing:
387   \cs_set_eq:NN \__tag_gincr_para_begin_int:\prg_do_nothing:
388   \cs_set_eq:NN \__tag_gincr_para_main_end_int:\prg_do_nothing:
389   \cs_set_eq:NN \__tag_gincr_para_end_int:\prg_do_nothing:
390 }
```

(End of definition for __tag_start_para_ints: and __tag_stop_para_ints:.)

We want to be able to inspect the current para main structure, so we need a command to store its structure number

```
\_tag_para_main_store_struct:
391 \cs_new:Npn \__tag_para_main_store_struct:
```

```

392 {
393   \tl_gset:Ne \g__tag_para_main_struct_tl {\int_use:N \c@g__tag_struct_abs_int }
394 }

```

(End of definition for __tag_para_main_store_struct:.)

temporary adaption for the block module:

```

395 \AddToHook{package/latex-lab-testphase-block/after}
396 {
397   \tl_if_exist:NT \l_tag_para_attr_class_tl
398   {
399     \tl_set:Nn \l__tag_para_attr_class_tl { \l_tag_para_attr_class_tl }
400   }
401 }

```

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

```

unittag (deprecated) 402 \keys_define:nn { __tag / setup }
para-flattened (deprecated) 403 {
  paratagging (deprecated) 404   para/tagging      .bool_set:N = \l__tag_para_bool,
  paratagging-show (deprecated) 405   debug/show/para   .code:n = {\bool_set_true:N \l__tag_para_show_bool},
  paratag (deprecated) 406   debug/show/paraOff .code:n = {\bool_set_false:N \l__tag_para_show_bool},
  407   para/tag          .tl_set:N   = \l__tag_para_tag_tl,
  408   para/maintag      .tl_set:N   = \l__tag_para_main_tag_tl,
  409   para/flattened     .bool_set:N = \l__tag_para_flattened_bool
  410 }
411 \keys_define:nn { tag / tool}
412 {
413   para/tagging      .bool_set:N = \l__tag_para_bool,
414   para/tag          .tl_set:N   = \l__tag_para_tag_tl,
415   para/maintag      .tl_set:N   = \l__tag_para_main_tag_tl,
416   para/flattened     .bool_set:N = \l__tag_para_flattened_bool
417 }

```

the deprecated names

```

418 \keys_define:nn { __tag / setup }
419 {
420   paratagging      .bool_set:N = \l__tag_para_bool,
421   paratagging-show .bool_set:N = \l__tag_para_show_bool,
422   paratag          .tl_set:N   = \l__tag_para_tag_tl
423 }
424 \keys_define:nn { tag / tool}
425 {
426   para      .bool_set:N = \l__tag_para_bool,
427   paratag   .tl_set:N   = \l__tag_para_tag_tl,
428   unittag   .tl_set:N   = \l__tag_para_main_tag_tl,
429   para-flattened .bool_set:N = \l__tag_para_flattened_bool
430 }

```

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

Helper command for debugging:

```

431 \cs_new_protected:Npn \__tag_check_para_begin_show:nn #1 #2
432   %#1 color, #2 prefix
433   {
434     \bool_if:NT \l__tag_para_show_bool
435     {
436       \tag_mc_begin:n{artifact}
437       \llap{\color_select:n{#1}\tiny#2\int_use:N\g__tag_para_begin_int\ }
438       \tag_mc_end:
439     }
440   }
441
442 \cs_new_protected:Npn \__tag_check_para_end_show:nn #1 #2
443   %#1 color, #2 prefix
444   {
445     \bool_if:NT \l__tag_para_show_bool
446     {
447       \tag_mc_begin:n{artifact}
448       \rlap{\color_select:n{#1}\tiny\ #2\int_use:N\g__tag_para_end_int}
449       \tag_mc_end:
450     }
451   }

```

The para/begin and para/end code. We have two variants here: a simpler one, which must be used if the block code is not used (and so probably will disappear at some time) and a more sophisticated one that must be used if the block code is used. It is possible that we will need more variants, so we setup a socket so that the code can be easily switched. This code moves into ltagging (2025/11), so we add a test for the transition.

```

452 \socket_if_exist:nF { tagsupport/para/begin }
453 {
454   \NewTaggingSocket {para/begin}{0}
455   \NewTaggingSocket {para/end}{0}
456
457   \NewTaggingSocketPlug{para/begin}{plain}
458   {
459     \bool_if:NT \l__tag_para_bool
460     {
461       \bool_if:NF \l__tag_para_flattened_bool
462       {
463         \__tag_gincr_para_main_begin_int:
464         \tag_struct_begin:n
465         {
466           tag=\l__tag_para_main_tag_tl,
467         }
468         \__tag_para_main_store_struct:
469       }
470       \__tag_gincr_para_begin_int:
471       \tag_struct_begin:n {tag=\l__tag_para_tag_tl}
472       \__tag_check_para_begin_show:nn {green}{}
473       \tag_mc_begin:n {}
474     }
475   }
476   \NewTaggingSocketPlug{para/begin}{block}
477   {

```



```

478 \bool_if:NT \l__tag_para_bool
479 {
480   \legacy_if:nF { @inlabel }
481   {
482     \__tag_check_typeout_v:n
483     {==>~ @endpe = \legacy_if:nTF { @endpe }{true}{false} \on@line }
484     \legacy_if:nF { @endpe }
485     {
486       \bool_if:NF \l__tag_para_flattened_bool
487       {
488         \__tag_gincr_para_main_begin_int:
489         \tag_struct_begin:n
490         {
491           tag=\l__tag_para_main_tag_tl,
492           attribute-class=\l__tag_para_main_attr_class_tl,
493         }
494         \__tag_para_main_store_struct:
495       }
496     }
497     \__tag_gincr_para_begin_int:
498     \__tag_check_typeout_v:n {==>~increment~ P \on@line }
499     \tag_struct_begin:n
500     {
501       tag=\l__tag_para_tag_tl
502       ,attribute-class=\l__tag_para_attr_class_tl
503     }
504     \__tag_check_para_begin_show:nn {green}{\PARALABEL}
505     \tag_mc_begin:n {}
506   }
507 }
508 }

```

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

```

509 \NewTaggingSocketPlug {para/end}{plain}
510 {
511   \bool_if:NT \l__tag_para_bool
512   {
513     \__tag_gincr_para_end_int:
514     \__tag_check_typeout_v:n {==>~increment~ /P \on@line }
515     \tag_mc_end:
516     \__tag_check_para_end_show:nn {red}{ }
517     \tag_struct_end:
518     \bool_if:NF \l__tag_para_flattened_bool
519     {
520       \__tag_gincr_para_main_end_int:
521       \tag_struct_end:
522     }
523   }
524 }
525 }

```

By default we assign the plain plug:

```

526 \AssignTaggingSocketPlug { para/begin}{plain}
527 \AssignTaggingSocketPlug { para/end}{plain}

```

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

```
528 \AddToHook{para/begin}{\tag_socket_use:n { para/begin } }
529 \AddToHook{para/end} { \tag_socket_use:n { para/end } }
```

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

```
530 \AddToHook{package/latex-lab-testphase-block/after}
531 {
532   \RemoveFromHook{para/begin}[tagpdf]
533   \RemoveFromHook{para/end}[latex-lab-testphase-block]
534   \AddToHook{para/begin}[tagpdf]
535   {
536     \socket_use:n { tagsupport/para/begin }
537   }
538   \AddToHook{para/end}[tagpdf]
539   {
540     \socket_use:n { tagsupport/para/end }
541   }
542   \socket_assign_plug:nn { tagsupport/para/begin}{block}
543 }
544
```

We check the para count at the end. If tagging is not active it is not a error, but we issue a warning as it perhaps indicates that the testphase code didn't guard everything correctly.

```
545 \AddToHook{enddocument/info}
546 {
547   \tag_if_active:F
548   {
549     \msg_redirect_name:nnn { tag } { para-hook-count-wrong } { warning }
550   }
551   \int_compare:nNnF {\g__tag_para_main_begin_int}={\g__tag_para_main_end_int}
552   {
553     \msg_error:nneee
554     {tag}
555     {para-hook-count-wrong}
556     {\int_use:N\g__tag_para_main_begin_int}
557     {\int_use:N\g__tag_para_main_end_int}
558     {text-unit}
559   }
560   \int_compare:nNnF {\g__tag_para_begin_int}={\g__tag_para_end_int}
561   {
562     \msg_error:nneee
563     {tag}
564     {para-hook-count-wrong}
565     {\int_use:N\g__tag_para_begin_int}
566     {\int_use:N\g__tag_para_end_int}
567     {text}
568   }
569 }
</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}`

```

570 <base>\newcommand\tagpdfparaOn {}
571 <base>\newcommand\tagpdfparaOff{}
572 <*package>
573 \renewcommand\tagpdfparaOn {\bool_set_true:N \l__tag_para_bool}
574 \renewcommand\tagpdfparaOff{\bool_set_false:N \l__tag_para_bool}

```

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

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

```

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

```

```

575 \NewDocumentCommand\tagpdfsuppressmarks{m}
576   {{\use:c{__tag_mc_disable_marks:} #1}}

```

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

13.5 Language support

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

test/lang (setup-key)

```

577 \keys_define:nn { __tag / setup }
578   {
579     text / lang .tl_set:N = \l__tag_struct_lang_tl
580   }

```

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

13.6 Header and footer

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

```

581 \cs_new_protected:Npn\__tag_hook_kernel_before_head:{}
582 \cs_new_protected:Npn\__tag_hook_kernel_after_head:{}
583 \cs_new_protected:Npn\__tag_hook_kernel_before_foot:{}
584 \cs_new_protected:Npn\__tag_hook_kernel_after_foot:{}

```

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

```
585 \AddToHook{begindocument}
586 {
587   \cs_if_exist:NT \@kernel@before@head
588   {
589     \tl_put_right:Nn \@kernel@before@head {\__tag_hook_kernel_before_head:}
590     \tl_put_left:Nn \@kernel@after@head {\__tag_hook_kernel_after_head:}
591     \tl_put_right:Nn \@kernel@before@foot {\__tag_hook_kernel_before_foot:}
592     \tl_put_left:Nn \@kernel@after@foot {\__tag_hook_kernel_after_foot:}
593   }
594 }
```

We use the page sockets.

```
595 \NewTaggingSocketPlug{build/page/header}{tagpdf}
596 {
597   \__tag_hook_kernel_before_head:
598   #2
599   \__tag_hook_kernel_after_head:
600 }
601
602 \AssignTaggingSocketPlug{build/page/header}{tagpdf}
603 \NewTaggingSocketPlug{build/page/footer}{tagpdf}
604 {
605   \__tag_hook_kernel_before_foot:
606   #2
607   \__tag_hook_kernel_after_foot:
608 }
609 \AssignTaggingSocketPlug{build/page/footer}{tagpdf}
610 \bool_new:N \g__tag_saved_in_mc_bool
611 \cs_new_protected:Npn \__tag_exclude_headfoot_begin:
612 {
613   \bool_set_false:N \l__tag_para_bool
614   \bool_if:NTF \g__tag_mode_lua_bool
615   {
616     \tag_mc_end_push:
617   }
618   {
619     \bool_gset_eq:NN \g__tag_saved_in_mc_bool \g__tag_in_mc_bool
620     \bool_gset_false:N \g__tag_in_mc_bool
621   }
622   \tag_mc_begin:n {artifact}
623   \tag_suspend:n{headfoot}
624 }
625 \cs_new_protected:Npn \__tag_exclude_headfoot_end:
626 {
627   \tag_resume:n{headfoot}
628   \tag_mc_end:
629   \bool_if:NTF \g__tag_mode_lua_bool
630   {
631     \tag_mc_begin_pop:n{ }
632   }
633   {
634     \bool_gset_eq:NN \g__tag_in_mc_bool \g__tag_saved_in_mc_bool
635   }
636 }
```

636 }

This version allows to use an Artifact structure

```
637 \__tag_attr_new_entry:nn {__tag/attr/pagination}{/0/Artifact/Type/Pagination}
638 \cs_new_protected:Npn \__tag_exclude_struct_headfoot_begin:n #1
639 {
640   \bool_set_false:N \l__tag_para_bool
641   \bool_if:NTF \g__tag_mode_lua_bool
642   {
643     \tag_mc_end_push:
644   }
645   {
646     \bool_gset_eq:NN \g__tag_saved_in_mc_bool \g__tag_in_mc_bool
647     \bool_gset_false:N \g__tag_in_mc_bool
648   }
649   \tag_struct_begin:n{tag=Artifact,attribute-class=__tag/attr/#1}
650   \tag_mc_begin:n {artifact=#1}
651   \tag_suspend:n{headfoot}
652 }
653
654 \cs_new_protected:Npn \__tag_exclude_struct_headfoot_end:
655 {
656   \tag_resume:n{headfoot}
657   \tag_mc_end:
658   \tag_struct_end:
659   \bool_if:NTF \g__tag_mode_lua_bool
660   {
661     \tag_mc_begin_pop:n{ }
662   }
663   {
664     \bool_gset_eq:NN \g__tag_in_mc_bool \g__tag_saved_in_mc_bool
665   }
666 }
```

And now the keys

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

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

```

685 {
686   \cs_set_eq:NN \__tag_hook_kernel_before_head: \prg_do_nothing:
687   \cs_set_eq:NN \__tag_hook_kernel_before_foot: \prg_do_nothing:
688   \cs_set_eq:NN \__tag_hook_kernel_after_head: \prg_do_nothing:
689   \cs_set_eq:NN \__tag_hook_kernel_after_foot: \prg_do_nothing:
690 },
691 page/exclude-header-footer .default:n = true,
692 page/exclude-header-footer .initial:n = true,

```

deprecated name

```

693   exclude-header-footer .meta:n = { page/exclude-header-footer = {#1} }
694 }

```

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

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

```

695 \AtBeginDocument
696 {
697   \socket_if_exist:nT{tag-support/parbox/before}
698   {
699     \NewTaggingSocketPlug{parbox/before}{tag/footer}
700     {
701       \tag_struct_begin:n{tag=Span}
702       \tag_mc_begin:n{ }
703     }
704
705     \NewTaggingSocketPlug{parbox/after}{tag/footer}
706     {
707       \tag_mc_end:
708       \tag_struct_end:
709     }
710   }
711 }
712
713 \cs_new_protected:Npn \__tag_headfoot_tagged_begin:n #1
714 {
715   \AssignTaggingSocketPlug{parbox/before}{tag/footer}
716   \AssignTaggingSocketPlug{parbox/after}{tag/footer}
717   \bool_set_false:N \l__tag_para_bool
718   \bool_if:NTF \g__tag_mode_lua_bool
719   {
720     \tag_mc_end_push:
721   }
722   {
723     \bool_gset_eq:NN \g__tag_saved_in_mc_bool \g__tag_in_mc_bool
724     \bool_gset_false:N \g__tag_in_mc_bool
725   }
726   \tag_struct_begin:n{tag=Artifact,attribute-class=__tag/attr/#1,parent=\tag_get:n{current_
727 }
728
729 \cs_new_protected:Npn \__tag_headfoot_tagged_end:
730 {
731   \tag_struct_end:

```

```

732 \bool_if:NTF \g__tag_mode_lua_bool
733 {
734   \tag_mc_begin_pop:n{ }
735 }
736 {
737   \bool_gset_eq:NN \g__tag_in_mc_bool\g__tag_saved_in_mc_bool
738 }
739 }
740 \keys_define:nn { __tag / setup }
741 {
742   page/tag-header-footer .code:n =
743   {
744     \cs_set:Nn \__tag_hook_kernel_before_head: { \__tag_headfoot_tagged_begin:n {pagination}
745     \cs_set:Nn \__tag_hook_kernel_before_foot: { \__tag_headfoot_tagged_begin:n {pagination}
746     \cs_set_eq:NN \__tag_hook_kernel_after_head: \__tag_headfoot_tagged_end:
747     \cs_set_eq:NN \__tag_hook_kernel_after_foot: \__tag_headfoot_tagged_end:
748   }
749 }

```

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 some commands that will be in the next LaTeX release (11/2025)

```

750 \AddToHookNext{class/before}
751 {
752   \cs_if_exist:NF \UseStructureName
753   {
754     \cs_new:Npn\UseStructureName#1
755     {
756       \cs:w l__tag_name_#1_tl\cs_end:
757     }
758   }
759   \cs_if_exist:NF \l__tag_name_link_tl
760   {
761     \tl_new:N \l__tag_name_link_tl
762     \tl_set:Nn \l__tag_name_link_tl{Link}
763   }
764 }

```

Tagging sockets for links

```

765 \socket_if_exist:nF {tagssupport/link/before}
766 {
767   \NewTaggingSocket{link/before}{1}
768   \NewTaggingSocket{link/after}{1}
769 }
770 \NewTaggingSocketPlug{link/before}{kernel}
771 {
772   \mode_leave_vertical:

```

```

773 \tag_mc_end_push:
774 \tag_struct_begin:n { tag=\UseStructureName{link} }
775 \tag_mc_begin:n {}
776 #1
777 }
778 \AssignTaggingSocketPlug{link/before}{kernel}
779
780 \NewTaggingSocketPlug{link/after}{kernel}
781 {
782 #1
783 \tag_mc_end:
784 \tag_struct_end:
785 \tag_mc_begin_pop:n{ }
786 }
787 \AssignTaggingSocketPlug{link/after}{kernel}
788
789
790 \bool_lazy_and:nnTF
791 { \sys_if_engine luatex_p: }
792 {
793 \tl_if_empty_p:e
794 {
795 \lua_now:e
796 { if~ luatexbase.in_callback('pre_shipout_filter','linksplit')~
797 then~else~tex.print('1')~end
798 }
799 }
800 }
801 {
802 \hook_gput_code:nnn
803 {pdfannot/link/URI/before}
804 {tagpdf}
805 {
806 \UseTaggingSocket{link/before}{ }
807 }
808
809 \hook_gput_code:nnn
810 {pdfannot/link/URI/after}
811 {tagpdf}
812 {
813 \UseTaggingSocket{link/after}{ }
814 }
815
816 \hook_gput_code:nnn
817 {pdfannot/link/GoTo/before}
818 {tagpdf}
819 {
820 \UseTaggingSocket{link/before}{ }
821 }
822
823 \hook_gput_code:nnn
824 {pdfannot/link/GoTo/after}
825 {tagpdf}
826 {

```



```

827     \UseTaggingSocket{link/after}{}
828 }
829
830 \hook_gput_code:nnn
831 {pdfannot/link/GoToR/before}
832 {tagpdf}
833 {
834     \UseTaggingSocket{link/before}
835 }
836
837 \hook_gput_code:nnn
838 {pdfannot/link/GoToR/after}
839 {tagpdf}
840 {
841     \UseTaggingSocket{link/after}{}
842 }
843
844 \hook_gput_code:nnn
845 {pdfannot/link/Launch/before}
846 {tagpdf}
847 {
848     \UseTaggingSocket{link/before}{}
849 }
850
851 \hook_gput_code:nnn
852 {pdfannot/link/Launch/after}
853 {tagpdf}
854 {
855     \UseTaggingSocket{link/after}{}
856 }
857
858 \hook_gput_code:nnn
859 {pdfannot/link/Named/before}
860 {tagpdf}
861 {
862     \UseTaggingSocket{link/before}{}
863 }
864
865 \hook_gput_code:nnn
866 {pdfannot/link/Named/after}
867 {tagpdf}
868 {
869     \UseTaggingSocket{link/after}{}
870 }
871 }
872 {
873     \hook_gput_code:nnn
874     {pdfannot/link/URI/before}
875     {tagpdf}
876     {
877         \UseTaggingSocket{link/before}
878         {
879             \pdfannot_dict_put:nne
880             { link/URI }
881             { StructParent }
882             { \tag_struct_parent_int: }

```

```

881     }
882 }
883
884 \hook_gput_code:nnn
885 {pdfannot/link/URI/after}
886 {tagpdf}
887 {
888   \UseTaggingSocket{link/after}
889   {
890     \tag_struct_insert_annot:ee
891     {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
892   }
893 }
894
895 \hook_gput_code:nnn
896 {pdfannot/link/GoTo/before}
897 {tagpdf}
898 {
899   \UseTaggingSocket{link/before}
900   {
901     \pdfannot_dict_put:nne
902     { link/GoTo }
903     { StructParent }
904     { \tag_struct_parent_int: }
905   }
906 }
907
908 \hook_gput_code:nnn
909 {pdfannot/link/GoTo/after}
910 {tagpdf}
911 {
912   \UseTaggingSocket{link/after}
913   {
914     \tag_struct_insert_annot:ee
915     {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
916   }
917 }
918
919 \hook_gput_code:nnn
920 {pdfannot/link/GoToR/before}
921 {tagpdf}
922 {
923   \UseTaggingSocket{link/before}
924   {
925     \pdfannot_dict_put:nne
926     { link/GoToR }
927     { StructParent }
928     { \tag_struct_parent_int: }
929   }
930 }
931
932 \hook_gput_code:nnn
933 {pdfannot/link/GoToR/after}
934 {tagpdf}

```

```

935     {
936         \UseTaggingSocket{link/after}
937         {
938             \tag_struct_insert_annot:ee
939             {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
940         }
941     }
942
943     \hook_gput_code:nnn
944     {pdfannot/link/Named/before}
945     {tagpdf}
946     {
947         \UseTaggingSocket{link/before}
948         {
949             \pdfannot_dict_put:nne
950             { link/Named }
951             { StructParent }
952             { \tag_struct_parent_int: }
953         }
954     }
955
956     \hook_gput_code:nnn
957     {pdfannot/link/Named/after}
958     {tagpdf}
959     {
960         \UseTaggingSocket{link/after}
961         {
962             \tag_struct_insert_annot:ee
963             {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
964         }
965     }
966
967     \hook_gput_code:nnn
968     {pdfannot/link/Launch/before}
969     {tagpdf}
970     {
971         \UseTaggingSocket{link/before}
972         {
973             \pdfannot_dict_put:nne
974             { link/Launch }
975             { StructParent }
976             { \tag_struct_parent_int: }
977         }
978     }
979
980     \hook_gput_code:nnn
981     {pdfannot/link/Launch/after}
982     {tagpdf}
983     {
984         \UseTaggingSocket{link/after}
985         {
986             \tag_struct_insert_annot:ee
987             {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
988         }
989     }

```

13.8 Attaching css-files for derivation

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

Such CSS style definitions can be given in two ways:

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

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

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

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

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

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

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

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

```

990 \prop_new:N \g__tag_css_prop
991 \prop_gset_from_keyval:Nn \g__tag_css_prop
992 {
993   latex-list-css.html=true,
994   latex-align-css.html=true
995 }
996
997
998 \bool_new:N \g__tag_css_bool
999 \bool_gset_true:N \g__tag_css_bool

```

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

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

```

1000 \tl_gput_left:Nn \g__kernel_pdfmanagement_end_run_code_tl
1001 {
1002   \bool_lazy_and:nnT { \g__tag_css_bool }{ \tag_if_active_p: }
1003   {
1004     \prop_map_inline:Nn \g__tag_css_prop
1005     {
1006       \keys_set:nn { __tag / setup }{ catalog-supplemental-file= {#1} }
1007     }
1008   }
1009 }
1010
1011 \keys_define:nn { __tag / setup }
1012 {
1013   attach-css .bool_gset:N = \g__tag_css_bool,
1014   css-list .code:n =
1015   {
1016     \tl_if_empty:nTF{#1}
1017     { \prop_gclear:N \g__tag_css_prop }
1018     { \prop_gput:Nnn \g__tag_css_prop { #1 }{true}}
1019   },
1020   css-list-add .code:n = { \prop_gput:Nnn \g__tag_css_prop { #1 }{true} },
1021   css-list-remove .code:n = { \prop_gremove:Nn \g__tag_css_prop { #1 } },
1022 }

```

</package> The tagpdf-tree module
 Commands trees and main dictionaries
 Part of the tagpdf package
 Ulrike Fischer
 Version 0.99y, released 2026-01-29

Part IV

```
1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-tree-code} {2026-01-29} {0.99y}
4 {part of tagpdf - code related to writing trees and dictionaries to the pdf}
5 </header>
```

1 Trees, pdfmanagement and finalization code

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

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

1.1 Check structure

`__tag_tree_final_checks:`

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

(End of definition for `__tag_tree_final_checks:`.)

1.2 Catalog: MarkInfo and StructTreeRoot and OpenAction

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

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

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

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

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

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

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

(End of definition for \g__tag_tree_openaction_struct_tl.)

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

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

```
35 {
```

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

```
37   {
```

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

```
39   }
```

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

```
41 }
```

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

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

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

```
43 {
```

```
44   \prop_get:cnNT
```

```
45   { \__kernel_pdfdict_name:n { g__pdf_Core/Catalog } }
```

```
46   {OpenAction}
```

```
47   \l__tag_tmpa_tl
```

```
48   {
```

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

```
49   \tl_if_head_eq_charcode:eNT { \tl_trim_spaces:o { \l__tag_tmpa_tl } } [ %]
```

```
50   {
```

```
51     \seq_set_split:Nno\l__tag_tmpa_seq {/} { \l__tag_tmpa_tl }
```

```
52     \pdfmanagement_add:nne {Catalog} { OpenAction }
```

```
53     {
```

```
54       <<
```

```
55       /S/GoTo \c_space_tl
```

```
56       /D~\l__tag_tmpa_tl\c_space_tl
```

```
57       /SD~[\pdf_object_ref_indexed:nn{__tag/struct}{\g__tag_tree_openaction_struct
```

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

```
58       \int_compare:nNnTF{ \seq_count:N \l__tag_tmpa_seq } > {1}
```

```
59       { /\seq_item:Nn\l__tag_tmpa_seq{2} }
```

```
60       { ] }
```

```
61     >>
```

```
62   }
```

```
63 }
```

```
64 }
```

```
65 }
```

```

66 \hook_gput_code:nnn{shipout/lastpage}{tagpdf}
67 {
68   \bool_if:NT \g__tag_active_tree_bool
69   {
70     \pdfmanagement_add:nnn { Catalog / MarkInfo } { Marked } { true }
71     \pdfmanagement_add:nne
72       { Catalog }
73       { StructTreeRoot }
74       { \pdf_object_ref_indexed:nn { __tag/struct } { 1 } }
75     \__tag_tree_update_openaction:
76   }
77 }

```

1.3 Writing the IDtree

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

\g__tag_tree_id_pad_int

```

78 \int_new:N\g__tag_tree_id_pad_int
(End of definition for \g__tag_tree_id_pad_int.)

```

Now we get the needed padding

```

79 \cs_generate_variant:Nn \tl_count:n {e}
80 \hook_gput_code:nnn{begindocument}{tagpdf}
81 {
82   \int_gset:Nn\g__tag_tree_id_pad_int
83     {\tl_count:e { \__tag_property_ref_lastpage:nn{tagstruct}{1000}}+1}
84 }
85

```

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

```

86 \cs_new_protected:Npn \__tag_tree_write_idtree:
87 {
88   \tl_clear:N \l__tag_tmpa_tl
89   \tl_clear:N \l__tag_tmpb_tl
90   \int_zero:N \l__tag_tmpa_int
91   \int_step_inline:nnn {2} {\c@g__tag_struct_abs_int}
92   {
93     \int_incr:N\l__tag_tmpa_int
94     \tl_put_right:Ne \l__tag_tmpa_tl
95     {
96       \__tag_struct_get_id:n{##1}~\pdf_object_ref_indexed:nn {__tag/struct}{##1}~
97     }
98     \int_compare:nNnF {\l__tag_tmpa_int}<{50} %
99     {
100       \pdf_object_unnamed_write:ne {dict}
101       { /Limits~[\__tag_struct_get_id:n{##1}-\l__tag_tmpa_int+1}~\__tag_struct_get_id:
102         /Names~[\l__tag_tmpa_tl]
103     }
104     \tl_put_right:Ne\l__tag_tmpb_tl {\pdf_object_ref_last:\c_space_tl}

```



```

105         \int_zero:N \l__tag_tmpa_int
106         \tl_clear:N \l__tag_tmpa_tl
107     }
108 }
109 \tl_if_empty:NF \l__tag_tmpa_tl
110 {
111     \pdf_object_unnamed_write:ne {dict}
112     {
113         /Limits~
114         [\_tag_struct_get_id:n{\c@g__tag_struct_abs_int-\l__tag_tmpa_int+1}~
115         \_tag_struct_get_id:n{\c@g__tag_struct_abs_int}]
116         /Names~[\l__tag_tmpa_tl]
117     }
118     \tl_put_right:Ne\l__tag_tmpb_tl {\pdf_object_ref_last:}
119 }
120 \pdf_object_unnamed_write:ne {dict}{/Kids~[\l__tag_tmpb_tl]}
121 \_tag_prop_gput:cne
122 { g__tag_struct_1_prop }
123 { IDTree }
124 { \pdf_object_ref_last: }
125 }

```

1.4 Writing structure elements

The following commands are needed to write out the structure.

`_tag_tree_write_structtreeroot:` This writes out the root object.

```

126 \cs_new_protected:Npn \_tag_tree_write_structtreeroot:
127 {
128     \_tag_prop_gput:cne
129     { g__tag_struct_1_prop }
130     { ParentTree }
131     { \pdf_object_ref:n { __tag/tree/parenttree } }
132     \_tag_prop_gput:cne
133     { g__tag_struct_1_prop }
134     { RoleMap }
135     { \pdf_object_ref:n { __tag/tree/rolemap } }
136     \_tag_struct_fill_kid_key:n { 1 }
137     \prop_gremove:cn { g__tag_struct_1_prop } {S}
138     \_tag_struct_get_dict_content:nN { 1 } \l__tag_tmpa_tl
139     \pdf_object_write_indexed:nnne
140     { __tag/struct } { 1 }
141     {dict}
142     {
143         \l__tag_tmpa_tl
144     }

```

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

```

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

```

(End of definition for `_tag_tree_write_structtreeroot:.`)

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

```

147 \cs_new_protected:Npn \_tag_tree_write_structelements:

```

```

148 {
149   \int_step_inline:nnnn {2}{1}{\c@_tag_struct_abs_int}
150   {
151     \__tag_struct_write_obj:n { ##1 }
152   }
153 }

```

(End of definition for __tag_tree_write_structelements:.)

1.5 ParentTree

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

```

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

```

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

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

`\c@_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@_tag_parenttree_obj_int
156 \hook_gput_code:nnn{begindocument}{tagpdf}
157 {
158   \int_gset:Nn
159     \c@_tag_parenttree_obj_int
160     { \__tag_property_ref_lastpage:nn{abspage}{100} }
161 }

```

(End of definition for \c@_tag_parenttree_obj_int.)

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

`\g__tag_parenttree_objr_tl`

```

162 \tl_new:N \g__tag_parenttree_objr_tl

```

(End of definition for \g__tag_parenttree_objr_tl.)

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

```

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

```

(End of definition for __tag_parenttree_add_objr:nn.)

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

```

170 \tl_new:N \l__tag_parenttree_content_tl

```

(End of definition for \l__tag_parenttree_content_tl.)

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

```

171 \cs_new_protected:Npn \_tag_tree_parenttree_rerun_msg: {}
172 \cs_new_protected:Npn \_tag_tree_fill_parenttree:
173 {
174   \int_step_inline:nnnn{1}{1}{\_tag_property_ref_lastpage:nn{abspage}{-1}} %not quite clear
175   { %page ##1
176     \prop_clear:N \l__tag_tmpa_prop
177     \int_step_inline:nnnn{1}{1}{\_tag_property_ref_lastpage:nn{tagmcabs}{-
178       1}}
179     {
180       %mcid###1
181       \int_compare:nT
182       {\property_ref:enn{mcid-###1}{tagabspage}{-1}=##1} %mcid is on current page
183       {% yes
184         \prop_get:NnNT
185         \g__tag_mc_parenttree_prop
186         {###1}
187         \l__tag_tmpa_tl
188         {
189           \prop_put:Nee
190           \l__tag_tmpa_prop
191           {\property_ref:enn{mcid-###1}{tagmcid}{-1}}
192           {\l__tag_tmpa_tl}
193         }
194       }
195       \tl_put_right:Ne\l__tag_parenttree_content_tl
196       {
197         \int_eval:n {##1-1}\c_space_tl
198         [\c_space_tl %]
199       }
200       \int_step_inline:nnnn %###1
201       {0}
202       {1}
203       { \prop_count:N \l__tag_tmpa_prop -1 }
204       {
205         \prop_get:NnNTF \l__tag_tmpa_prop {###1} \l__tag_tmpa_tl
206         {% page#1:mcid##1:\l__tag_tmpa_tl :content
207         \tl_put_right:Ne \l__tag_parenttree_content_tl
208         {
209           \prop_if_exist:cTF { g__tag_struct_ \l__tag_tmpa_tl _prop }
210           {
211             \pdf_object_ref_indexed:nn { __tag/struct }{ \l__tag_tmpa_tl }
212           }
213           {
214             null
215           }
216           \c_space_tl
217         }
218       }
219       {
220         \cs_set_protected:Npn \_tag_tree_parenttree_rerun_msg:
221         {

```

```

222             \msg_warning:nn { tag } {tree-mcid-index-wrong}
223         }
224     }
225 }
226 \tl_put_right:Nn
227   \l__tag_parenttree_content_tl
228   {%[
229     ]^^J
230   }
231 }
232 }

```

(End of definition for __tag_tree_fill_parenttree:.)

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

```

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

```

(End of definition for __tag_tree_lua_fill_parenttree:.)

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

```

246 \cs_new_protected:Npn \__tag_tree_write_parenttree:
247 {
248   \bool_if:NTF \g__tag_mode_lua_bool
249   {
250     \__tag_tree_lua_fill_parenttree:
251   }
252   {
253     \__tag_tree_fill_parenttree:
254   }
255   \__tag_tree_parenttree_rerun_msg:
256   \tl_put_right:No \l__tag_parenttree_content_tl { \g__tag_parenttree_objr_tl }
257   \pdf_object_write:nne { __tag/tree/parenttree }{dict}
258   {
259     /Nums\c_space_tl [\l__tag_parenttree_content_tl]
260   }
261 }

```

(End of definition for __tag_tree_write_parenttree:.)

1.6 Rolemap dictionary

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

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

```
262 \pdf_object_new:n { __tag/tree/rolemap }  
(End of definition for __tag/tree/rolemap.)
```

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

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

1.7 Classmap dictionary

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

`__tag_tree_write_classmap:`

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

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

```
289   \seq_map_inline:Nn \g__tag_attr_class_used_seq  
290   {  
291     \prop_gput:Nnn \g__tag_attr_class_used_prop {##1}{}  
292   }  
293   \prop_map_inline:Nn \g__tag_attr_class_used_prop  
294   {  
295     \prop_get:NnNT \g__tag_attr_entries_prop {##1} \l__tag_tmpb_tl  
296     {  
297       \tl_put_right:Ne \l__tag_tmpa_tl  
298       {
```

```

299         ##1\c_space_tl
300         <<
301         \l__tag_tmpb_tl
302         >>
303         \iow_newline:
304     }
305 }
306 }
307 \tl_if_empty:NF
308 \l__tag_tmpa_tl
309 {
310     \pdf_object_new:n { __tag/tree/classmap }
311     \pdf_object_write:nne
312     { __tag/tree/classmap }
313     {dict}
314     { \l__tag_tmpa_tl }
315     \__tag_prop_gput:cne
316     { g__tag_struct_1_prop }
317     { ClassMap }
318     { \pdf_object_ref:n { __tag/tree/classmap } }
319 }
320 }

```

(End of definition for __tag_tree_write_classmap:.)

1.8 Namespaces

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

__tag/tree/namespaces

```

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

```

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

__tag_tree_write_namespaces:

```

322 \cs_new_protected:Npn \__tag_tree_write_namespaces:
323 {
324     \pdf_version_compare:NnF < {2.0}
325     {
326         \prop_map_inline:Nn \g__tag_role_NS_prop
327         {
328             \pdfdict_if_empty:NF {g__tag_role/RoleMapNS_##1_dict}
329             {
330                 \pdf_object_write:nne {__tag/RoleMapNS/##1}{dict}
331                 {
332                     \pdfdict_use:n {g__tag_role/RoleMapNS_##1_dict}
333                 }
334                 \pdfdict_gput:nne{g__tag_role/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 \</package>
(End of definition for \__tag_finish_structure:.)

```

1.10 StructParents entry for Page

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

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

The tagpdf-mc-shared module

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

Part of the tagpdf package

Ulrike Fischer

Version 0.99y, released 2026-01-29

Part V

1 Public Commands

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

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

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

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

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

New: 2019-11-20

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

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

New: 2021-04-22

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

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

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

New: 2023-06-11

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

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

New: 2024-11-18

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

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

New: 2024-11-18

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

2 Public keys

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

3 Marked content code – shared

```

1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-mc-code-shared} {2026-01-29} {0.99y}
4   {part of tagpdf - code related to marking chunks -
5    code shared by generic and luamode }
6 </header>

```

3.1 Variables and counters

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

```

g__tag_MCID_abs_int
7 <*base>
8 \newcounter { g__tag_MCID_abs_int }
(End of definition for g__tag_MCID_abs_int.)

```

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

```

9 \cs_new:Npn \__tag_get_data_mc_counter:
10   {
11     \int_use:N \c@g__tag_MCID_abs_int
12   }
13 </base>
(End of definition for \__tag_get_data_mc_counter:.)

```

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

```

14 <*shared>
15 \cs_new:Npn \__tag_get_mc_abs_cnt: { \int_use:N \c@g__tag_MCID_abs_int }
(End of definition for \__tag_get_mc_abs_cnt:.)

```

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

```

16 \bool_new:N \g__tag_in_mc_bool
(End of definition for \g__tag_in_mc_bool.)

```

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

```

17 \__tag_prop_new_linked:N \g__tag_mc_parenttree_prop
(End of definition for \g__tag_mc_parenttree_prop.)

```

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

```

18 \seq_new:N \g__tag_mc_stack_seq
(End of definition for \g__tag_mc_parenttree_prop.)

```

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

```

19 \tl_new:N \l__tag_mc_artifact_type_tl
(End of definition for \l__tag_mc_artifact_type_tl.)

```

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

```

20 \bool_new:N \l__tag_mc_key_stash_bool
21 \bool_new:N \l__tag_mc_artifact_bool
(End of definition for \l__tag_mc_key_stash_bool and \l__tag_mc_artifact_bool.)

```

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

```

22 \tl_new:N \l__tag_mc_lang_tl
(End of definition for \l__tag_mc_lang_tl.)

```

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

```

23 \tl_new:N \l__tag_mc_key_tag_tl
24 \tl_new:N \g__tag_mc_key_tag_tl
25 \tl_new:N \l__tag_mc_key_label_tl
26 \tl_new:N \l__tag_mc_key_properties_tl
(End of definition for \l__tag_mc_key_tag_tl and others.)

```

3.2 Functions

`__tag_mc_handle_mc_label:e` The commands labels a mc-chunk. It is used if the user explicitly labels the mc-chunk with the `label` key. The argument is the value provided by the user. It stores the attributes
tagabspage: the absolute page, `\g_shipout_readonly_int`,
tagmcabs: the absolute mc-counter `\c@g_@@_MCID_abs_int`. The reference command is based on `l3ref`.

```

27 \cs_new:Npn \__tag_mc_handle_mc_label:e #1
28 {
29   \__tag_property_record:en{tagpdf-#1}{tagabspage,tagmcabs}
30 }
(End of definition for \__tag_mc_handle_mc_label:e.)

```

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

```

31 \cs_new_protected:Npn \__tag_mc_set_label_used:n #1 %#1 labelname
32 {
33   \tl_new:c { g__tag_mc_label_\tl_to_str:n{#1}_used_tl }
34 }
35 </shared>
(End of definition for \__tag_mc_set_label_used:n.)

```

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

TODO: is testing for struct the right test?

```

36 <base>\cs_new_protected:Npn \tag_mc_use:n #1 { \__tag_whatsits: }
37 <*shared>
38 \cs_set_protected:Npn \tag_mc_use:n #1 %#1: label name
39 {
40   \__tag_check_if_active_struct:T
41   {
42     \tl_set: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 81.)

```

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

```

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 81.)

\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 81.)

\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 81.)

__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_tmppb_tl
164     \__tag_role_check_parent_child:ooooN
165     { \l__tag_get_parent_tmpa_tl }
166     { \l__tag_get_parent_tmppb_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_tmppb_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 83.)

```

220 </shared>

```

The tagpdf-mc-generic module

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

Part of the tagpdf package

Ulrike Fischer

Version 0.99y, released 2026-01-29

Part VI

1 Marked content code – generic mode

```
1 <@@=tag>
2 <*generic>
3 \ProvidesExplPackage {tagpdf-mc-code-generic} {2026-01-29} {0.99y}
4 {part of tagpdf - code related to marking chunks - generic mode}
5 </generic>
6 <*debug>
7 \ProvidesExplPackage {tagpdf-debug-generic} {2026-01-29} {0.99y}
8 {part of tagpdf - debugging code related to marking chunks - generic mode}
9 </debug>
```

1.1 Variables

```
10 <*generic>
```

`\l__tag_mc_ref_abspage_tl` We need a ref-label system to ensure that the MCID cnt restarts at 0 on a new page This will be used to store the tagabspage attribute retrieved from a label.

```
11 \tl_new:N \l__tag_mc_ref_abspage_tl
(End of definition for \l__tag_mc_ref_abspage_tl.)
```

`\l__tag_mc_tmpa_tl` temporary variable

```
12 \tl_new:N \l__tag_mc_tmpa_tl
(End of definition for \l__tag_mc_tmpa_tl.)
```

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

```
13 \newmarks \g__tag_mc_marks
(End of definition for \g__tag_mc_marks.)
```

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

```
14 \seq_new:N \g__tag_mc_main_marks_seq
15 \seq_new:N \g__tag_mc_footnote_marks_seq
16 \seq_new:N \g__tag_mc_multicol_marks_seq
(End of definition for \g__tag_mc_main_marks_seq, \g__tag_mc_footnote_marks_seq, and \g__tag_mc_multicol_marks_seq.)
```

`\tag_mc_new_stream:n`

```
17 \cs_new_protected:Npn \tag_mc_new_stream:n #1
18 {
19   \seq_new:c { g__tag_mc_multicol_#1_seq }
20 }
```

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

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

```

21 \seq_new:N \l__tag_mc_firstmarks_seq
22 \seq_new:N \l__tag_mc_botmarks_seq

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

```

1.2 Functions

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

```

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

```

```

62
63 \cs_new_protected:Npn \__tag_mc_end_marks:
64 {
65   \tex_marks:D \g__tag_mc_marks
66   {
67     e-, %first of end pair
68     \int_use:N\c@g__tag_MCID_abs_int, %mc-num
69     \g__tag_struct_stack_current_tl, %structure num
70   }
71   \tex_marks:D \g__tag_mc_marks
72   {
73     e+, %second of end pair
74     \int_use:N\c@g__tag_MCID_abs_int, %mc-num
75     \g__tag_struct_stack_current_tl, %structure num
76   }
77 }

```

(End of definition for __tag_mc_begin_marks:nn, __tag_mc_artifact_begin_marks:n, and __tag_mc_end_marks:.)

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

```

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

```

(End of definition for __tag_mc_disable_marks:.)

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

```

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

```

(End of definition for __tag_mc_get_marks:.)

__tag_mc_store:nnn This inserts the mc-chunk $\langle mc-num \rangle$ into the structure struct-num after the $\langle mc-prev \rangle$. The structure must already exist. The additional mcid dictionary is stored in a property. The item is retrieved when the kid entry is built. We test if there is already an addition and append if needed.

```

93 \cs_new_protected:Npn \__tag_mc_store:nnn #1 #2 #3 %#1 mc-prev, #2 mc-num #3 structure-
    num
94 {
95   %\prop_show:N \g__tag_struct_cont_mc_prop
96   \prop_get:NnNTF \g__tag_struct_cont_mc_prop {#1} \l__tag_tmpa_tl
97   {
98     \prop_gput:Nne \g__tag_struct_cont_mc_prop {#1}{ \l__tag_tmpa_tl \__tag_struct_mcid_c
99   }

```

```

100     {
101       \prop_gput:Nne \g__tag_struct_cont_mc_prop {#1}{ \__tag_struct_mcid_dict:n {#2}}
102     }
103     \prop_gput:Nee \g__tag_mc_parenttree_prop
104       {#2}
105       {#3}
106   }
107   \cs_generate_variant:Nn \__tag_mc_store:nnn {eee}
(End of definition for \__tag_mc_store:nnn.)

```

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

```

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

```

```

142         }
143         {
144             %stashed -> warning!!
145         }
146     }
147 }
148 {
149     \__tag_check_typeout_v:n {=>~ TMB~ not~ missing}
150 }
151 }
152
153 \cs_new_protected:Npn \__tag_mc_insert_extra_tme:n #1 % #1 stream, eg. main or footnote
154 {
155     \__tag_check_if_mc_tme_missing:TF
156     {
157         \__tag_check_typeout_v:n {=>~ TME~ ~ missing~ --- inserted}
158         \__tag_mc_emc:
159         \seq_gset_eq:cN
160         { g__tag_mc_#1_marks_seq }
161         \l__tag_mc_botmarks_seq
162     }
163     {
164         \__tag_check_typeout_v:n {=>~ TME~ not~ missing}
165     }
166 }

```

(End of definition for __tag_mc_insert_extra_tmb:n and __tag_mc_insert_extra_tme:n.)

1.3 Looking at MC marks in boxes

__tag_add_missing_mcs:Nn Assumptions:

- test for tagging active outside;
- mark retrieval also outside.

This takes a box register as its first argument (or the register number in a count register, as used by multicol). It adds an extra tmb at the top of the box if necessary and similarly an extra tme at the end. This is done by adding hboxes in a way that the positioning and the baseline of the given box is not altered. The result is written back to the box. The second argument is the stream this box belongs to und is currently either `main` for the main galley, `footnote` for footnote note text, or `multicol` for boxes produced for columns in that environment. Other streams may follow over time.

```

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

```

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

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

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

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

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

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

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

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

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

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

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

```
\tag_mc_add_missing_to_stream:Nn
\__tag_add_missing_mcs_to_stream:Nn
```

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

If we aren't in the main stream then processing is a bit more complicated because to get at the marks in the box we need to artificially split it and then look at the split marks. First argument is the box to update and the second is the "stream". In lua mode the command is a no-op.

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

First set up a temp box for trial splitting.

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

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

```
194 \vbox_set_split_to_ht:NNn \l__tag_tmpa_box \l__tag_tmpa_box \c_max_dim
```

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

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

Some debugging info:

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

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

```

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

```

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

```

208   \seq_set_eq:NN \l__tag_mc_botmarks_seq \l__tag_mc_firstmarks_seq
209 }

```

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

```

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

```

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

```

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

```

(End of definition for `\tag_mc_add_missing_to_stream:Nn` and `__tag_add_missing_mcs_to_stream:Nn`. This function is documented on page 82.)

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

`__tag_mc_if_in:TF`

`\tag_mc_if_in_p:`

`\tag_mc_if_in:TF`

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

```

226 \prg_new_conditional:Nnn \__tag_mc_if_in: {p,T,F,TF}
227 {
228   \bool_if:NTF \g__tag_in_mc_bool

```



```

229     { \prg_return_true: }
230     { \prg_return_false: }
231   }
232
233 \prg_new_eq_conditional:NNn \tag_mc_if_in: \__tag_mc_if_in: {p,T,F,TF}
(End of definition for \__tag_mc_if_in:TF and \tag_mc_if_in:TF. This function is documented on page
81.)

```

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

```

234 % #1 tag, #2 properties
235 \cs_set_eq:NN \__tag_mc_bmc:n \pdf_bmc:n
236 \cs_set_eq:NN \__tag_mc_emc: \pdf_emc:
237 \cs_set_eq:NN \__tag_mc_bdc:nn \pdf_bdc:nn
238 \cs_set_eq:NN \__tag_mc_bdc_shipout:ee \pdf_bdc_shipout:ee
(End of definition for \__tag_mc_bmc:n, \__tag_mc_emc:, and \__tag_mc_bdc:nn.)

```

```

\__tag_mc_bdc_mcid:nn
\__tag_mc_bdc_mcid:n
\__tag_mc_handle_mcid:nn
\__tag_mc_handle_mcid:oo

```

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

```

239 \hook_gput_code:nnn {shipout/before}{tagpdf}{\flag_clear:n { __tag/mcid } }
240 \cs_set_protected:Npn \__tag_mc_bdc_mcid:nn #1 #2
241 {
242   \int_gincr:N \c@g__tag_MCID_abs_int
243   \__tag_property_record:eo
244   {
245     mcid-\int_use:N \c@g__tag_MCID_abs_int
246   }
247   { \c__tag_property_mc_clist }
248   \__tag_mc_bdc_shipout:ee
249   {#1}
250   {
251     /MCID~\flag_height:n { __tag/mcid }
252     \flag_raise:n { __tag/mcid }~ #2
253   }
254 }
255 \cs_new_protected:Npn \__tag_mc_bdc_mcid:n #1
256 {
257   \__tag_mc_bdc_mcid:nn {#1} {}
258 }
259
260 \cs_new_protected:Npn \__tag_mc_handle_mcid:nn #1 #2 %#1 tag, #2 properties
261 {
262   \__tag_mc_bdc_mcid:nn {#1} {#2}
263 }
264

```

```

265 \cs_generate_variant:Nn \__tag_mc_handle_mcid:nn {oo}
(End of definition for \__tag_mc_bdc_mcid:nn, \__tag_mc_bdc_mcid:n, and \__tag_mc_handle-
mcid:nn.)

```

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

```

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

```

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

```

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

```

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

```

293 \cs_new:Nn \__tag_get_data_mc_tag: { \g__tag_mc_key_tag_tl }
294 </generic>
(End of definition for \__tag_get_data_mc_tag:.)

```

\tag_mc_begin:n These are the core public commands to open and close an mc. They don't need to be in the same group or grouping level, but the code expect that they are issued linearly.

\tag_mc_end:

The tag and the state is passed to the end command through a global var and a global boolean.

```

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

```

set default MC tags to structure:

```

314   \tl_set_eq:NN \l__tag_mc_key_tag_tl \g__tag_struct_tag_tl
315   \tl_gset_eq:NN \g__tag_mc_key_tag_tl \g__tag_struct_tag_tl
316   \tl_if_empty:NTF \l__tag_mc_lang_tl
317   {
318     \keys_set:nn { __tag / mc }{ #1 }
319   }
320   {
321     \keys_set:nn { __tag / mc }{ lang=\l__tag_mc_lang_tl, #1 }
322   }
323   \bool_if:NTF \l__tag_mc_artifact_bool
324   { %handle artifact
325     \__tag_mc_handle_artifact:N \l__tag_mc_artifact_type_tl
326     \exp_args:No
327     \__tag_mc_artifact_begin_marks:n { \l__tag_mc_artifact_type_tl }
328   }
329   { %handle mcid type
330     \__tag_check_mc_tag:N \l__tag_mc_key_tag_tl
331     \__tag_mc_handle_mcid:oo
332     { \l__tag_mc_key_tag_tl }
333     { \l__tag_mc_key_properties_tl }
334     \__tag_mc_begin_marks:oo{ \l__tag_mc_key_tag_tl }{ \l__tag_mc_key_label_tl }
335     \tl_if_empty:NF { \l__tag_mc_key_label_tl }
336     {
337       \__tag_mc_handle_mc_label:e { \l__tag_mc_key_label_tl }
338     }

```

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

```

339   \bool_if:NF \l__tag_mc_key_stash_bool
340   {
341     \socket_use:nn{tag/check/parent-child}
342     {
343       \__tag_mc_check_parent_child:o

```

```

344         { \g__tag_struct_stack_current_t1 }
345     }
346     \__tag_mc_handle_stash:e { \int_use:N \c@g__tag_MCID_abs_int }
347
348     }
349 }
350 \group_end:
351 }
352 <*debug>
353 {
354     \__tag_debug_mc_begin_ignore:n { #1 }
355 }
356 </debug>
357 }
358 <*generic>
359 \cs_set_protected:Nn \tag_mc_end:
360 {
361     \__tag_check_if_active_mc:T
362     {
363 </generic>
364 <*debug>
365 \cs_set_protected:Nn \tag_mc_end:
366 {
367     \__tag_check_if_active_mc:TF
368     {
369         \__tag_debug_mc_end_insert:
370 </debug>
371     \__tag_check_mc_if_open:
372         \bool_gset_false:N \g__tag_in_mc_bool
373         \tl_gset:Nn \g__tag_mc_key_tag_tl { }
374         \__tag_mc_emc:
375         \__tag_mc_end_marks:
376     }
377 <*debug>
378     {
379         \__tag_debug_mc_end_ignore:
380     }
381 </debug>
382 }
383 </generic | debug>

```

(End of definition for \tag_mc_begin:n and \tag_mc_end:. These functions are documented on page 81.)

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) 384 <*generic>
alt (mc-key) 385 \keys_define:nn { __tag / mc }
actualtext (mc-key) 386 {
label (mc-key) 387     tag .code:n = % the name (H,P,Span) etc
artifact (mc-key) 388     {

```

```

389     \tl_set:Ne \l__tag_mc_key_tag_tl { #1 }
390     \tl_gset:Ne \g__tag_mc_key_tag_tl { #1 }
391   },
392   raw .code:n =
393   {
394     \tl_put_right:Ne \l__tag_mc_key_properties_tl { #1 }
395   },
396   alt .code:n = % Alt property
397   {
398     \str_set_convert:Noon
399     \l__tag_tmpa_str
400     { #1 }
401     { default }
402     { utf16/hex }
403     \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
404     \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
405   },
406   alttext .meta:n = {alt=#1},

```

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

```

407   lang .code:n = % Lang property
408   {
409     \tl_put_right:Ne \l__tag_mc_key_properties_tl { /Lang~(##1) }
410   },
411   actualtext .code:n = % ActualText property
412   {
413     \tl_if_empty:oF{##1}
414     {
415       \str_set_convert:Noon
416       \l__tag_tmpa_str
417       { ##1 }
418       { default }
419       { utf16/hex }
420       \tl_put_right:Nn \l__tag_mc_key_properties_tl { /ActualText~< }
421       \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
422     }
423   },
424   label .tl_set:N = \l__tag_mc_key_label_tl,
425   artifact .code:n =
426   {
427     \exp_args:Nne
428     \keys_set:nn
429     { __tag / mc }
430     { __artifact-bool, __artifact-type=##1 }
431   },
432   artifact .default:n = {notype}
433 }
434 </generic>

```

(End of definition for tag (mc-key) and others. These functions are documented on page 82.)

The tagpdf-mc-luacode module

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

Part of the tagpdf package

Ulrike Fischer
Version 0.99y, released 2026-01-29

Part VII

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

1 Marked content code – luamode code

luamode uses attributes to mark mc-chunks. The two attributes used are defined in the backend file. The backend also load the lua file, as it can contain functions needed elsewhere. The attributes for mc are global (between 0.6 and 0.81 they were local but this was reverted). The attributes are setup only in lua, and one should use the lua functions to set and get them.

`g_@@_mc_type_attr`: the value represent the type

`g_@@_mc_cnt_attr`: will hold the `\c@g_@@_MCID_abs_int` value

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

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

tag : the type (a string)

raw : more properties (string)

label: a string.

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

kids: a array of tables

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

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

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

```
1 <@@=tag>
2 <*luamode>
3 \ProvidesExplPackage {tagpdf-mc-code-lua} {2026-01-29} {0.99y}
4   {tagpdf - mc code only for the luamode }
5 </luamode>
6 <*debug>
7 \ProvidesExplPackage {tagpdf-debug-lua} {2026-01-29} {0.99y}
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 82.)

`_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 81.)

`_tag_mc_lua_set_mc_type_attr:n`

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

`_tag_mc_lua_set_mc_type_attr:o`

`_tag_mc_lua_unset_mc_type_attr:`

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

```
84 {
```

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

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

```
87   \lua_now:e
```

```
88   {
```

```
89     tex.setattribute
```

```
90     (
```

```
91       "global",
```

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

```
93       \l__tag_tmpa_tl
```

```
94     )
```

```
95   }
```

```
96   \lua_now:e
```

```
97   {
```

```
98     tex.setattribute
```

```
99     (
```

```
100       "global",
```

```
101       luatexbase.attributes.g__tag_mc_cnt_attr,
```

```
102       \_tag_get_mc_abs_cnt:
```

```
103     )
```

```
104   }
```

```
105 }
```

```
106
```

```
107 \cs_generate_variant:Nn \_tag_mc_lua_set_mc_type_attr:n { o }
```

```
108
```

```
109 \cs_new:Nn \_tag_mc_lua_unset_mc_type_attr:
```

```
110 {
```

```

111 \lua_now:e
112 {
113     tex.setattribute
114     (
115         "global",
116         luatexbase.attributes.g__tag_mc_type_attr,
117         -2147483647
118     )
119 }
120 \lua_now:e
121 {
122     tex.setattribute
123     (
124         "global",
125         luatexbase.attributes.g__tag_mc_cnt_attr,
126         -2147483647
127     )
128 }
129 }
130

```

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

__tag_mc_insert_mcid_kids:n These commands will in the finish code replace the dummy for a mc by the real mcid 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 <debug>          }
158   \lua_now:e
159   {
160     ltx.__tag.func.store_struct_mcabs
161     (
162       \g__tag_struct_stack_current_tl,#1
163     )
164   }
165 }
166 </luamode | debug>
167 <*luamode>
168 \cs_generate_variant:Nn \__tag_mc_handle_stash:n { e }
(End of definition for \__tag_mc_handle_stash:n.)

```

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

```

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

```

set the default tag to the structure:

```

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

```

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

```

185     \tl_if_empty:NTF \l__tag_mc_lang_tl
186     {
187       \keys_set:nn { __tag / mc } { label={}, #1 }
188     }
189     {
190       \keys_set:nn { __tag / mc } { label={}, lang=\l__tag_mc_lang_tl, #1 }
191     }
192     %check that a tag or artifact has been used
193     \__tag_check_mc_tag:N \l__tag_mc_key_tag_tl
194     %set the attributes:
195     \__tag_mc_lua_set_mc_type_attr:o { \l__tag_mc_key_tag_tl }
196     \bool_if:NF \l__tag_mc_artifact_bool
197     { % store the absolute num name in a label:
198       \tl_if_empty:NF { \l__tag_mc_key_label_tl }
199       {
200         \__tag_mc_handle_mc_label:e { \l__tag_mc_key_label_tl }
201       }
202       % if not stashed record the absolute number

```

```

203         \bool_if:NF \l__tag_mc_key_stash_bool
204         {
205             \socket_use:nn{tag/check/parent-child}
206             {
207                 \__tag_mc_check_parent_child:o
208                 { \g__tag_struct_stack_current_tl }
209             }
210             \__tag_mc_handle_stash:e { \__tag_get_mc_abs_cnt: }
211         }
212     }
213     \group_end:
214 }
215 }

```

(End of definition for \tag_mc_begin:n. This function is documented on page 81.)

\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 81.)

\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 81.)

__tag_get_data_mc_tag:

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

```

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

```

(End of definition for __tag_get_data_mc_tag:.)

1.2 Key definitions

tag (mc-key) TODO: check conversion, check if local/global setting is right.
 raw (mc-key) 238 \keys_define:nn { __tag / mc }
 alt (mc-key) 239 {
 lang (mc-key= 240 tag .code:n = %
 actualtext (mc-key) 241 {
 label (mc-key) 242 \tl_set:Nc \l__tag_mc_key_tag_tl { #1 }
 artifact (mc-key) 243 \tl_gset:Nc \g__tag_mc_key_tag_tl { #1 }
 244 \lua_now:e
 245 {
 246 ltx.__tag.func.store_mc_data(__tag_get_mc_abs_cnt:,"tag","#1")
 247 }
 248 },
 249 raw .code:n =
 250 {
 251 \tl_put_right:Nc \l__tag_mc_key_properties_tl { #1 }
 252 \lua_now:e
 253 {
 254 ltx.__tag.func.store_mc_data(__tag_get_mc_abs_cnt:,"raw","#1")
 255 }
 256 },
 257 alt .code:n = % Alt property
 258 {
 259 \tl_if_empty:oF{#1}
 260 {
 261 \str_set_convert:Noon
 262 \l__tag_tmpa_str
 263 { #1 }
 264 { default }
 265 { utf16/hex }
 266 \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
 267 \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
 268 \lua_now:e
 269 {
 270 ltx.__tag.func.store_mc_data
 271 (
 272 __tag_get_mc_abs_cnt:,"alt","/Alt~<\str_use:N \l__tag_tmpa_str>"
 273)
 274 }
 275 }
 276 },
 277 lang .code:n = % Lang property
 278 {
 279 \tl_if_empty:oF{#1}
 280 {
 281 \tl_put_right:Nc \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 }
 292 }
 293 }
 294 }
 295 }
 296 }
 297 }
 298 }
 299 }
 300 }
 301 }
 302 }
 303 }
 304 }
 305 }
 306 }
 307 }
 308 }
 309 }
 310 }
 311 }
 312 }
 313 }
 314 }
 315 }
 316 }
 317 }
 318 }
 319 }
 320 }
 321 }
 322 }
 323 }
 324 }
 325 }
 326 }
 327 }
 328 }
 329 }
 330 }
 331 }
 332 }
 333 }
 334 }
 335 }
 336 }
 337 }
 338 }
 339 }
 340 }
 341 }
 342 }
 343 }
 344 }
 345 }
 346 }
 347 }
 348 }
 349 }
 350 }
 351 }
 352 }
 353 }
 354 }
 355 }
 356 }
 357 }
 358 }
 359 }
 360 }
 361 }
 362 }
 363 }
 364 }
 365 }
 366 }
 367 }
 368 }
 369 }
 370 }
 371 }
 372 }
 373 }
 374 }
 375 }
 376 }
 377 }
 378 }
 379 }
 380 }
 381 }
 382 }
 383 }
 384 }
 385 }
 386 }
 387 }
 388 }
 389 }
 390 }
 391 }
 392 }
 393 }
 394 }
 395 }
 396 }
 397 }
 398 }
 399 }
 400 }
 401 }
 402 }
 403 }
 404 }
 405 }
 406 }
 407 }
 408 }
 409 }
 410 }
 411 }
 412 }
 413 }
 414 }
 415 }
 416 }
 417 }
 418 }
 419 }
 420 }
 421 }
 422 }
 423 }
 424 }
 425 }
 426 }
 427 }
 428 }
 429 }
 430 }
 431 }
 432 }
 433 }
 434 }
 435 }
 436 }
 437 }
 438 }
 439 }
 440 }
 441 }
 442 }
 443 }
 444 }
 445 }
 446 }
 447 }
 448 }
 449 }
 450 }
 451 }
 452 }
 453 }
 454 }
 455 }
 456 }
 457 }
 458 }
 459 }
 460 }
 461 }
 462 }
 463 }
 464 }
 465 }
 466 }
 467 }
 468 }
 469 }
 470 }
 471 }
 472 }
 473 }
 474 }
 475 }
 476 }
 477 }
 478 }
 479 }
 480 }
 481 }
 482 }
 483 }
 484 }
 485 }
 486 }
 487 }
 488 }
 489 }
 490 }
 491 }
 492 }
 493 }
 494 }
 495 }
 496 }
 497 }
 498 }
 499 }
 500 }
 501 }
 502 }
 503 }
 504 }
 505 }
 506 }
 507 }
 508 }
 509 }
 510 }
 511 }
 512 }
 513 }
 514 }
 515 }
 516 }
 517 }
 518 }
 519 }
 520 }
 521 }
 522 }
 523 }
 524 }
 525 }
 526 }
 527 }
 528 }
 529 }
 530 }
 531 }
 532 }
 533 }
 534 }
 535 }
 536 }
 537 }
 538 }
 539 }
 540 }
 541 }
 542 }
 543 }
 544 }
 545 }
 546 }
 547 }
 548 }
 549 }
 550 }
 551 }
 552 }
 553 }
 554 }
 555 }
 556 }
 557 }
 558 }
 559 }
 560 }
 561 }
 562 }
 563 }
 564 }
 565 }
 566 }
 567 }
 568 }
 569 }
 570 }
 571 }
 572 }
 573 }
 574 }
 575 }
 576 }
 577 }
 578 }
 579 }
 580 }
 581 }
 582 }
 583 }
 584 }
 585 }
 586 }
 587 }
 588 }
 589 }
 590 }
 591 }
 592 }
 593 }
 594 }
 595 }
 596 }
 597 }
 598 }
 599 }
 600 }
 601 }
 602 }
 603 }
 604 }
 605 }
 606 }
 607 }
 608 }
 609 }
 610 }
 611 }
 612 }
 613 }
 614 }
 615 }
 616 }
 617 }
 618 }
 619 }
 620 }
 621 }
 622 }
 623 }
 624 }
 625 }
 626 }
 627 }
 628 }
 629 }
 630 }
 631 }
 632 }
 633 }
 634 }
 635 }
 636 }
 637 }
 638 }
 639 }
 640 }
 641 }
 642 }
 643 }
 644 }
 645 }
 646 }
 647 }
 648 }
 649 }
 650 }
 651 }
 652 }
 653 }
 654 }
 655 }
 656 }
 657 }
 658 }
 659 }
 660 }
 661 }
 662 }
 663 }
 664 }
 665 }
 666 }
 667 }
 668 }
 669 }
 670 }
 671 }
 672 }
 673 }
 674 }
 675 }
 676 }
 677 }
 678 }
 679 }
 680 }
 681 }
 682 }
 683 }
 684 }
 685 }
 686 }
 687 }
 688 }
 689 }
 690 }
 691 }
 692 }
 693 }
 694 }
 695 }
 696 }
 697 }
 698 }
 699 }
 700 }
 701 }
 702 }
 703 }
 704 }
 705 }
 706 }
 707 }
 708 }
 709 }
 710 }
 711 }
 712 }
 713 }
 714 }
 715 }
 716 }
 717 }
 718 }
 719 }
 720 }
 721 }
 722 }
 723 }
 724 }
 725 }
 726 }
 727 }
 728 }
 729 }
 730 }
 731 }
 732 }
 733 }
 734 }
 735 }
 736 }
 737 }
 738 }
 739 }
 740 }
 741 }
 742 }
 743 }
 744 }
 745 }
 746 }
 747 }
 748 }
 749 }
 750 }
 751 }
 752 }
 753 }
 754 }
 755 }
 756 }
 757 }
 758 }
 759 }
 760 }
 761 }
 762 }
 763 }
 764 }
 765 }
 766 }
 767 }
 768 }
 769 }
 770 }
 771 }
 772 }
 773 }
 774 }
 775 }
 776 }
 777 }
 778 }
 779 }
 780 }
 781 }
 782 }
 783 }
 784 }
 785 }
 786 }
 787 }
 788 }
 789 }
 790 }
 791 }
 792 }
 793 }
 794 }
 795 }
 796 }
 797 }
 798 }
 799 }
 800 }
 801 }
 802 }
 803 }
 804 }
 805 }
 806 }
 807 }
 808 }
 809 }
 810 }
 811 }
 812 }
 813 }
 814 }
 815 }
 816 }
 817 }
 818 }
 819 }
 820 }
 821 }
 822 }
 823 }
 824 }
 825 }
 826 }
 827 }
 828 }
 829 }
 830 }
 831 }
 832 }
 833 }
 834 }
 835 }
 836 }
 837 }
 838 }
 839 }
 840 }
 841 }
 842 }
 843 }
 844 }
 845 }
 846 }
 847 }
 848 }
 849 }
 850 }
 851 }
 852 }
 853 }
 854 }
 855 }
 856 }
 857 }
 858 }
 859 }
 860 }
 861 }
 862 }
 863 }
 864 }
 865 }
 866 }
 867 }
 868 }
 869 }
 870 }
 871 }
 872 }
 873 }
 874 }
 875 }
 876 }
 877 }
 878 }
 879 }
 880 }
 881 }
 882 }
 883 }
 884 }
 885 }
 886 }
 887 }
 888 }
 889 }
 890 }
 891 }
 892 }
 893 }
 894 }
 895 }
 896 }
 897 }
 898 }
 899 }
 900 }
 901 }
 902 }
 903 }
 904 }
 905 }
 906 }
 907 }
 908 }
 909 }
 910 }
 911 }
 912 }
 913 }
 914 }
 915 }
 916 }
 917 }
 918 }
 919 }
 920 }
 921 }
 922 }
 923 }
 924 }
 925 }
 926 }
 927 }
 928 }
 929 }
 930 }
 931 }
 932 }
 933 }
 934 }
 935 }
 936 }
 937 }
 938 }
 939 }
 940 }
 941 }
 942 }
 943 }
 944 }
 945 }
 946 }
 947 }
 948 }
 949 }
 950 }
 951 }
 952 }
 953 }
 954 }
 955 }
 956 }
 957 }
 958 }
 959 }
 960 }
 961 }
 962 }
 963 }
 964 }
 965 }
 966 }
 967 }
 968 }
 969 }
 970 }
 971 }
 972 }
 973 }
 974 }
 975 }
 976 }
 977 }
 978 }
 979 }
 980 }
 981 }
 982 }
 983 }
 984 }
 985 }
 986 }
 987 }
 988 }
 989 }
 990 }
 991 }
 992 }
 993 }
 994 }
 995 }
 996 }
 997 }
 998 }
 999 }
 1000 }
 1001 }
 1002 }
 1003 }
 1004 }
 1005 }
 1006 }
 1007 }
 1008 }
 1009 }
 1010 }
 1011 }
 1012 }
 1013 }
 1014 }
 1015 }
 1016 }
 1017 }
 1018 }
 1019 }
 1020 }
 1021 }
 1022 }
 1023 }
 1024 }
 1025 }
 1026 }
 1027 }
 1028 }
 1029 }
 1030 }
 1031 }
 1032 }
 1033 }
 1034 }
 1035 }
 1036 }
 1037 }
 1038 }
 1039 }
 1040 }
 1041 }
 1042 }
 1043 }
 1044 }
 1045 }
 1046 }
 1047 }
 1048 }
 1049 }
 1050 }
 1051 }
 1052 }
 1053 }
 1054 }
 1055 }
 1056 }
 1057 }
 1058 }
 1059 }
 1060 }
 1061 }
 1062 }
 1063 }
 1064 }
 1065 }
 1066 }
 1067 }
 1068 }
 1069 }
 1070 }
 1071 }
 1072 }
 1073 }
 1074 }
 1075 }
 1076 }
 1077 }
 1078 }
 1079 }
 1080 }
 1081 }
 1082 }
 1083 }
 1084 }
 1085 }
 1086 }
 1087 }
 1088 }
 1089 }
 1090 }
 1091 }
 1092 }
 1093 }
 1094 }
 1095 }
 1096 }
 1097 }
 1098 }
 1099 }
 1100 }
 1101 }
 1102 }
 1103 }
 1104 }
 1105 }
 1106 }
 1107 }
 1108 }
 1109 }
 1110 }
 1111 }
 1112 }
 1113 }
 1114 }
 1115 }
 1116 }
 1117 }
 1118 }
 1119 }
 1120 }
 1121 }
 1122 }
 1123 }
 1124 }
 1125 }
 1126 }
 1127 }
 1128 }
 1129 }
 1130 }
 1131 }
 1132 }
 1133 }
 1134 }
 1135 }
 1136 }
 1137 }
 1138 }
 1139 }
 1140 }
 1141 }
 1142 }
 1143 }
 1144 }
 1145 }
 1146 }
 1147 }
 1148 }
 1149 }
 1150 }
 1151 }
 1152 }
 1153 }
 1154 }
 1155 }
 1156 }
 1157 }
 1158 }
 1159 }
 1160 }
 1161 }
 1162 }
 1163 }
 1164 }
 1165 }
 1166 }
 1167 }
 1168 }
 1169 }
 1170 }
 1171 }
 1172 }
 1173 }
 1174 }
 1175 }
 1176 }
 1177 }
 1178 }
 1179 }
 1180 }
 1181 }
 1182 }
 1183 }
 1184 }
 1185 }
 1186 }
 1187 }
 1188 }
 1189 }
 1190 }
 1191 }
 1192 }
 1193 }
 1194 }
 1195 }
 1196 }
 1197 }
 1198 }
 1199 }
 1200 }
 1201 }
 1202 }
 1203 }
 1204 }
 1205 }
 1206 }
 1207 }
 1208 }
 1209 }
 1210 }
 1211 }
 1212 }
 1213 }
 1214 }
 1215 }
 1216 }
 1217 }
 1218 }
 1219 }
 1220 }
 1221 }
 1222 }
 1223 }
 1224 }
 1225 }
 1226 }
 1227 }
 1228 }
 1229 }
 1230 }
 1231 }
 1232 }
 1233 }
 1234 }
 1235 }
 1236 }
 1237 }
 1238 }
 1239 }
 1240 }
 1241 }
 1242 }
 1243 }
 1244 }
 1245 }
 1246 }
 1247 }
 1248 }
 1249 }
 1250 }
 1251 }
 1252 }
 1253 }
 1254 }
 1255 }
 1256 }
 1257 }
 1258 }
 1259 }
 1260 }
 1261 }
 1262 }
 1263 }
 1264 }
 1265 }
 1266 }
 1267 }
 1268 }
 1269 }
 1270 }
 1271 }
 1272 }
 1273 }
 1274 }
 1275 }
 1276 }
 1277 }
 1278 }
 1279 }
 1280 }
 1281 }
 1282 }
 1283 }
 1284 }
 1285 }
 1286 }
 1287 }
 1288 }
 1289 }
 1290 }
 1291 }
 1292 }
 1293 }
 1294 }
 1295 }
 1296 }
 1297 }
 1298 }
 1299 }
 1300 }
 1301 }
 1302 }
 1303 }
 1304 }
 1305 }
 1306 }
 1307 }
 1308 }
 1309 }
 1310 }
 1311 }
 1312 }
 1313 }
 1314 }
 1315 }
 1316 }
 1317 }
 1318 }
 1319 }
 1320 }
 1321 }
 1322 }
 1323 }
 1324 }
 1325 }
 1326 }
 1327 }
 1328 }
 1329 }
 1330 }
 1331 }
 1332 }
 1333 }
 1334 }
 1335 }
 1336 }
 1337 }
 1338 }
 1339 }
 1340 }
 1341 }
 1342 }
 1343 }
 1344 }
 1345 }
 1346 }
 1347 }
 1348 }
 1349 }
 1350 }
 1351 }
 1352 }
 1353 }
 1354 }
 1355 }
 1356 }
 1357 }
 1358 }
 1359 }
 1360 }
 1361 }
 1362 }
 1363 }
 1364 }
 1365 }
 1366 }
 1367 }
 1368 }
 1369 }
 1370 }
 1371 }
 1372 }
 1373 }
 1374 }
 1375 }
 1376 }
 1377 }
 1378 }
 1379 }
 1380 }
 1381 }
 1382 }
 1383 }
 1384 }
 1385 }
 1386 }
 1387 }
 1388 }
 1389 }
 1390 }
 1391 }
 1392 }
 1393 }
 1

```

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 82.)

The tagpdf-struct module
 Commands to create the structure
 Part of the tagpdf package
 Ulrike Fischer
 Version 0.99y, released 2026-01-29

Part VIII

1 Public Commands

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

These commands start and end a new structure. They don't start a group. They set all their values globally. `\tag_struct_end:n` does nothing special normally (apart from swallowing its argument, but if `tagpdf-debug` is loaded, it will check if the `{<tag>}` (after expansion) is identical to the current structure on the stack. The tag is not role mapped!

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

These commands insert a structure previously stashed away as kid into the currently active structure. A structure should be used only once, if the structure already has a parent a warning is issued.

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

This is a small wrapper around `\pdf_object_ref:n` to retrieve the object reference of the structure with the number `<struct number>`. This number can be retrieved and stored for the current structure for example with `\tag_get:n{<structnum>}`. Be aware that it can only be used if the structure has already been created and that it doesn't check if the object actually exists!

The following two functions are used to add annotations. They must be used together and with care to get the same numbers. Perhaps some improvements are needed here.

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

This inserts an annotation in the structure. `<object reference>` is there reference to the annotation. `<struct parent number>` should be the same number as had been inserted with `\tag_struct_parent_int:` as `StructParent` value to the dictionary of the annotation. The command will increase the value of the counter used by `\tag_struct_parent_int:`.

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

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

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

This is a command that allows to update the data of a structure. This often can't done simply by replacing the value, as we have to preserve and extend existing content. We use therefore dedicated functions adjusted to the key in question. The first argument is the number of the structure, the second a keyword referring to a function, the third the value. Currently the only keyword is `ref` which updates the `Ref` key (an array)

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

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

2 Public keys

2.1 Keys for the structure commands

tag (*struct key*) This is required. The value of the key is normally one of the standard types listed in the main tagpdf documentation. It is possible to setup new tags/types. The value can also be of the form `type/NS`, where NS is the shorthand of a declared name space. Currently the names spaces `pdf`, `pdf2`, `mathml` and `user` are defined. This allows to use a different name space than the one connected by default to the tag. But normally this should not be needed.

stash (*struct key*) Normally a new structure inserts itself as a kid into the currently active structure. This key prohibits this. The structure is nevertheless from now on “the current active structure” and parent for following marked content and structures.

label (*struct key*) This key sets a label by which one can refer to the structure. It is e.g. used by `\tag_struct_use:n` (where a real label is actually not needed as you can only use structures already defined), and by the `ref` key (which can refer to future structures). Internally the label name will start with `tagpdfstruct-` and it stores the two attributes `tagstruct` (the structure number) and `tagstructobj` (the object reference).

parent (*struct key*) By default a structure is added as kid to the currently active structure. With the parent key one can choose another parent. The value is a structure number which must refer to an already existing, previously created structure. Such a structure number can for example be have been stored with `\tag_get:n`, but one can also use a label on the parent structure and then use `\property_ref:nn{tagpdfstruct-label}{tagstruct}` to retrieve it.

firstkid (*struct key*) If this key is used the structure is added at the left of the kids of the parent structure (if the structure is not stashed). This means that it will be the first kid of the structure (unless some later structure uses the key too).

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

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

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

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

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

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

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

AF (*struct key*) These keys handle associated files in the structure element.

AFref (*struct key*)

AFinline (*struct key*) **AF** = `<object name>`

AFinline-o (*struct key*) **AFref** = `<object reference>`

texsource (*struct key*) **AF-inline** = `<text content>`

mathml (*struct key*)

The value `<object name>` should be the name of an object pointing to the `/Filespec` dictionary as expected by `\pdf_object_ref:n` from a current `l3kernel`.

The value **AF-inline** is some text, which is embedded in the PDF as a text file with mime type `text/plain`. **AF-inline-o** is like **AF-inline** but expands the value once.

Future versions will perhaps extend this to more mime types, but it is still a research task to find out what is really needed.

texsource is a special variant of **AF-inline-o** which embeds the content as `.tex` source with the `/AFrelationship` key set to `/Source`. It also sets the `/Desc` key to a (currently) fix text.

mathml is a special variant of **AF-inline-o** which embeds the content as `.xml` file with the `/AFrelationship` key set to `/Supplement`. It also sets the `/Desc` key to a (currently) fix text.

The argument of **AF** is an object name referring an embedded file as declared for example with `\pdf_object_new:n` or with the `l3pdffile` module. **AF** expands its argument (this allows e.g. to use some variable for automatic numbering) and can be used more than once, to associate more than one file.

The argument of **AFref** is an object reference to an embedded file or a variable expanding to such a object reference in the format as you would get e.g. from `\pdf_object_ref_last:` or `\pdf_object_ref:n` (and which is different for the various engines!). The key allows to make use of anonymous objects. Like **AF** the **AFref** key expands its argument and can be used more than once, to associate more than one file. *It does not check if the reference is valid!*

The inline keys can be used only once per structure. Additional calls are ignored.

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

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

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

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

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

2.2 Setup keys

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

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

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

`root-AF` (setup key)

`root-AF = <object name>`

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

```
1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-struct-code} {2026-01-29} {0.99y}
4 {part of tagpdf - code related to storing structure}
5 </header>
```

3 Variables

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

```
6 <base>\int_new:N \c@g__tag_struct_abs_int
7 <base>\int_gset:Nn \c@g__tag_struct_abs_int { 1 }
(End of definition for \c@g__tag_struct_abs_int.)
```

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

```
8 <*package>
9 \__tag_seq_new:N \g__tag_struct_objR_seq
(End of definition for \g__tag_struct_objR_seq.)
```

`\c__tag_struct_null_tl` In lua mode we have to test if the kids a null

```
10 \tl_const:Nn\c__tag_struct_null_tl {null}
(End of definition for \c__tag_struct_null_tl.)
```

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

```

11 \__tag_prop_new:N \g__tag_struct_cont_mc_prop
(End of definition for \g__tag_struct_cont_mc_prop.)

```

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

```

12 \seq_new:N \g__tag_struct_stack_seq
13 \seq_gpush:Nn \g__tag_struct_stack_seq {1}
(End of definition for \g__tag_struct_stack_seq.)

```

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

```

14 \seq_new:N \g__tag_struct_tag_stack_seq
15 \seq_gpush:Nn \g__tag_struct_tag_stack_seq {{Root}}{StructTreeRoot}}
(End of definition for \g__tag_struct_tag_stack_seq.)

```

`\g__tag_struct_stack_current_tl`
`\l__tag_struct_stack_parent_tmpa_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.

```

16 \</package>
17 \<base>\tl_new:N \g__tag_struct_stack_current_tl
18 \<base>\tl_gset:Nn \g__tag_struct_stack_current_tl {\int_use:N\c@g__tag_struct_abs_int}
19 \<*package>
20 \tl_new:N \l__tag_struct_stack_parent_tmpa_tl
(End of definition for \g__tag_struct_stack_current_tl and \l__tag_struct_stack_parent_tmpa_tl.)

```

In luatex we will store the structure number as attribute.

```

21 \sys_if_engine_luatex:TF
22 {
23   \cs_new:Npn \__tag_struct_set_attribute:
24   {
25     \lua_now:e
26     {
27       tex.setattribute
28       (
29         "global",
30         luatexbase.attributes.g__tag_structnum_attr,
31         \g__tag_struct_stack_current_tl
32       )
33     }
34   }
35 }
36 {
37   \cs_new_eq:NN \__tag_struct_set_attribute: \prg_do_nothing:
38 }

```

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

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

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

All properties have at least the keys

Type StructTreeRoot or StructElem

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

```

39 \seq_const_from_clist:Nn \c__tag_struct_StructTreeRoot_entries_seq
40   {%p. 857/858
41     Type,                % always /StructTreeRoot
42     K,                   % kid, dictionary or array of dictionaries
43     IDTree,              % currently unused
44     ParentTree,          % required,obj ref to the parent tree
45     ParentTreeNextKey,   % optional
46     RoleMap,
47     ClassMap,
48     Namespaces,
49     AF                   %pdf 2.0
50   }
51
52 \seq_const_from_clist:Nn \c__tag_struct_StructElem_entries_seq
53   {%p 858 f
54     Type,                %always /StructElem
55     S,                   %tag/type
56     P,                   %parent
57     ID,                  %optional
58     Ref,                 %optional, pdf 2.0 Use?
59     Pg,                  %obj num of starting page, optional
60     K,                   %kids
61     A,                   %attributes, probably unused
62     C,                   %class ""
63     %R,                  %attribute revision number, irrelevant for us as we
64                          % don't update/change existing PDF and (probably)
65                          % deprecated in PDF 2.0
66     T,                   %title, value in () or <>
67     Lang,                %language
68     Alt,                 % value in () or <>
69     E,                   % abbreviation
70     ActualText,
71     AF,                  %pdf 2.0, array of dict, associated files
72     NS,                  %pdf 2.0, dict, namespace
73     PhoneticAlphabet,    %pdf 2.0
74     Phoneme              %pdf 2.0
75   }

```

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

3.1 Variables used by the keys

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.

```

76 \tl_new:N \g__tag_struct_tag_tl
77 \tl_new:N \g__tag_struct_tag_NS_tl
78 \tl_new:N \l__tag_struct_roletag_tl
79 \tl_new:N \l__tag_struct_roletag_NS_tl
80 \tl_new:N \l__tag_struct_parenttag_tl
81 \tl_set:Nn \l__tag_struct_parenttag_tl {STASHED}
82 \tl_new:N \l__tag_struct_parenttag_NS_tl
83 \tl_set:Nn \l__tag_struct_parenttag_NS_tl {latex}
(End of definition for \g__tag_struct_tag_tl and others.)

```

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.

```

84 \prop_new_linked:N \g__tag_struct_label_num_prop
(End of definition for \g__tag_struct_label_num_prop.)

```

This will keep track of the stash status

```

85 \bool_new:N \l__tag_struct_elem_stash_bool
(End of definition for \l__tag_struct_elem_stash_bool.)

```

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

```

86 \tl_new:N \l__tag_struct_addkid_tl
87 \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

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

```

88 \</package>
89 \<base>\prop_new_linked:N \g__tag_struct_dest_num_prop
90 \<*package>
(End of definition for \g__tag_struct_dest_num_prop.)

```

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

```

91 \prop_new_linked:N \g__tag_struct_ref_by_dest_prop
92 \</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:

```

93 <*package| debug>
94 <package>\\cs_new_protected:Npn \\_tag_struct_prop_gput:nnn #1 #2 #3
95 <debug>\\cs_set_protected:Npn \\_tag_struct_prop_gput:nnn #1 #2 #3
96 {
97   \\_tag_prop_gput:cnn
98   { g__tag_struct_#1_prop }{#2}{#3}
99 <debug>\\prop_gput:cnn { g__tag_struct_debug_#1_prop } {#2} {#3}
100 }
101 \\cs_generate_variant:Nn \\_tag_struct_prop_gput:nnn {onn,nne,nee,nnn}
102 </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.

```

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

\\_tag_pdf_name_e:n
105 \\cs_new:Npn \\_tag_pdf_name_e:n #1{\\pdf_name_from_unicode_e:n{#1}}
106 </package>
(End of definition for \\_tag_pdf_name_e:n.)

g__tag_struct_1_prop
g__tag_struct_kids_1_seq
107 <*package>
108 \\_tag_prop_new:c { g__tag_struct_1_prop }
109 \\_tag_seq_new:c { g__tag_struct_kids_1_seq }
110
111 \\_tag_struct_prop_gput:nne
112 { 1 }
113 { Type }
114 { \\pdf_name_from_unicode_e:n {StructTreeRoot} }
115
116 \\_tag_struct_prop_gput:nne
117 { 1 }
118 { S }
119 { \\pdf_name_from_unicode_e:n {StructTreeRoot} }
120
121 \\_tag_struct_prop_gput:nne
122 { 1 }
123 { tag }
124 { {StructTreeRoot}{pdf} }
125
126 \\_tag_struct_prop_gput:nne

```

```

127 { 1 }
128 { rolemap }
129 { {StructTreeRoot}{pdf} }
130
131 \__tag_struct_prop_gput:nne
132 { 1 }
133 { parentrole }
134 { {StructTreeRoot}{pdf} }
135

```

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

```

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

```

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

```

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

```

(End of definition for g__tag_struct_1_prop and g__tag_struct_kids_1_seq.)

4.2 Adding the /ID key

Every structure gets automatically an ID which is currently simply calculated from the structure number.

__tag_struct_get_id:n

```

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

```

(End of definition for __tag_struct_get_id:n.)

4.3 Filling in the tag info

__tag_struct_set_tag_info:nnn

This adds or updates the tag info to a structure given by a number. We need also the original data, so we store both.

```

159 \pdf_version_compare:NnTF < {2.0}
160 {
161   \cs_new_protected:Npn \__tag_struct_set_tag_info:nnn #1 #2 #3

```



```

162     %#1 structure number, #2 tag, #3 NS
163     {
164         \__tag_struct_prop_gput:nne
165         { #1 }
166         { S }
167         { \pdf_name_from_unicode_e:n {#2} } %
168         \__tag_struct_prop_gput:nnn
169         { #1 }
170         { tag }
171         { {#2} {} }
172     }
173 }
174 {
175     \cs_new_protected:Npn \__tag_struct_set_tag_info:nnn #1 #2 #3
176     {
177         \__tag_struct_prop_gput:nne
178         { #1 }
179         { S }
180         { \pdf_name_from_unicode_e:n {#2} } %
181         \prop_get:NnNT \g__tag_role_NS_prop {#3} \l__tag_get_tmpc_tl
182         {
183             \__tag_struct_prop_gput:nne
184             { #1 }
185             { NS }
186             { \l__tag_get_tmpc_tl } %
187         }
188         \__tag_struct_prop_gput:nnn
189         { #1 }
190         { tag }
191         { {#2} {#3} }
192     }
193 }
194 \cs_generate_variant:Nn \__tag_struct_set_tag_info:nnn {eoo}
(End of definition for \__tag_struct_set_tag_info:nnn.)

```

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

```

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

```

```

211         \tl_clear:N#4
212     }
213 }
214 \cs_generate_variant:Nn\__tag_struct_get_role:nnNN {enNN}
(End of definition for \__tag_struct_get_role:nnNN.)

```

4.4 Handlings kids

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

```

\__tag_struct_kid_mc_gput_right:nn
\__tag_struct_kid_mc_gput_right:ne

```

The command to store an mc-chunk, this is a dictionary of type MCR. It would be possible to write out the content directly as unnamed object and to store only the object reference, but probably this would be slower, and the PDF is more readable like this. The code doesn't try to avoid the use of the /Pg key by checking page numbers. That imho only slows down without much gain. In generic mode the page break code will perhaps have to insert an additional mcid after an existing one. For this we use a property list At first an auxiliary to write the MCID dict. This should normally be expanded!

```

215 \cs_new:Npn \__tag_struct_mcid_dict:n #1 %#1 MCID absnum
216 {
217     <<
218     /Type \c_space_tl /MCR \c_space_tl
219     /Pg
220     \c_space_tl
221     \pdf_pageobject_ref:n { \property_ref:enn{mcid-#1}{tagabspage}{1} }
222     /MCID \c_space_tl \property_ref:enn{mcid-#1}{tagmcid}{1}
223     >>
224 }
225 (/package)
226 (*package|debug)
227 (package)\cs_new_protected:Npn \__tag_struct_kid_mc_gput_right:nn #1 #2
228 (debug)\cs_set_protected:Npn \__tag_struct_kid_mc_gput_right:nn #1 #2
229 %#1 structure num, #2 MCID absnum%
230 {
231     \__tag_seq_gput_right:ce
232     { g__tag_struct_kids_#1_seq }
233     {
234         \__tag_struct_mcid_dict:n {#2}
235     }
236 (debug) \seq_gput_right:cn
237 (debug) { g__tag_struct_debug_kids_#1_seq }
238 (debug) {
239 (debug) MC~#2
240 (debug) }
241 \__tag_seq_gput_right:cn
242 { g__tag_struct_kids_#1_seq }
243 {
244     \prop_item:Nn \g__tag_struct_cont_mc_prop {#2}
245 }
246 }
247 (package)\cs_generate_variant:Nn \__tag_struct_kid_mc_gput_right:nn {ne}

```

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

`_tag_struct_kid_struct_gput_right:nn` This commands adds a structure as kid. We only need to record the object reference in the sequence.
`_tag_struct_kid_struct_gput_right:ee`

```

248 <package>\cs_new_protected:Npn\_tag_struct_kid_struct_gput_right:nn #1 #2
249 <debug>\cs_set_protected:Npn\_tag_struct_kid_struct_gput_right:nn #1 #2
250 %%#1 num of parent struct, #2 kid struct
251 {
252   \_tag_seq_gput_right:ce
253   { g\_tag_struct_kids_#1_seq }
254   {
255     \pdf_object_ref_indexed:nn { \_tag/struct }{ #2 }
256   }
257 <debug>   \seq_gput_right:cn
258 <debug>   { g\_tag_struct_debug_kids_#1_seq }
259 <debug>   {
260     Struct~#2
261 <debug>   }
262 }
263 <package>\cs_generate_variant:Nn \_tag_struct_kid_struct_gput_right:nn {ee}
(End of definition for \_tag_struct_kid_struct_gput_right:nn.)

```

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

```

264 <package>\cs_new_protected:Npn\_tag_struct_kid_struct_gput_left:nn #1 #2
265 <debug>\cs_set_protected:Npn\_tag_struct_kid_struct_gput_left:nn #1 #2
266 %%#1 num of parent struct, #2 kid struct
267 {
268   \_tag_seq_gput_left:ce
269   { g\_tag_struct_kids_#1_seq }
270   {
271     \pdf_object_ref_indexed:nn { \_tag/struct }{ #2 }
272   }
273 <debug>   \seq_gput_left:cn
274 <debug>   { g\_tag_struct_debug_kids_#1_seq }
275 <debug>   {
276     Struct~#2
277 <debug>   }
278 }
279 <package>\cs_generate_variant:Nn \_tag_struct_kid_struct_gput_left:nn {ee}
(End of definition for \_tag_struct_kid_struct_gput_left:nn.)

```

`_tag_struct_kid_OBJR_gput_right:nnn` 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
`_tag_struct_kid_OBJR_gput_right:eee`

```

280 <package>\cs_new_protected:Npn\_tag_struct_kid_OBJR_gput_right:nnn #1 #2 #3
281 <package>
282 <package>
283 <debug>\cs_set_protected:Npn\_tag_struct_kid_OBJR_gput_right:nnn #1 #2 #3
284 %%#1 num of parent struct,#2 obj reference,#3 page object reference
285 {
286   \pdf_object_unnamed_write:nn
287   { dict }

```

```

288     {
289       /Type/ObjR/Obj~#2/Pg~#3
290     }
291     \__tag_seq_gput_right:ce
292     { g__tag_struct_kids_#1_seq }
293     {
294       \pdf_object_ref_last:
295     }
296     <debug> \seq_gput_right:ce
297     <debug> { g__tag_struct_debug_kids_#1_seq }
298     <debug> {
299       <debug> OBJR~reference
300     <debug> }
301   }
302 </package | debug>
303 <*package>
304 \cs_generate_variant:Nn\__tag_struct_kid_OBJR_gput_right:nnn { eee }
(End of definition for \__tag_struct_kid_OBJR_gput_right:nnn.)

```

__tag_struct_exchange_kid_command:N
__tag_struct_exchange_kid_command:c

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

```

305 \cs_new_protected:Npn\__tag_struct_exchange_kid_command:N #1 %#1 = seq var
306 {
307   \seq_gpop_left:NN #1 \l__tag_tmpa_tl
308   \tl_replace_once:Nnn \l__tag_tmpa_tl
309   {\__tag_mc_insert_mcid_kids:n}
310   {\__tag_mc_insert_mcid_single_kids:n}
311   \seq_gput_left:No #1 { \l__tag_tmpa_tl }
312 }
313
314 \cs_generate_variant:Nn\__tag_struct_exchange_kid_command:N { c }
(End of definition for \__tag_struct_exchange_kid_command:N.)

```

__tag_struct_fill_kid_key:n

This command adds the kid info to the K entry. In lua mode the content contains commands which are expanded later. The argument is the structure number.

```

315 \cs_new_protected:Npn \__tag_struct_fill_kid_key:n #1 %#1 is the struct num
316 {
317   \bool_if:NF \g__tag_mode_lua_bool
318   {
319     \seq_clear:N \l__tag_tmpa_seq
320     \seq_map_inline:cn { g__tag_struct_kids_#1_seq }
321     { \seq_put_right:Ne \l__tag_tmpa_seq { ##1 } }
322     %\seq_show:c { g__tag_struct_kids_#1_seq }
323     %\seq_show:N \l__tag_tmpa_seq
324     \seq_remove_all:Nn \l__tag_tmpa_seq {}
325     %\seq_show:N \l__tag_tmpa_seq
326     \seq_gset_eq:cN { g__tag_struct_kids_#1_seq } \l__tag_tmpa_seq
327   }
328
329   \int_case:nnF
330   {

```

```

331     \seq_count:c
332     {
333         g__tag_struct_kids_#1_seq
334     }
335 }
336 {
337     { 0 }
338     { } %no kids, do nothing
339     { 1 } % 1 kid, insert
340     {
341         % in this case we need a special command in
342         % luamode to get the array right. See issue #13
343         \sys_if_engine luatex:TF
344         {
345             \__tag_struct_exchange_kid_command:c
346             {g__tag_struct_kids_#1_seq}

```

check if we get null

```

347         \tl_set:N\l__tag_tmpa_tl
348         {\use:e{\seq_item:cn {g__tag_struct_kids_#1_seq} {1}}}
349         \tl_if_eq:NNF\l__tag_tmpa_tl \c__tag_struct_null_tl
350         {
351             \__tag_struct_prop_gput:nne
352             {#1}
353             {K}
354             {
355                 \seq_item:cn
356                 {
357                     g__tag_struct_kids_#1_seq
358                 }
359                 {1}
360             }
361         }
362     }
363     {
364         \__tag_struct_prop_gput:nne
365         {#1}
366         {K}
367         {
368             \seq_item:cn
369             {
370                 g__tag_struct_kids_#1_seq
371             }
372             {1}
373         }
374     }
375 } %
376 }
377 { %many kids, use an array
378     \__tag_struct_prop_gput:nne
379     {#1}
380     {K}
381     {
382         [

```

```

383         \seq_use:cn
384         {
385             g__tag_struct_kids_#1_seq
386         }
387         {
388             \c_space_tl
389         }
390     ]
391 }
392 }
393 }
394
(End of definition for \__tag_struct_fill_kid_key:n.)

```

4.5 Output of the object

```

\__tag_struct_get_dict_content:nN
This maps the dictionary content of a structure into a tl-var. Basically it does what
\pdfdict_use:n does. This is used a lot so should be rather fast.
395 \cs_new_protected:Npn \__tag_struct_get_dict_content:nN #1 #2 %#1: structure num
396 {
397     \tl_clear:N #2
398     \prop_map_inline:cn { g__tag_struct_#1_prop }
399     {

```

Some keys needs the option to format the value, e.g. add brackets for an array, we also need the option to ignore some entries in the properties.

```

400         \cs_if_exist_use:cTF {__tag_struct_format_##1:nnN}
401         {
402             {##1}{##2}#2
403         }
404         {
405             \tl_put_right:Ne #2 { \c_space_tl/##1~##2 }
406         }
407     }
408 }
(End of definition for \__tag_struct_get_dict_content:nN.)

```

```

\__tag_struct_format_rolemap:nnN
\__tag_struct_format_parentrole:nnN
\__tag_struct_format_P:nnN
\__tag_struct_format_tag:nnN
This three entries should not end in the PDF. Todo: check if the S/NS keys can be
dropped and replaced by a processing of the tag key.
409 \cs_new:Nn\__tag_struct_format_rolemap:nnN{}
410 \cs_new:Nn\__tag_struct_format_parentrole:nnN{}
411 \cs_new:Nn\__tag_struct_format_tag:nnN{}
(End of definition for \__tag_struct_format_rolemap:nnN and others.)

```

```

\__tag_struct_format_parentnum:nnN
parent is a structure number and should expand to the object reference.
412 \cs_new_protected:Nn\__tag_struct_format_parentnum:nnN
413 {
414     \tl_put_right:Ne #3 { ~/P~\pdf_object_ref_indexed:nn { __tag/struct} { #2 } }
415 }
(End of definition for \__tag_struct_format_parentnum:nnN.)

```

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

```

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

```

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

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

```

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

```

It can happen that a structure is not used and so has not parent. Simply ignoring it is problematic as it is also recorded in the IDTree, so we make an artifact out of it.

```

432     \prop_get:cnNF { g__tag_struct_#1_prop } {parentnum}\l__tag_tmpb_tl
433     {
434       %       \prop_gput:cne { g__tag_struct_#1_prop } {P}
435       %       {\pdf_object_ref_indexed:nn { __tag/struct }{1}}
436       \prop_gput:cne { g__tag_struct_#1_prop } {parentnum}{1}
437       \prop_gput:cne { g__tag_struct_#1_prop } {S}{/Artifact}
438       \seq_if_empty:cF {g__tag_struct_kids_#1_seq}
439       {
440         \msg_warning:nnee
441         {tag}
442         {struct-orphan}
443         { #1 }
444         {\seq_count:c{g__tag_struct_kids_#1_seq}}
445       }
446     }
447     \__tag_struct_fill_kid_key:n { #1 }
448     \__tag_struct_get_dict_content:nN { #1 } \l__tag_tmpa_tl
449     \pdf_object_write_indexed:nnne
450     { __tag/struct }{ #1 }
451     {dict}
452     {
453       \l__tag_tmpa_tl\c_space_tl
454       /ID~\__tag_struct_get_id:n{#1}
455     }
456   }
457   {
458     \msg_error:nnn { tag } { struct-no-objnum } { #1}
459

```

```

460     }
461 }
(End of definition for \_tag_struct_write_obj:n.)

```

_tag_struct_insert_annot:nn This is the command to insert an annotation into the structure. It can probably be used for xform too.
 Annotations used as structure content must

1. add a StructParent integer to their dictionary
2. push the object reference as OBJR object in the structure
3. Add a Structparent/obj-nr reference to the parent tree.

For a link this looks like this

```

(1) \tag_struct_begin:n { tag=Link }
    \tag_mc_begin:n { tag=Link }
    \pdfannot_dict_put:nne
      { link/URI }
      { StructParent }
      { \int_use:N\c@g_@@_parenttree_obj_int }
    <start link> link text <stop link>
(2+3) \@@_struct_insert_annot:nn {obj ref}{parent num}
      \tag_mc_end:
      \tag_struct_end:

462 \cs_new_protected:Npn \_tag_struct_insert_annot:nn #1 #2
463   %#1 object reference to the annotation/xform
464   %#2 structparent number
465   {
466     \bool_if:NT \g__tag_active_struct_bool
467     {
468       %get the number of the parent structure:
469       \seq_get:NNF
470         \g__tag_struct_stack_seq
471         \l__tag_struct_stack_parent_tmpa_tl
472         {
473           \msg_error:nn { tag } { struct-faulty-nesting }
474         }
475       %put the obj number of the annot in the kid entry, this also creates
476       %the OBJR object
477       \_tag_property_record:nn {@tag@objr@page@#2 }{ tagabspage }
478       \_tag_struct_kid_OBJR_gput_right:eee
479       {
480         \l__tag_struct_stack_parent_tmpa_tl
481       }
482       {
483         #1 %
484       }
485       {
486         \pdf_pageobject_ref:n
487         { \property_ref:nnn {@tag@objr@page@#2 }{ tagabspage }{1} }
488       }

```



```

489      % add the parent obj number to the parent tree:
490      % the command always expands its arguments!
491      \__tag_parenttree_add_objr:nn
492      {
493          #2
494      }
495      {
496          \pdf_object_ref_indexed:nn
497          { __tag/struct }{ \l__tag_struct_stack_parent_tmpa_tl }
498      }
499      % increase the int:
500      \int_gincr:N \c@g__tag_parenttree_obj_int
501  }
502 }

```

(End of definition for __tag_struct_insert_annot:nn.)

__tag_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.

```

503 \cs_new_protected:Npn \__tag_struct_insert_annot_shipout:nnn #1#2#3
504 % #1 structnum, #2 object reference, #3 StructParentNum
505 {
506     \__tag_struct_kid_OBJR_gput_right:eee
507     {
508         #1
509     }
510     {
511         #2
512     }
513     {
514         \pdf_pageobject_ref:n
515         { \int_use:N \g_shipout_readonly_int } %
516     }
517     % add the parent obj number to the parent tree:
518     % the command always expands its arguments!
519     \__tag_parenttree_add_objr:nn
520     {
521         #3
522     }
523     {
524         \pdf_object_ref_indexed:nn
525         { __tag/struct }{ #1 }
526     }
527 }

```

(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**.

```

528 \cs_new:Npn \__tag_get_data_struct_tag:
529 {

```

```

530     \exp_args:Ne
531     \tl_tail:n
532     {
533       \prop_item:cn {g__tag_struct_\g__tag_struct_stack_current_tl _prop}{S}
534     }
535   }
(End of definition for \__tag_get_data_struct_tag:.)

```

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

```

536 \cs_new:Npn \__tag_get_data_struct_id:
537 {
538   \__tag_struct_get_id:n {\g__tag_struct_stack_current_tl}
539 }
540 \endpackage
(End of definition for \__tag_get_data_struct_id:.)

```

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

```

541 \begin{base}
542 \cs_new:Npn \__tag_get_data_struct_num:
543 {
544   \g__tag_struct_stack_current_tl
545 }
546 \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.

```

547 \begin{base}
548 \cs_new:Npn \__tag_get_data_struct_counter:
549 {
550   \int_use:N \c@g__tag_struct_abs_int
551 }
552 \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`

```

553 \begin{package}
554 \cs_new_protected:Npn \__tag_struct_check_parent_child_aux:nnnnN #1#2#3#4#5
555 {
556   % #1 structure number of parent
557   % #2 key to use to retrieve role of parent (either rolemap or parentrole field)
558   % #3 structure number of parent
559   % #4 key to use to retrieve role of child (either rolemap or parentrole field)
560   % #5 tl for return value

```

get parent rolemap

```

561 \__tag_struct_get_role:nnNN
562   {#1}
563   {#2}
564 \l__tag_get_parent_tmpa_tl
565 \l__tag_get_parent_tmpb_tl

```

get child rolemap

```

566 \__tag_struct_get_role:nnNN
567   {#3}
568   {#4}
569 \l__tag_get_child_tmpa_tl
570 \l__tag_get_child_tmpb_tl

```

check

```

571 \__tag_role_check_parent_child:ooooN
572   { \l__tag_get_parent_tmpa_tl } % rolemapped from above
573   { \l__tag_get_parent_tmpb_tl } % rolemapped from above
574   { \l__tag_get_child_tmpa_tl } %
575   { \l__tag_get_child_tmpb_tl } %
576   #5
577 }

```

(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.

```

578 \cs_new_protected:Npn \__tag_struct_check_parent_child:nn #1 #2
579 % #1 structure number of parent
580 % #2 structure number of child. %
581 % This assumes that the fields rolemap/parentrole has already been filled.
582 {

```

This records if logging is on

```

583 \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
584 {
585   \prop_get:cnN{g__tag_struct_#1_prop}{tag}\l__tag_get_parent_tmpa_tl
586   \prop_get:cnN{g__tag_struct_#2_prop}{tag}\l__tag_get_parent_tmpb_tl
587   \msg_note:nnee
588   { tag }
589   { role-parent-child-check }
590   {
591     \quark_if_no_value:NTF \l__tag_get_parent_tmpa_tl
592     {??}
593     {
594       \exp_last_unbraced:No\use_ii:nn
595       { \l__tag_get_parent_tmpa_tl }
596       :
597       \exp_last_unbraced:No\use_i:nn
598       { \l__tag_get_parent_tmpa_tl }
599     }
600   }
601   {
602     \quark_if_no_value:NTF \l__tag_get_parent_tmpb_tl
603     {??}
604     {
605       \exp_last_unbraced:No\use_ii:nn

```

```

606         { \l__tag_get_parent_tmpb_tl }
607         :
608         \exp_last_unbraced:No\use_i:nn
609         { \l__tag_get_parent_tmpb_tl }
610     }
611 }
612 }
613 \__tag_struct_check_parent_child_aux:nnnnN
614 {#1}
615 {rolemap}
616 {#2}
617 {rolemap}
618 \l__tag_parent_child_check_tl

```

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

```

619 \int_compare:nNnT {\l__tag_parent_child_check_tl} = { \c__tag_role_rule_checkparent_tl }
620 {
621     \__tag_struct_check_parent_child_aux:nnnnN
622     {#1}
623     {parentrole}
624     {#2}
625     {rolemap}
626     \l__tag_parent_child_check_tl
627 }
628 \__tag_check_struct_forbidden_parent_child:onn
629 {\l__tag_parent_child_check_tl}
630 {#1}
631 {#2}
632 }
633 \cs_generate_variant:Nn \__tag_struct_check_parent_child:nn {oo}
(End of definition for \__tag_struct_check_parent_child:nn.)

```

__tag_struct_use_check_parent_child:nn

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

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

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

```

634 \cs_new_protected:Npn \__tag_struct_use_check_parent_child:nn #1 #2
635 % #1 structure number of parent
636 % #2 structure number of child. %
637 {
638     \__tag_struct_get_role:enNN
639     {#2}
640     {rolemap}
641     \l__tag_get_child_tmpa_tl
642     \l__tag_get_child_tmpb_tl
643     \str_case:onTF { \l__tag_get_child_tmpa_tl }
644     {

```

```

645     {Part} {}
646     {Div} {}
647     {NonStruct} {}
648 }
649 { %child=Part etc
650   \__tag_struct_get_role:enNN
651   {#2}
652   {parentrole}
653   \l__tag_get_child_tmpa_tl
654   \l__tag_get_child_tmpb_tl
655   \str_if_eq:noTF
656   {STASHED}{\l__tag_get_child_tmpa_tl}
657   {
658     % warn about unknown relationship
659   }
660   {
661     % test if
662     \__tag_struct_get_role:enNN
663     {#1}
664     {parentrole}
665     \l__tag_get_parent_tmpa_tl
666     \l__tag_get_parent_tmpb_tl
667     \tl_if_eq:NNTF\l__tag_get_parent_tmpa_tl \l__tag_get_child_tmpa_tl
668     {
669       \__tag_struct_check_parent_child:nn {#1}{#2}
670     }
671     {
672       %warn that parent-tag was misused.
673     }
674   }
675 }
676 {
677   %child not Part etc, normal parent child test.
678   \__tag_struct_check_parent_child:nn {#1}{#2}
679 }
680 }
681 \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.

```

682 \socket_new:nn { tag/struct/tag }{1}
683 \socket_new_plug:nnn { tag/struct/tag }{ latex-tags }
684 {
685   \prop_get:NeNTF \g__tag_role_tags_NS_prop {#1} \l__tag_tmp_unused_tl
686   {
687     \seq_set_split:Nne \l__tag_tmpa_seq { / }
688     {#1/\l__tag_tmp_unused_tl}

```

```

689     }
690     {
691         \seq_set_split:Nne \l__tag_tmpa_seq { / }
692         {#1/}
693     }
694     \tl_gset:Ne \g__tag_struct_tag_tl { \seq_item:Nn\l__tag_tmpa_seq {1} }
695     \tl_gset:Ne \g__tag_struct_tag_NS_tl{ \seq_item:Nn\l__tag_tmpa_seq {2} }
696     \__tag_check_structure_tag:N \g__tag_struct_tag_tl
697 }
698
699 \socket_new_plug:nnn { tag/struct/tag }{ pdf-tags }
700 {
701     \prop_get:NeNTF \g__tag_role_tags_NS_prop {#1} \l__tag_tmp_unused_tl
702     {
703         \seq_set_split:Nne \l__tag_tmpa_seq { / }
704         {#1/\l__tag_tmp_unused_tl}
705     }
706     {
707         \seq_set_split:Nne \l__tag_tmpa_seq { / }
708         {#1/}
709     }
710     \tl_gset:Ne \g__tag_struct_tag_tl { \seq_item:Nn\l__tag_tmpa_seq {1} }
711     \tl_gset:Ne \g__tag_struct_tag_NS_tl{ \seq_item:Nn\l__tag_tmpa_seq {2} }
712     \__tag_role_get:ooNN
713     { \g__tag_struct_tag_tl }
714     { \g__tag_struct_tag_NS_tl}
715     \l__tag_tmpa_tl
716     \l__tag_tmppb_tl
717     \tl_gset:Ne \g__tag_struct_tag_tl {\l__tag_tmpa_tl}
718     \tl_gset:Ne \g__tag_struct_tag_NS_tl{\l__tag_tmppb_tl}
719     \__tag_check_structure_tag:N \g__tag_struct_tag_tl
720 }
721 \socket_assign_plug:nn { tag/struct/tag } { latex-tags}

label (struct key)
stash (struct key) 722 \keys_define:nn { __tag / struct }
parent (struct key) 723 {
firstkid (struct key) 724     label .code:n      =
tag (struct key) 725     {
title (struct key) 726         \prop_gput:Nee\g__tag_struct_label_num_prop
title-o (struct key) 727         {#1}{\int_use:N \c@g__tag_struct_abs_int}
alt (struct key) 728         \__tag_property_record:eo
actualtext (struct key) 729         {tagpdfstruct-#1}
lang (struct key) 730         { \c__tag_property_struct_clist }
ref (struct key) 731     },
E (struct key) 732     stash .bool_set:N    = \l__tag_struct_elem_stash_bool,
phoneme (struct key) 733     parent .code:n      =
734     {
735         \bool_lazy_and:nnTF
736         {
737             \prop_if_exist_p:c { g__tag_struct\_int_eval:n {#1}_prop }
738         }
739         {
740             \int_compare_p:nNn {#1}<{\c@g__tag_struct_abs_int}

```

```

741     }
742     { \tl_set:Nc \l__tag_struct_stack_parent_tmpa_tl { \int_eval:n {#1} } }
743     {
744         \msg_warning:nnee { tag } { struct-unknown }
745         { \int_eval:n {#1} }
746         { parent~key~ignored }
747     }
748 },
749 parent .default:n = {-1},
750 parent-tag .code:n =
751 {
752     \prop_get:NcNTF \g__tag_role_tags_NS_prop {#1} \l__tag_tmp_unused_tl
753     {
754         \seq_set_split:Nne \l__tag_tmpa_seq { / }
755         {#1/\l__tag_tmp_unused_tl}
756     }
757     {
758         \seq_set_split:Nne \l__tag_tmpa_seq { / }
759         {#1/}
760     }
761     \tl_set:Nc \l__tag_struct_parenttag_tl { \seq_item:Nn \l__tag_tmpa_seq {1} }
762     \tl_set:Nc \l__tag_struct_parenttag_NS_tl { \seq_item:Nn \l__tag_tmpa_seq {2} }
763     \__tag_role_get:ooNN
764     { \l__tag_struct_parenttag_tl }
765     { \l__tag_struct_parenttag_NS_tl }
766     \l__tag_tmpa_tl
767     \l__tag_tmppb_tl
768     \tl_set:Nc \l__tag_struct_parenttag_tl { \l__tag_tmpa_tl }
769     \tl_set:Nc \l__tag_struct_parenttag_NS_tl { \l__tag_tmppb_tl }
770     \__tag_check_structure_tag:N \l__tag_struct_parenttag_tl
771 },
772 firstkid .code:n = { \tl_set:Nn \l__tag_struct_addkid_tl {left} },
773 tag .code:n = % S property
774 {
775     \socket_use:nn { tag/struct/tag }{#1}
776 },
777 title .code:n = % T property
778 {
779     \str_set_convert:Nnnn
780     \l__tag_tmpa_str
781     { #1 }
782     { default }
783     { utf16/hex }
784     \__tag_struct_prop_gput:nne
785     { \int_use:N \c@g__tag_struct_abs_int }
786     { T }
787     { <\l__tag_tmpa_str> }
788 },
789 title-o .code:n = % T property
790 {
791     \str_set_convert:Nonn
792     \l__tag_tmpa_str
793     { #1 }
794     { default }

```

```

795         { utf16/hex }
796     \_tag_struct_prop_gput:nne
797     { \int_use:N \c@g__tag_struct_abs_int }
798     { T }
799     { <\l__tag_tmpa_str> }
800 },
801 alt .code:n      = % Alt property
802 {
803     \tl_if_empty:oF{#1}
804     {
805         \str_set_convert:Noon
806         \l__tag_tmpa_str
807         { #1 }
808         { default }
809         { utf16/hex }
810         \_tag_struct_prop_gput:nne
811         { \int_use:N \c@g__tag_struct_abs_int }
812         { Alt }
813         { <\l__tag_tmpa_str> }
814     }
815 },
816 alttext .meta:n = {alt=#1},
817 actualtext .code:n = % ActualText property
818 {
819     \tl_if_empty:oF{#1}
820     {
821         \str_set_convert:Noon
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         { ActualText }
829         { <\l__tag_tmpa_str>}
830     }
831 },
832 phoneme .code:n = % Phoneme property
833 {
834     \tl_if_empty:oF{#1}
835     {
836         \str_set_convert:Noon
837         \l__tag_tmpa_str
838         { #1 }
839         { default }
840         { utf16/hex }
841         \_tag_struct_prop_gput:nne
842         { \int_use:N \c@g__tag_struct_abs_int }
843         { Phoneme }
844         { <\l__tag_tmpa_str>}
845     }
846 },
847 lang .code:n      = % Lang property
848 {

```



```

849         \__tag_struct_prop_gput:nne
850         { \int_use:N \c@g__tag_struct_abs_int }
851         { Lang }
852         { (#1) }
853     },
854 }

```

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

```

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

```

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

```

855 \cs_new_protected:Npn \__tag_struct_Ref_obj:nN #1 #2 %#1 a object reference
856 {
857     \tl_put_right:Ne#2
858     {
859         \c_space_tl#1
860     }
861 }
862
863 \cs_new_protected:Npn \__tag_struct_Ref_label:nN #1 #2 %#1 a label
864 {
865     \prop_get:NnNTF \g__tag_struct_label_num_prop {#1} \l__tag_tmpb_tl
866     {
867         \tl_put_right:Ne#2
868         {
869             \c_space_tl\tag_struct_object_ref:e{ \l__tag_tmpb_tl }
870         }
871     }
872     {
873         \msg_warning:nnn {tag}{struct-Ref-unknown}{Label~'#1'}
874     }
875 }
876 \cs_new_protected:Npn \__tag_struct_Ref_dest:nN #1 #2 %#1 a dest name
877 {
878     \prop_get:NnNTF \g__tag_struct_dest_num_prop {#1} \l__tag_tmpb_tl
879     {
880         \tl_put_right:Ne#2
881         {
882             \c_space_tl\tag_struct_object_ref:e{ \l__tag_tmpb_tl }
883         }
884     }
885     {
886         \msg_warning:nnn {tag}{struct-Ref-unknown}{Destination~'#1'}
887     }
888 }
889 \cs_new_protected:Npn \__tag_struct_Ref_num:nN #1 #2 %#1 a structure number
890 {
891     \tl_put_right:Ne#2
892     {
893         \c_space_tl\tag_struct_object_ref:e{ #1 }

```

```

894     }
895   }
896
(End of definition for \__tag_struct_Ref_obj:nN and others.)

```

```

ref (struct key)
E (struct key) 897 \keys_define:nn { __tag / struct }
898   {
899     ref .code:n      = % ref property
900     {
901       \clist_map_inline:on {#1}
902       {
903         \tag_struct_gput:nne
904         {\int_use:N \c@g__tag_struct_abs_int}{ref_label}{ ##1 }
905       }
906     },
907     E .code:n      = % E property
908     {
909       \str_set_convert:Nnon
910       \l__tag_tmpa_str
911       { #1 }
912       { default }
913       { utf16/hex }
914       \__tag_struct_prop_gput:nne
915       { \int_use:N \c@g__tag_struct_abs_int }
916       { E }
917       { <\l__tag_tmpa_str> }
918     },
919   }

```

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

```

920 \int_new:N\g__tag_struct_AFobj_int
921 \cs_generate_variant:Nn \pdffile_embed_stream:nnN {nN}
922 \cs_new_protected:Npn \__tag_struct_add_inline_AF:nn #1 #2
923 % #1 content, #2 extension
924 {
925   \tl_if_empty:nF{#1}
926   {
927     \group_begin:
928     \int_gincr:N \g__tag_struct_AFobj_int
929     \pdffile_embed_stream:nN
930     {#1}
931     {tag-AFfile\int_use:N\g__tag_struct_AFobj_int.#2}
932     \l__tag_tmpa_tl
933     \__tag_struct_add_AF:ee
934     { \int_use:N \c@g__tag_struct_abs_int }
935     { \l__tag_tmpa_tl }

```

```

936     \__tag_struct_prop_gput:nne
937     { \int_use:N \c@g__tag_struct_abs_int }
938     { AF }
939     {
940     [
941     \tl_use:c
942     { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
943     ]
944     }
945     \group_end:
946   }
947 }
948
949 \cs_generate_variant:Nn \__tag_struct_add_inline_AF:nn {on}
950 \cs_new_protected:Npn \__tag_struct_add_AF:nn #1 #2
951 % #1 struct num #2 object reference
952 {
953   \tl_if_exist:cTF
954   {
955     g__tag_struct_#1_AF_tl
956   }
957   {
958     \tl_gput_right:ce
959     { g__tag_struct_#1_AF_tl }
960     { \c_space_tl #2 }
961   }
962   {
963     \tl_new:c
964     { g__tag_struct_#1_AF_tl }
965     \tl_gset:ce
966     { g__tag_struct_#1_AF_tl }
967     { #2 }
968   }
969 }
970 \cs_generate_variant:Nn \__tag_struct_add_AF:nn {en,ee}
971 \keys_define:nn { __tag / struct }
972 {
973   AF .code:n      = % AF property
974   {
975     \pdf_object_if_exist:eTF {#1}
976     {
977       \__tag_struct_add_AF:ee
978       { \int_use:N \c@g__tag_struct_abs_int }{\pdf_object_ref:e {#1}}
979       \__tag_struct_prop_gput:nne
980       { \int_use:N \c@g__tag_struct_abs_int }
981       { AF }
982       {
983       [
984       \tl_use:c
985       { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
986       ]
987       }
988     }
989     {

```

```

990         % message?
991     }
992 },
993 AFref .code:n          = % AF property
994 {
995     \tl_if_empty:eF {#1}
996     {
997         \__tag_struct_add_AF:ee { \int_use:N \c@g__tag_struct_abs_int }{#1}
998         \__tag_struct_prop_gput:nne
999         { \int_use:N \c@g__tag_struct_abs_int }
1000         { AF }
1001         {
1002             [
1003                 \tl_use:c
1004                 { g__tag_struct\_int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
1005             ]
1006         }
1007     }
1008 },
1009 ,AFinline .code:n =
1010 {
1011     \__tag_struct_add_inline_AF:nn {#1}{txt}
1012 }
1013 ,AFinline-o .code:n =
1014 {
1015     \__tag_struct_add_inline_AF:on {#1}{txt}
1016 }
1017 ,texsource .code:n =
1018 {
1019     \group_begin:
1020     \pdfdict_put:nnn { l_pdffile/Filespec } {Desc}{(TeX~source)}
1021     \pdfdict_put:nnn { l_pdffile/Filespec }{AFRelationship} { /Source }
1022     \__tag_struct_add_inline_AF:on {#1}{tex}
1023     \group_end:
1024 }
1025 ,mathml .code:n =
1026 {
1027     \group_begin:
1028     \pdfdict_put:nnn { l_pdffile/Filespec } {Desc}{(mathml~representation)}
1029     \pdfdict_put:nnn { l_pdffile/Filespec }{AFRelationship} { /Supplement }
1030     \pdfdict_put:nne { l_pdffile }{Subtype}
1031     { \pdf_name_from_unicode_e:n{application/mathml+xml} }
1032     \__tag_struct_add_inline_AF:on {#1}{xml}
1033     \group_end:
1034 }
1035 }

```

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!

```

1036 \keys_define:nn { __tag / setup }
1037 {
1038     root-AF .code:n =
1039     {
1040         \pdf_object_if_exist:nTF {#1}

```

```

1041     {
1042       \__tag_struct_add_AF:ee { 1 }{\pdf_object_ref:n {#1}}
1043       \__tag_struct_prop_gput:nne
1044         { 1 }
1045         { AF }
1046         {
1047           [
1048             \tl_use:c
1049             { g__tag_struct_1_AF_tl }
1050           ]
1051         }
1052       }
1053     {
1054       }
1055   },
1056 }
1057 }

```

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

```

1058 \keys_define:nn { __tag / setup }
1059 {
1060   root-supplemental-file .code:n =
1061   {
1062     \group_begin:
1063     \pdfdict_put:nnn {l_pdffile/Filespec} {AFRelationship}{/Supplement}
1064     \int_gincr:N \g__tag_unique_cnt_int
1065     \pdffile_embed_file:eee
1066     {#1}
1067     {#1}
1068     {__tag_latex_css_\int_use:N\g__tag_unique_cnt_int}
1069     \keys_set:nn
1070     {__tag / setup}
1071     {root-AF={__tag_latex_css_\int_use:N\g__tag_unique_cnt_int}}
1072     \group_end:
1073   }
1074 }

```

catalog-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.

```

1075 \keys_define:nn { __tag / setup }
1076 {
1077   catalog-supplemental-file .code:n =
1078   {
1079     \group_begin:
1080     \pdfdict_put:nnn {l_pdffile/Filespec} {AFRelationship}{/Supplement}
1081     \int_gincr:N \g__tag_unique_cnt_int
1082     \pdffile_embed_file:eee
1083     {#1}
1084     {#1}
1085     {__tag_latex_css_\int_use:N\g__tag_unique_cnt_int}
1086     \pdfmanagement_add:nne
1087     {Catalog}
1088     {AF}

```

```

1089         {\pdf_object_ref:e{__tag_latex_css_int_use:N\g__tag_unique_cnt_int }}
1090     \group_end:
1091 }
1092 }

```

6 User commands

We allow to set a language by default

`\l__tag_struct_lang_tl`

```

1093 \tl_new:N \l__tag_struct_lang_tl
1094 \</package>
(End of definition for \g__tag_struct_AFobj_int and \l__tag_struct_lang_tl.)

\tag_struct_begin:n
\tag_struct_end:
1095 <base>\cs_new_protected:Npn \tag_struct_begin:n #1 {\int_gincr:N \c@g__tag_struct_abs_int}
1096 <base>\cs_new_protected:Npn \tag_struct_end: {}
1097 <base>\cs_new_protected:Npn \tag_struct_end:n {}
1098 <*package|debug>
1099 <package>\cs_set_protected:Npn \tag_struct_begin:n #1 %#1 key-val
1100 <debug>\cs_set_protected:Npn \tag_struct_begin:n #1 %#1 key-val
1101 {
1102 <package>\__tag_check_if_active_struct:T
1103 <debug>\__tag_check_if_active_struct:TF
1104 {
1105     \group_begin:
1106     \int_gincr:N \c@g__tag_struct_abs_int
1107     \__tag_struct_prop_new:c { g__tag_struct_int_eval:n { \c@g__tag_struct_abs_int }_prop }
1108 <debug>     \prop_new:c { g__tag_struct_debug_int_eval:n { \c@g__tag_struct_abs_int }_prop
1109     \__tag_struct_seq_new:c { g__tag_struct_kids_int_eval:n { \c@g__tag_struct_abs_int }_seq }
1110 <debug>     \seq_new:c { g__tag_struct_debug_kids_int_eval:n { \c@g__tag_struct_abs_int }_
1111     \pdf_object_new_indexed:nn { __tag/struct }
1112     { \c@g__tag_struct_abs_int }
1113     \__tag_struct_prop_gput:nnn
1114     { \int_use:N \c@g__tag_struct_abs_int }
1115     { Type }
1116     { /StructElem }
1117     \tl_if_empty:NF \l__tag_struct_lang_tl
1118     {
1119         \__tag_struct_prop_gput:nne
1120         { \int_use:N \c@g__tag_struct_abs_int }
1121         { Lang }
1122         { (\l__tag_struct_lang_tl) }
1123     }
1124     \__tag_struct_prop_gput:nnn
1125     { \int_use:N \c@g__tag_struct_abs_int }
1126     { Type }
1127     { /StructElem }
1128
1129     \tl_set:Nn \l__tag_struct_stack_parent_tmpa_tl {-1}
1130     \keys_set:nn { __tag / struct } { #1 }

```

```

1131     \__tag_struct_set_tag_info:eoo
1132     { \int_use:N \c@g__tag_struct_abs_int }
1133     { \g__tag_struct_tag_tl }
1134     { \g__tag_struct_tag_NS_tl }
1135     \__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.

```

1136     \int_compare:nNnT { \l__tag_struct_stack_parent_tmpa_tl } = { -1 }
1137     {
1138         \seq_get:NNF
1139         \g__tag_struct_stack_seq
1140         \l__tag_struct_stack_parent_tmpa_tl
1141         {
1142             \msg_error:nn { tag } { struct-faulty-nesting }
1143         }
1144     }
1145     \seq_gpush:NV \g__tag_struct_stack_seq \c@g__tag_struct_abs_int
1146     \__tag_role_get:ooNN
1147     { \g__tag_struct_tag_tl }
1148     { \g__tag_struct_tag_NS_tl }
1149     \l__tag_struct_roletag_tl
1150     \l__tag_struct_roletag_NS_tl

```

We push the role tag on the stack:

```

1151     \seq_gpush:Ne \g__tag_struct_tag_stack_seq
1152     {{\g__tag_struct_tag_tl}{\l__tag_struct_roletag_tl}}
1153     \tl_gset:NV \g__tag_struct_stack_current_tl \c@g__tag_struct_abs_int
1154     \__tag_struct_set_attribute:
1155     %\seq_show:N \g__tag_struct_stack_seq

```

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

```

1156     \__tag_struct_prop_gput:nne
1157     { \int_use:N \c@g__tag_struct_abs_int }
1158     { rolemap }
1159     {
1160         {\l__tag_struct_roletag_tl}{\l__tag_struct_roletag_NS_tl}
1161     }

```

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.

```

1162     \str_case:onTF { \l__tag_struct_roletag_tl }
1163     {
1164         {Part} {}
1165         {Div} {}
1166         {NonStruct} {}
1167     }
1168     {
1169         \bool_if:NTF \l__tag_struct_elem_stash_bool
1170         {
1171             \__tag_struct_prop_gput:nne

```

```

1172         { \int_use:N \c@g__tag_struct_abs_int }
1173         { parentrole }
1174         {
1175             {\l__tag_struct_parenttag_tl}{\l__tag_struct_parenttag_NS_tl}
1176         }
1177     }
1178     {
1179         \prop_get:cnNT
1180         { g__tag_struct_ \l__tag_struct_stack_parent_tmpa_tl _prop }
1181         { parentrole }
1182         \l__tag_get_tmpc_tl
1183         {
1184             \__tag_struct_prop_gput:nno
1185             { \int_use:N \c@g__tag_struct_abs_int }
1186             { parentrole }
1187             {
1188                 \l__tag_get_tmpc_tl
1189             }
1190         }
1191     }
1192 }
1193 {
1194     \__tag_struct_prop_gput:nne
1195     { \int_use:N \c@g__tag_struct_abs_int }
1196     { parentrole }
1197     {
1198         {\l__tag_struct_roletag_tl}{\l__tag_struct_roletag_NS_tl}
1199     }
1200 }
1201 \bool_if:NF
1202     \l__tag_struct_elem_stash_bool
1203     {

```

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.

```

1204         \socket_use:nn{tag/check/parent-child}
1205         {
1206             \__tag_struct_check_parent_child:oo
1207             { \l__tag_struct_stack_parent_tmpa_tl }
1208             { \int_use:N \c@g__tag_struct_abs_int }
1209         }

```

Set the Parent structure number.

```

1210         \__tag_struct_prop_gput:nne
1211         { \int_use:N \c@g__tag_struct_abs_int }
1212         { parentnum }
1213         {
1214             \l__tag_struct_stack_parent_tmpa_tl
1215         }
1216 %record this structure as kid:
1217 %\tl_show:N \g__tag_struct_stack_current_tl
1218 %\tl_show:N \l__tag_struct_stack_parent_tmpa_tl
1219 \use:c { __tag_struct_kid_struct_gput_ \l__tag_struct_addkid_tl :ee }
1220     { \l__tag_struct_stack_parent_tmpa_tl }

```



```

1221         { \g__tag_struct_stack_current_t1 }
1222         %\prop_show:c { g__tag_struct_\g__tag_struct_stack_current_t1 _prop }
1223         %\seq_show:c {g__tag_struct_kids_\l__tag_struct_stack_parent_tmpa_t1 _seq}
1224     }

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.

1225 <debug>         \prop_gset_eq:cc
1226 <debug>         { g__tag_struct_debug_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
1227 <debug>         { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
1228 <debug>         \prop_gput:cne
1229 <debug>         { g__tag_struct_debug_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
1230 <debug>         { parentnum }
1231 <debug>         {
1232 <debug>             \bool_if:NTF \l__tag_struct_elem_stash_bool
1233 <debug>             {no-parent:~stashed}
1234 <debug>             {
1235 <debug>                 \l__tag_struct_stack_parent_tmpa_t1\c_space_t1 =~
1236 <debug>                 \prop_item:cn{ g__tag_struct_\l__tag_struct_stack_parent_tmpa_t1 _p
1237 <debug>             }
1238 <debug>         }
1239 <debug>         \prop_gput:cne
1240 <debug>         { g__tag_struct_debug_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
1241 <debug>         { NS }
1242 <debug>         { \g__tag_struct_tag_NS_t1 }

1243         %\prop_show:c { g__tag_struct_\g__tag_struct_stack_current_t1 _prop }
1244         %\seq_show:c {g__tag_struct_kids_\l__tag_struct_stack_parent_tmpa_t1 _seq}
1245 <debug> \__tag_debug_struct_begin_insert:n { #1 }
1246         \group_end:
1247     }
1248 <debug>{ \__tag_debug_struct_begin_ignore:n { #1 }}
1249     }
1250 <package>\cs_set_protected:Nn \tag_struct_end:
1251 <debug>\cs_set_protected:Nn \tag_struct_end:
1252     { %take the current structure num from the stack:
1253       %the objects are written later, lua mode hasn't all needed info yet
1254       %\seq_show:N \g__tag_struct_stack_seq
1255 <package>\__tag_check_if_active_struct:T
1256 <debug>\__tag_check_if_active_struct:TF
1257     {
1258         \seq_gpop:NN \g__tag_struct_tag_stack_seq \l__tag_tmpa_t1
1259         \seq_gpop:NNTF \g__tag_struct_stack_seq \l__tag_tmpa_t1
1260         {
1261             \__tag_check_info_closing_struct:o { \g__tag_struct_stack_current_t1 }
1262         }
1263         { \__tag_check_no_open_struct: }
1264         % get the previous one, shouldn't be empty as the root should be there
1265         \seq_get:NNTF \g__tag_struct_stack_seq \l__tag_tmpa_t1
1266         {
1267             \tl_gset:No \g__tag_struct_stack_current_t1 { \l__tag_tmpa_t1 }
1268             \__tag_struct_set_attribute:
1269         }
1270         {

```

```

1271     \__tag_check_no_open_struct:
1272   }
1273   \seq_get:NNT \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
1274   {
1275     \tl_gset:Ne \g__tag_struct_tag_tl
1276       { \exp_last_unbraced:No\use_i:nn { \l__tag_tmpa_tl } }
1277     \prop_get:NoNT\g__tag_role_tags_NS_prop { \g__tag_struct_tag_tl } \l__tag_tmpa_tl
1278     {
1279       \tl_gset:Ne \g__tag_struct_tag_NS_tl { \l__tag_tmpa_tl }
1280     }
1281   }
1282   <debug>\__tag_debug_struct_end_insert:
1283   }
1284   <debug>{\__tag_debug_struct_end_ignore:}
1285   }
1286
1287   \cs_set_protected:Npn \tag_struct_end:n #1
1288   {
1289     <debug> \__tag_check_if_active_struct:T{\__tag_debug_struct_end_check:n{#1}}
1290     \tag_struct_end:
1291   }
1292   </package | debug>

```

(End of definition for \tag_struct_begin:n and \tag_struct_end:. These functions are documented on page 112.)

\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.

```

1293   <base>\cs_new_protected:Npn \tag_struct_use:n #1 {}
1294   <*package | debug>
1295   \cs_set_protected:Npn \tag_struct_use:n #1 %#1 is the label
1296   {
1297     \__tag_check_if_active_struct:T
1298     {
1299       \prop_if_exist:cTF
1300       { g__tag_struct_ \property_ref:enn{tagpdfstruct-#1}{tagstruct}{unknown}_prop } %
1301       {
1302         \__tag_check_struct_used:n {#1}
1303         \tl_set:Ne \l__tag_get_child_tmpa_tl
1304           { \property_ref:enn{tagpdfstruct-#1}{tagstruct}{1} } }

```

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

```

1305     \__tag_struct_kid_struct_gput_right:ee
1306     { \g__tag_struct_stack_current_tl }
1307     { \l__tag_get_child_tmpa_tl }

```

add the current structure to the labeled one as parents

```

1308     \__tag_prop_gput:cne
1309     { g__tag_struct_ \l__tag_get_child_tmpa_tl _prop }
1310     { parentnum }
1311     {
1312       \g__tag_struct_stack_current_tl
1313     }

```

debug code

```

1314   <debug> \prop_gput:cne

```

```

1315 <debug>          { g__tag_struct_debug_ \l__tag_get_child_tmpa_tl _prop }
1316 <debug>          { parentnum }
1317 <debug>          {
1318 <debug>          \g__tag_struct_stack_current_tl\c_space_tl=~
1319 <debug>          \g__tag_struct_tag_tl
1320 <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.

```

1321          \socket_use:nn{tag/check/parent-child}
1322          {
1323          \__tag_struct_use_check_parent_child:oo
1324          { \g__tag_struct_stack_current_tl }
1325          { \l__tag_get_child_tmpa_tl }
1326          }
1327          }
1328          {
1329          \msg_warning:nnn{ tag }{struct-label-unknown}{#1}
1330          }
1331          }
1332          }
1333 </package|debug>

```

(End of definition for `\tag_struct_use:n`. This function is documented on page 112.)

`\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.

```

1334 <base>\cs_new_protected:Npn \tag_struct_use_num:n #1 {}
1335 <*package|debug>
1336 \cs_set_protected:Npn \tag_struct_use_num:n #1 %#1 is structure number
1337 {
1338   \__tag_check_if_active_struct:T
1339   {
1340     \prop_if_exist:cTF
1341     { g__tag_struct_#1_prop } %
1342     {
1343       \prop_get:cnNT
1344       {g__tag_struct_#1_prop}
1345       {parentnum}
1346       \l__tag_tmpa_tl
1347       {
1348         \msg_warning:nnn { tag } {struct-used-twice} {#1}
1349       }

```

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

```

1350       \__tag_struct_kid_struct_gput_right:ee
1351       { \g__tag_struct_stack_current_tl }
1352       { #1 }

```

add the current structure to #1 as parent

```

1353       \__tag_struct_prop_gput:nne

```

```

1354         { #1 }
1355         { parentnum }
1356         {
1357             \g__tag_struct_stack_current_tl
1358         }
1359     <debug>         \prop_gput:cne
1360     <debug>         { g__tag_struct_debug_#1_prop }
1361     <debug>         { parentnum }
1362     <debug>         {
1363     <debug>             \g__tag_struct_stack_current_tl\c_space_tl=~
1364     <debug>             \g__tag_struct_tag_tl
1365     <debug>         }

```

check if the tag is allowed as child.

```

1366         \socket_use:nn{tag/check/parent-child}
1367         {
1368             \__tag_struct_use_check_parent_child:oo
1369             {\g__tag_struct_stack_current_tl}
1370             {#1}
1371         }
1372     }
1373     {
1374         \msg_warning:nnn{ tag }{struct-label-unknown}{#1}
1375     }
1376 }
1377 }
1378 </package>|debug>

```

(End of definition for \tag_struct_use_num:n. This function is documented on page 112.)

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

```

1379 <*package>
1380 \cs_new:Npn \tag_struct_object_ref:n #1
1381 {
1382     \pdf_object_ref_indexed:nn {\__tag/struct}{ #1 }
1383 }
1384 \cs_generate_variant:Nn \tag_struct_object_ref:n {e}
1385 </package>

```

(End of definition for \tag_struct_object_ref:n. This function is documented on page 112.)

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

dictionary, so for example /O /Layout /BBox [10 10 50 50]. The content is stored in an object and the object reference is then added to the /A.

```

1386 <base>\cs_new_protected:Npn \tag_struct_gput:nnn #1 #2 #3{
1387 <*package>
1388 \cs_set_protected:Npn \tag_struct_gput:nnn #1 #2 #3
1389 {
1390   \cs_if_exist_use:cF {__tag_struct_gput_data_#2:nn}
1391   { %warning??
1392     \use_none:nn
1393   }
1394   {#1}{#3}
1395 }
1396 \cs_generate_variant:Nn \tag_struct_gput:nnn {ene,nne}
1397 </package>

```

(End of definition for \tag_struct_gput:nnn. This function is documented on page 112.)

__tag_struct_gput_data_ref_aux:nnn

```

1398 <*package>
1399 \cs_new_protected:Npn \__tag_struct_gput_data_ref_aux:nnn #1 #2 #3
1400   % #1 receiving struct num, #2 key word #3 value
1401   {
1402     \prop_get:cnNTF
1403     { g__tag_struct_#1_prop }
1404     {Ref}
1405     \l__tag_get_tmpc_tl
1406     {
1407       \tl_put_right:No \l__tag_get_tmpc_tl
1408       {\cs:w __tag_struct_Ref_#2:nN \cs_end: {#3},}
1409     }
1410     {
1411       \tl_set:No \l__tag_get_tmpc_tl
1412       {\cs:w __tag_struct_Ref_#2:nN \cs_end: {#3},}
1413     }
1414     \__tag_struct_prop_gput:nno
1415     { #1 }
1416     { Ref }
1417     { \l__tag_get_tmpc_tl }
1418   }
1419 \cs_new_protected:Npn \__tag_struct_gput_data_ref:nn #1 #2
1420   {
1421     \__tag_struct_gput_data_ref_aux:nnn {#1}{obj}{#2}
1422   }
1423 \cs_new_protected:Npn \__tag_struct_gput_data_ref_label:nn #1 #2
1424   {
1425     \__tag_struct_gput_data_ref_aux:nnn {#1}{label}{#2}
1426   }
1427 \cs_new_protected:Npn \__tag_struct_gput_data_ref_dest:nn #1 #2
1428   {
1429     \__tag_struct_gput_data_ref_aux:nnn {#1}{dest}{#2}
1430   }
1431 \cs_new_protected:Npn \__tag_struct_gput_data_ref_num:nn #1 #2
1432   {
1433     \__tag_struct_gput_data_ref_aux:nnn {#1}{num}{#2}

```

```

1434 }
1435
1436 \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
1437 \cs_new_protected:Npn \__tag_struct_gput_data_attribute:nn #1 #2
1438 {
1439   \pdf_object_unnamed_write:nn {dict} {#2}
1440   \prop_get:cnNTF { g__tag_struct_#1_prop }{A} \l__tag_tmpa_tl
1441   {
1442     \tl_remove_once:Nn\l__tag_tmpa_tl{[]}
1443     \tl_remove_once:Nn\l__tag_tmpa_tl{[] }
1444     \__tag_prop_gput:cne { g__tag_struct_#1_prop }
1445     { A }
1446     {
1447       [ \l__tag_tmpa_tl \c_space_tl \pdf_object_ref_last: ]
1448     }
1449   }
1450   {
1451     \__tag_prop_gput:cne { g__tag_struct_#1_prop }
1452     { A }
1453     { \pdf_object_ref_last: }
1454   }
1455 }
(End of definition for \__tag_struct_gput_data_attribute:nn.)

```

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

\tag_struct_insert_annot:ee

\tag_struct_insert_annot:ee

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

```

1456 \cs_new_protected:Npn \tag_struct_insert_annot:nn #1 #2 %#1 should be an object reference
1457                                     %#2 struct parent num
1458 {
1459   \__tag_check_if_active_struct:T
1460   {
1461     \__tag_struct_insert_annot:nn {#1}{#2}
1462   }
1463 }
1464
1465 \cs_generate_variant:Nn \tag_struct_insert_annot:nn {xx,ee}
1466 \cs_new:Npn \tag_struct_parent_int: {\int_use:c { c@g__tag_parenttree_obj_int }}
1467
1468 \</package>
1469
(End of definition for \tag_struct_insert_annot:nn and \tag_struct_parent_int:. These functions
are documented on page 112.)

```

7 Attributes and attribute classes

```

1470 \*header

```

```

1471 \ProvidesExplPackage {tagpdf-attr-code} {2026-01-29} {0.99y}
1472 {part of tagpdf - code related to attributes and attribute classes}
1473 \</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`

```

1474 <*package>
1475 \prop_new:N \g__tag_attr_entries_prop
1476 \prop_new_linked:N \g__tag_attr_class_used_prop
1477 \tl_new:N \l__tag_attr_value_tl
1478 \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.

```

1479 \seq_new:N \g__tag_attr_class_used_seq
(End of definition for \g__tag_attr_entries_prop and others.)

```

7.2 Commands and keys

`__tag_attr_new_entry:nn` This allows to define attributes. Defined attributes are stored in a global property.
`role/new-attribute (setup-key)` `role/new-attribute` expects two brace group, the name and the content. The content typically needs an /O key for the owner. An example look like this.
`newattribute (deprecated)` TODO: consider to put them directly in the ClassMap, that is perhaps more effective.

```

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

1480 \cs_new_protected:Npn \__tag_attr_new_entry:nn #1 #2 %#1:name, #2: content
1481 {
1482   \prop_gput:Nen \g__tag_attr_entries_prop
1483     {\pdf_name_from_unicode_e:n{#1}}{#2}
1484 }
1485
1486 \cs_generate_variant:Nn \__tag_attr_new_entry:nn {ee}
1487 \keys_define:nn { __tag / setup }
1488 {
1489   role/new-attribute .code:n =
1490   {
1491     \__tag_attr_new_entry:nn #1
1492   }
}

deprecated name
1493 ,newattribute .code:n =
1494 {

```

```

1495         \__tag_attr_new_entry:nn #1
1496     },
1497 }

```

(End of definition for `__tag_attr_new_entry:nn`, `role/new-attribute` (setup-key), and `newattribute` (deprecated). These functions are documented on page 115.)

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

```

1498 \keys_define:nn { __tag / struct }
1499 {
1500     attribute-class .code:n =
1501     {
1502         \clist_set:Ne \l__tag_tmpa_clist { #1 }
1503         \seq_set_from_clist:NN \l__tag_tmpb_seq \l__tag_tmpa_clist
1504         \seq_set_map_e:NNn \l__tag_tmpa_seq \l__tag_tmpb_seq
1505         {
1506             \pdf_name_from_unicode_e:n {##1}
1507         }
1508         \seq_map_inline:Nn \l__tag_tmpa_seq
1509         {
1510             \prop_get:NnNF \g__tag_attr_entries_prop {##1}\l__tag_tmpa_tl
1511             {
1512                 \msg_error:nnn { tag } { attr-unknown } { ##1 }
1513             }
1514             \prop_gput:Nnn\g__tag_attr_class_used_prop { ##1} {}
1515         }
1516         \tl_set:Ne \l__tag_tmpa_tl
1517         {
1518             \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}
1519             \seq_use:Nn \l__tag_tmpa_seq { \c_space_tl }
1520             \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}
1521         }
1522         \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 0 }
1523         {
1524             \__tag_struct_prop_gput:nne
1525             { \int_use:N \c@g__tag_struct_abs_int }
1526             { C }
1527             { \l__tag_tmpa_tl }
1528             %\prop_show:c { g__tag_struct_\int_eval:n { \c@g__tag_struct_abs_int }_prop }
1529         }
1530     }
1531 }

```

attribute (*struct key*)

```

1532 \keys_define:nn { __tag / struct }
1533 {
1534     attribute .code:n = % A property (attribute, value currently a dictionary)
1535     {
1536         \clist_set:Ne \l__tag_tmpa_clist { #1 }
1537         \clist_if_empty:NF \l__tag_tmpa_clist
1538         {
1539             \seq_set_from_clist:NN \l__tag_tmpb_seq \l__tag_tmpa_clist

```


we convert the names into pdf names with slash

```

1540 \seq_set_map_e:Nn \l__tag_tmpa_seq \l__tag_tmpb_seq
1541 {
1542   \pdf_name_from_unicode_e:n {##1}
1543 }
1544 \tl_set:N \l__tag_attr_value_tl
1545 {
1546   \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}%
1547 }
1548 \seq_map_inline:Nn \l__tag_tmpa_seq
1549 {
1550   \prop_get:NnNF \g__tag_attr_entries_prop {##1}\l__tag_tmp_unused_tl
1551   {
1552     \msg_error:nnn { tag } { attr-unknown } { ##1 }
1553   }
1554   \prop_get:NnNF \g__tag_attr_objref_prop {##1}\l__tag_tmpa_tl
1555   {%\prop_show:N \g__tag_attr_entries_prop
1556     \pdf_object_unnamed_write:ne
1557     { dict }
1558     {
1559       \prop_item:Nn \g__tag_attr_entries_prop {##1}
1560     }
1561     \prop_gput:Nne \g__tag_attr_objref_prop {##1} {\pdf_object_ref_last:}
1562   }
1563   \tl_put_right:N \l__tag_attr_value_tl
1564   {
1565     \c_space_tl
1566     \prop_item:Nn \g__tag_attr_objref_prop {##1}
1567   }
1568   % \tl_show:N \l__tag_attr_value_tl
1569   }
1570   \tl_put_right:N \l__tag_attr_value_tl
1571   { %[
1572     \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}%
1573   }
1574   % \tl_show:N \l__tag_attr_value_tl
1575   \__tag_struct_prop_gput:nne
1576   { \int_use:N \c@g__tag_struct_abs_int }
1577   { A }
1578   { \l__tag_attr_value_tl }
1579 }
1580 },
1581 }
1582 \end{package}

```

The tagpdf-luatex.def
 Driver for luatex
 Part of the tagpdf package
 Ulrike Fischer
 Version 0.99y, released 2026-01-29

Part IX

```
1 <@@=tag>
2 <*luatex>
3 \ProvidesExplFile {tagpdf-luatex.def} {2026-01-29} {0.99y}
4 {tagpdf~driver~for~luatex}
```

1 Loading the lua

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

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

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

```

    \__tag_prop_new:N
    \__tag_seq_new:N
    \__tag_prop_gput:Nnn
    \__tag_seq_gput_right:Nn
    \__tag_seq_gput_left:Nn
    \__tag_seq_item:cn
    \__tag_prop_item:cn
    \__tag_seq_show:N
    \__tag_prop_show:N
9 \cs_set_protected:Npn \__tag_prop_new:N #1
10 {
11   \prop_new:N #1
12   \lua_now:e { ltx.__tag.tables['\cs_to_str:N#1'] = {} }
13 }
14
15 \cs_set_protected:Npn \__tag_prop_new_linked:N #1
16 {
17   \prop_new_linked:N #1
18   \lua_now:e { ltx.__tag.tables['\cs_to_str:N#1'] = {} }
19 }
20
21
22 \cs_set_protected:Npn \__tag_seq_new:N #1
23 {
24   \seq_new:N #1
25   \lua_now:e { ltx.__tag.tables['\cs_to_str:N#1'] = {} }
26 }
27
28
29 \cs_set_protected:Npn \__tag_prop_gput:Nnn #1 #2 #3
30 {
31   \prop_gput:Nnn #1 { #2 } { #3 }
32   \lua_now:e { ltx.__tag.tables['\cs_to_str:N#1'] ["#2"] = "\lua_escape:n{#3}" }
33 }
34
35 \cs_set_protected:Npn \__tag_seq_gput_right:Nn #1 #2
```

```

36 {
37   \seq_gput_right:Nn #1 { #2 }
38   \lua_now:e { table.insert(ltx.__tag.tables['\cs_to_str:N#1'], "#2") }
39 }

```

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

```

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

```

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

```

71 </luatex>

```

The module declaration

```

72 <*lua>
73 -- tagpdf.lua
74 -- Ulrike Fischer
75
76 local ProvidesLuaModule = {
77   name      = "tagpdf",
78   version   = "0.99y",      --TAGVERSION
79   date      = "2026-01-29", --TAGDATE
80   description = "tagpdf lua code",
81   license   = "The LATEX Project Public License 1.3c"
82 }
83

```

```

84 if luatexbase and luatexbase.provides_module then
85   luatexbase.provides_module (ProvidesLuaModule)
86 end
87
88 --[[
89 The code has quite probably a number of problems
90 - more variables should be local instead of global
91 - the naming is not always consistent due to the development of the code
92 - the traversing of the shipout box must be tested with more complicated setups
93 - it should probably handle more node types
94 -
95 --]]
96

```

Some comments about the lua structure.

```

97 --[[
98 the main table is named ltx.__tag. It contains the functions and also the data
99 collected during the compilation.
100
101 ltx.__tag.mc      will contain mc connected data.
102 ltx.__tag.role    will contain data related to parent-child relations.
103 ltx.__tag.struct  will contain structure related data.
104 ltx.__tag.page    will contain page data
105 ltx.__tag.tables  contains also data from mc and struct (from older code). This needs cleaning
106                 There are certainly dublettes, but I don't dare yet ...
107 ltx.__tag.func    will contain (public) functions.
108 ltx.__tag.trace   will contain tracing/logging functions.
109 local functions starts with __
110 functions meant for users will be in ltx.tag
111
112 functions
113 ltx.__tag.func.get_num_from (tag):   takes a tag (string) and returns the id number
114 ltx.__tag.func.output_num_from (tag): takes a tag (string) and prints (to tex) the id number
115 ltx.__tag.func.get_tag_from (num):   takes a num and returns the tag
116 ltx.__tag.func.output_tag_from (num): takes a num and prints (to tex) the tag
117 ltx.__tag.func.store_mc_data (num,key,data): stores key=data in ltx.__tag.mc[num]
118 ltx.__tag.func.store_mc_label (label,num): stores label=num in ltx.__tag.mc.labels
119 ltx.__tag.func.store_mc_kid (mcnum,kid,page): stores the mc-kids of mcnum on page page
120 ltx.__tag.func.store_mc_in_page(mcnum,mcpagecnt,page): stores in the page table the number of
121 ltx.__tag.func.store_struct_mcabs (structnum,mcnum): stores relations structnum<->mcnum (abs
122 ltx.__tag.func.mc_insert_kids (mcnum): inserts the /K entries for mcnum by wandering through
123 ltx.__tag.func.mark_page_elements(box,mcpagecnt,mccntprev,mcopen,name,mctypeprev) : the main
124 ltx.__tag.func.mark_shipout (): a wrapper around the core function which inserts the last EN
125 ltx.__tag.func.fill_parent_tree_line (page): outputs the entries of the parenttree for this
126 ltx.__tag.func.output_parenttree(): outputs the content of the parenttree
127 ltx.__tag.func.pdf_object_ref(name,index): outputs the object reference for the object name
128 ltx.__tag.func.markspaceon(), ltx.__tag.func.markspaceoff(): (de)activates the marking of po
129 ltx.__tag.trace.show_mc_data (num,loglevel): shows ltx.__tag.mc[num] is the current log leve
130 ltx.__tag.trace.show_all_mc_data (max,loglevel): shows a maximum about mc's if the current l
131 ltx.__tag.trace.show_seq: shows a sequence (array)
132 ltx.__tag.trace.show_struct_data (num): shows data of structure num
133 ltx.__tag.trace.show_prop: shows a prop
134 ltx.__tag.trace.log
135 ltx.__tag.trace.showspace : boolean
136

```

```

137 ltx.tag.get_structnum: number, shows the current structure number
138 ltx.tag.get_structnum_next: number, shows the next structure number
139 --]]
140

```

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

```

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

```

148 local tagunmarkedbool= token.create("g__tag_tagunmarked_bool")
149 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.

```

150 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.

```

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

```

```

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

```

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

```

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

```

\
204 local __tag_get_struct_num =
205   function()
206     local a = token.get_macro("g__tag_struct_stack_current_t1")
207     return a
208   end
209
210 local __tag_get_struct_counter =
211   function()
212     local a = tex.getcount("c@g__tag_struct_abs_int")
213     return a
214   end

```

```

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

```

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

```

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

```

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

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

```

242 local __tag_pairs_prop =
243   function (prop)
244     local a = {}
245     for n in pairs(prop) do tableinsert(a, n) end
246     table.sort(a)
247     local i = 0 -- iterator variable
248     local iter = function () -- iterator function
249       i = i + 1
250       if a[i] == nil then return nil

```

```

251         else return a[i], prop[a[i]]
252     end
253 end
254 return iter
255 end
256
257
258 function ltx.__tag.trace.show_prop (prop)
259 if (type(prop) == "table") then
260   for i,v in __tag_pairs_prop (prop) do
261     __tag_log ("[" .. i .. "] => " .. tostring(v),1)
262   end
263 else
264   __tag_log ("prop " .. tostring(prop) .. " not found or not a table",1)
265 end
266 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

```

267 function ltx.__tag.trace.show_mc_data (num,loglevel)
268 if ltx.__tag and ltx.__tag.mc and ltx.__tag.mc[num] then
269   for k,v in pairs(ltx.__tag.mc[num]) do
270     __tag_log ("mc"..num..": " ..tostring(k).."=>" ..tostring(v),loglevel)
271   end
272   if ltx.__tag.mc[num]["kids"] then
273     __tag_log ("mc" .. num .. " has " .. #ltx.__tag.mc[num]["kids"] .. " kids",loglevel)
274     for k,v in ipairs(ltx.__tag.mc[num]["kids"]) do
275       __tag_log ("mc " .. num .. " kid "..k.." =>" .. v.kid.." on page " ..v.page,loglevel)
276     end
277   end
278 else
279   __tag_log ("mc"..num.." not found",loglevel)
280 end
281 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.

```

282 function ltx.__tag.trace.show_all_mc_data (min,max,loglevel)
283 for i = min, max do
284   ltx.__tag.trace.show_mc_data (i,loglevel)
285 end
286 texio.write_nl("")
287 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.

```

288 function ltx.__tag.trace.show_struct_data (num)
289 if ltx.__tag and ltx.__tag.struct and ltx.__tag.struct[num] then
290   for k,v in ipairs(ltx.__tag.struct[num]) do
291     __tag_log ("struct "..num..": " ..tostring(k).."=>" ..tostring(v),1)
292   end

```



```

293 else
294   __tag_log    ("struct "..num.." not found ",1)
295 end
296 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.

```

297 local __tag_get_mc_cnt_type_tag = function (n)
298   local mcnt      = nodegetattribute(n,mccntattributeid) or -1
299   local mctype    = nodegetattribute(n,mctypeattributeid) or -1
300   local tag       = ltx.__tag.func.get_tag_from(mctype)
301   return mcnt,mctype,tag
302 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.

```

303 local function __tag_get_mathsubtype (mathnode)
304   if mathnode.subtype == 0 then
305     subtype = "beginmath"
306   else
307     subtype = "endmath"
308   end
309   return subtype
310 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.

```

311 ltx.__tag.tables.role_tag_attribute = {}
312 ltx.__tag.tables.role_attribute_tag = {}
(End of definition for ltx.__tag.tables.role_tag_attribute.)

```

`ltx.__tag.func.alloctag`

```

313 local __tag_alloctag =
314   function (tag)
315     if not ltx.__tag.tables.role_tag_attribute[tag] then
316       table.insert(ltx.__tag.tables.role_attribute_tag,tag)
317       ltx.__tag.tables.role_tag_attribute[tag]=#ltx.__tag.tables.role_attribute_tag
318       __tag_log    ("Add "..tag.." " "..ltx.__tag.tables.role_tag_attribute[tag],3)
319     end
320   end
321 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.

```

322 local __tag_get_num_from =
323   function (tag)
324     if ltx.__tag.tables.role_tag_attribute[tag] then
325       a= ltx.__tag.tables.role_tag_attribute[tag]
326     else
327       a= -1
328     end
329     return a
330   end
331
332 ltx.__tag.func.get_num_from = __tag_get_num_from
333
334 function ltx.__tag.func.output_num_from (tag)
335   local num = __tag_get_num_from (tag)
336   tex.sprint(catlatex,num)
337   if num == -1 then
338     __tag_log ("Unknown tag "..tag.." used")
339   end
340 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.

```

341 local __tag_get_tag_from =
342   function (num)
343     if ltx.__tag.tables.role_attribute_tag[num] then
344       a = ltx.__tag.tables.role_attribute_tag[num]
345     else
346       a= "UNKNOWN"
347     end
348     return a
349   end
350
351 ltx.__tag.func.get_tag_from = __tag_get_tag_from
352
353 function ltx.__tag.func.output_tag_from (num)
354   tex.sprint(catlatex,__tag_get_tag_from (num))
355 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.

```

356 function ltx.__tag.func.store_mc_data (num,key,data)
357   ltx.__tag.mc[num] = ltx.__tag.mc[num] or { }
358   ltx.__tag.mc[num][key] = data
359   __tag_log ("INFO TEX-STORE-MC-DATA: "..num.." => "..tostring(key).." => "..tostring(data),3)
360 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.

```

361 function ltx.__tag.func.store_mc_label (label,num)
362   ltx.__tag.mc["labels"] = ltx.__tag.mc["labels"] or { }
363   ltx.__tag.mc.labels[label] = num
364 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.

```

365 function ltx.__tag.func.store_mc_kid (mcnum,kid,page)
366   __tag_log("INFO TAG-STORE-MC-KID: "..mcnum.." => " .. kid.." on page " .. page,3)
367   ltx.__tag.mc[mcnum]["kids"] = ltx.__tag.mc[mcnum]["kids"] or { }
368   local kidtable = {kid=kid,page=page}
369   tableinsert(ltx.__tag.mc[mcnum]["kids"], kidtable )
370 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.

```

371 function ltx.__tag.func.mc_num_of_kids (mcnum)
372   local num = 0
373   if ltx.__tag.mc[mcnum] and ltx.__tag.mc[mcnum]["kids"] then
374     num = #ltx.__tag.mc[mcnum]["kids"]
375   end
376   __tag_log ("INFO MC-KID-NUMBERS: " .. mcnum .. "has " .. num .. "KIDS",4)
377   return num
378 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

```

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

```

```

396     local emcnode = nodenew("whatsit","pdf_literal")
397     emcnode.data = "EMC"
398     emcnode.mode=1
399     return emcnode
400 end
401 end
402
403 local function __tag_insert_emc_node (head,current)
404     local emcnode= __tag_backend_create_emc_node()
405     head = node.insert_before(head,current,emcnode)
406     return head
407 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

```

408 local __tag_backend_create_bmc_node
409 if tex.outputmode == 0 then
410     if token.get_macro("c_sys_backend_str") == "dvipdfmx" then
411         function __tag_backend_create_bmc_node (tag)
412             local bmcnode = nodenew("whatsit","special")
413             bmcnode.data = "pdf:code /"..tag.." BMC"
414             return bmcnode
415         end
416     else -- assume a dvips variant
417         function __tag_backend_create_bmc_node (tag)
418             local bmcnode = nodenew("whatsit","special")
419             bmcnode.data = "ps:SDict begin mark/"..tag.." /BMC pdfmark end"
420             return bmcnode
421         end
422     end
423 else -- pdf mode
424     function __tag_backend_create_bmc_node (tag)
425         local bmcnode = nodenew("whatsit","pdf_literal")
426         bmcnode.data = "/"..tag.." BMC"
427         bmcnode.mode=1
428         return bmcnode
429     end
430 end
431
432 local function __tag_insert_bmc_node (head,current,tag)
433     local bmcnode = __tag_backend_create_bmc_node (tag)
434     head = node.insert_before(head,current,bmcnode)
435     return head
436 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.

```

437 local __tag_backend_create_bdc_node
438
439 if tex.outputmode == 0 then
440     if token.get_macro("c_sys_backend_str") == "dvipdfmx" then
441         function __tag_backend_create_bdc_node (tag,dict)

```

```

442     local bdcnode = nodenew("whatsit","special")
443     bdcnode.data = "pdf:code /"..tag.."<<"..dict..">> BDC"
444     return bdcnode
445 end
446 else -- assume a dvips variant
447     function __tag_backend_create_bdc_node (tag,dict)
448         local bdcnode = nodenew("whatsit","special")
449         bdcnode.data = "ps:SDict begin mark/"..tag.."<<"..dict..">> /BDC pdfmark end"
450         return bdcnode
451     end
452 end
453 else -- pdf mode
454     function __tag_backend_create_bdc_node (tag,dict)
455         local bdcnode = nodenew("whatsit","pdf_literal")
456         bdcnode.data = "/"..tag.."<<"..dict..">> BDC"
457         bdcnode.mode=1
458         return bdcnode
459     end
460 end
461
462 local function __tag_insert_bdc_node (head,current,tag,dict)
463     bdcnode= __tag_backend_create_bdc_node (tag,dict)
464     head = node.insert_before(head,current,bdcnode)
465     return head
466 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.

```

467 local function __tag_pdf_object_ref (name,index)
468     local object
469     if ltx.pdf.object_id then
470         object = ltx.pdf.object_id (name,index) .. ' 0 R'
471     else
472         local tokenname = 'c__pdf_object_'..name..'/'..index..'__int'
473         object = token.create(tokenname).mode .. ' 0 R'
474     end
475     return object
476 end
477 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.

```

478 local function __tag_show_spacemark (head,current,color,height)
479     local markcolor = color or "1 0 0"
480     local markheight = height or 10
481     local pdfstring
482     if tex.outputmode == 0 then
483         -- ignore dvi mode for now

```

```

484 else
485   pdfstring = node.new("whatsit", "pdf_literal")
486   pdfstring.data =
487     string.format("q ..markcolor.." RG "..markcolor.." rg 0.4 w 0 %g m 0 %g l S Q",-
3,markheight)
488   head = node.insert_after(head,current,pdfstring)
489   return head
490 end
491 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
492 local function __tag_fakespace()
493   tex.setattribute(iwspaceattributeid,1)
494   tex.setattribute(iwfontattributeid,font.current())
495 end
496 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.

```

497 --[[ a function to mark up places where real space chars should be inserted
498       it only sets an attribute.
499 --]]
500
501 local function __tag_mark_spaces (head)
502   local inside_math = false
503   for n in nodetraverse(head) do
504     local id = n.id
505     if id == GLYPH then
506       local glyph = n
507       default_currfontid = glyph.font
508       if glyph.next and (glyph.next.id == GLUE)
509         and not inside_math and (glyph.next.width > 0)
510       then
511         nodesetattribute(glyph.next,iwspaceattributeid,1)
512         nodesetattribute(glyph.next,iwfontattributeid,glyph.font)
513         -- for debugging
514         if ltx.__tag.trace.showspace then
515           __tag_show_spacemark (head,glyph)
516         end
517       elseif glyph.next and (glyph.next.id==KERN) and not inside_math then
518         local kern = glyph.next
519         if kern.next and (kern.next.id== GLUE) and (kern.next.width > 0)
520         -- the attribute is also set on the kern in case the kern+glue is
521         -- discarded at a line break tagging issue #1102
522         -- TODO iterate back through all discardable nodes.
523       then
524         nodesetattribute(kern,iwspaceattributeid,1)
525         nodesetattribute(kern,iwfontattributeid,glyph.font)
526         nodesetattribute(kern.next,iwspaceattributeid,1)
527         nodesetattribute(kern.next,iwfontattributeid,glyph.font)

```

```

528     end
529   end
530   -- look also back
531   if glyph.prev and (glyph.prev.id == GLUE)
532     and not inside_math
533     and (glyph.prev.width > 0)
534     and not nodehasattribute(glyph.prev,iwspaceattributeid)
535   then
536     nodesetattribute(glyph.prev,iwspaceattributeid,1)
537     nodesetattribute(glyph.prev,iwfontattributeid,glyph.font)
538   -- for debugging
539   if ltx.__tag.trace.showspace then
540     __tag_show_spacemark (head,glyph)
541   end
542   end
543   elseif id == PENALTY then
544     local glyph = n
545     -- __tag_log ("PENALTY ".. n.subtype.."VALUE"..n.penalty,3)
546     if glyph.next and (glyph.next.id == GLUE)
547       and not inside_math and (glyph.next.width > 0) and n.subtype==0
548     then
549       nodesetattribute(glyph.next,iwspaceattributeid,1)
550       -- changed 2024-01-18, issue #72
551       nodesetattribute(glyph.next,iwfontattributeid,default_currfontid)
552     -- for debugging
553     if ltx.__tag.trace.showspace then
554       __tag_show_spacemark (head,glyph)
555     end
556     end
557   elseif id == MATH then
558     inside_math = (n.subtype == 0)
559   end
560   end
561   return head
562 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
563 local function __tag_activate_mark_space ()
564   if not luatexbase.in_callback ("pre_linebreak_filter","markspaces") then
565     luatexbase.add_to_callback("pre_linebreak_filter",__tag_mark_spaces,"markspaces")
566     luatexbase.add_to_callback("hpack_filter",__tag_mark_spaces,"markspaces")
567   end
568 end
569
570 ltx.__tag.func.markspaceon=__tag_activate_mark_space
571
572 local function __tag_deactivate_mark_space ()
573   if luatexbase.in_callback ("pre_linebreak_filter","markspaces") then
574     luatexbase.remove_from_callback("pre_linebreak_filter","markspaces")
575     luatexbase.remove_from_callback("hpack_filter","markspaces")
576   end
577 end

```

```

578
579 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.

```

580 local default_space_char = nodenew(GLYPH)
581 local default_fontid      = fontid("TU/lmr/m/n/10")
582 local default_currfontid = fontid("TU/lmr/m/n/10")
583 default_space_char.char   = 32
584 default_space_char.font   = default_fontid

```

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

```

585 local function __tag_font_has_space (fontid)
586   t= fonts.hashes.identifiers[fontid]
587   if luaotfload.aux.slot_of_name(fontid,"space")
588     or t and t.characters and t.characters[32] and t.characters[32]["unicode"]==32
589   then
590     return true
591   else
592     return false
593   end
594 end

```

```

__tag_space_chars_shipout
ltx.__tag.func.space_chars_shipout

```

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

```

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

```



```

623         n.width      = n.width - space.width
624         space.attr    = n.attr
625     end
626 end
627 end
628 box.head = head
629 end
630 end
631
632 function ltx.__tag.func.space_chars_shipout (box)
633     __tag_space_chars_shipout (box)
634 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.

```

635 function ltx.__tag.func.mc_insert_kids (mcnum,single)
636     if ltx.__tag.mc[mcnum] then
637         __tag_log("INFO TEX-MC-INSERT-KID-TEST: " .. mcnum,4)
638         if ltx.__tag.mc[mcnum]["kids"] then
639             if #ltx.__tag.mc[mcnum]["kids"] > 1 and single==1 then
640                 tex.sprint(catlatex,"")
641             end
642             for i,kidstable in ipairs( ltx.__tag.mc[mcnum]["kids"] ) do
643                 local kidnum = kidstable["kid"]
644                 local kidpage = kidstable["page"]
645                 local kidpageobjnum = pdfpageref(kidpage)
646                 __tag_log("INFO TEX-MC-INSERT-KID: " .. mcnum ..
647                     " insert KID " .. i ..
648                     " with num " .. kidnum ..
649                     " on page " .. kidpage.."/"..kidpageobjnum,3)
650                 tex.sprint(catlatex,"<</Type /MCR /Pg " .. kidpageobjnum .. " 0 R /MCID " .. kidnum .. ">> ")
651             end
652             if #ltx.__tag.mc[mcnum]["kids"] > 1 and single==1 then
653                 tex.sprint(catlatex,"")
654             end
655         else
656             -- this is typically not a problem, e.g. empty hbox in footer/header can
657             -- trigger this warning.
658             __tag_log("WARN TEX-MC-INSERT-NO-KIDS: " .. mcnum .. " has no kids",2)
659             if single==1 then
660                 tex.sprint(catlatex,"null")
661             end
662         end
663     else
664         __tag_log("WARN TEX-MC-INSERT-MISSING: " .. mcnum .. " doesn't exist",0)
665     end
666 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.

```

667 function ltx.__tag.func.store_struct_mcabs (structnum,mcnum)
668   ltx.__tag.struct[structnum]=ltx.__tag.struct[structnum] or { }
669   ltx.__tag.struct[structnum]["mc"]=ltx.__tag.struct[structnum]["mc"] or { }
670   -- a structure can contain more than on mc chunk, the content should be ordered
671   tableinsert(ltx.__tag.struct[structnum]["mc"],mcnum)
672   __tag_log("INFO TEX-MC-INTO-STRUCT: "..
673             mcnum.." inserted in struct "..structnum,3)
674   -- but every mc can only be in one structure
675   ltx.__tag.mc[mcnum]= ltx.__tag.mc[mcnum] or { }
676   ltx.__tag.mc[mcnum]["parent"] = structnum
677 end
678

```

(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.

```

679 -- pay attention: lua counts arrays from 1, tex pages from one
680 -- mcid and arrays in pdf count from 0.
681 function ltx.__tag.func.store_mc_in_page (mcnum,mcpagecnt,page)
682   ltx.__tag.page[page] = ltx.__tag.page[page] or {}
683   ltx.__tag.page[page][mcpagecnt] = mcnum
684   __tag_log("INFO TAG-MC-INTO-PAGE: page " .. page ..
685             ": inserting MCID " .. mcpagecnt .. " => " .. mcnum,3)
686 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)

```

687 local function __tag_update_mc_attributes (head,mcnum,type)
688   for n in node.traverse(head) do
689     node.set_attribute(n,mccntattributeid,mcnum)
690     node.set_attribute(n,mctypeattributeid,type)
691     if n.id == HLIST or n.id == VLIST then
692       __tag_update_mc_attributes (n.list,mcnum,type)
693     end
694   end
695   return head
696 end
697 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.

```

698 --[[
699   Now follows the core function
700   It wades through the shipout box and checks the attributes
701   ARGUMENTS
702   box: is a box,
703   mcpagecnt: num, the current page cnt of mc (should start at -1 in shipout box), needed for

```

```

704     mcntprev: num, the attribute cnt of the previous node/whatever - if different we have a
705     mcopy: num, records if some bdc/emc is open
706     These arguments are only needed for log messages, if not present are replaced by fix strings
707     name: string to describe the box
708     mctypeprev: num, the type attribute of the previous node/whatever
709
710     there are lots of logging messages currently. Should be cleaned up in due course.
711     One should also find ways to make the function shorter.
712 --]]
713
714 function ltx.__tag.func.mark_page_elements (box,mcpagencnt,mcntprev,mcopy,name,mctypeprev)
715     local name = name or ("SOMEBOX")
716     local mctypeprev = mctypeprev or -1
717     local abspage = status.total_pages + 1 -- the real counter is increased
718                                           -- inside the box so one off
719                                           -- if the callback is not used. (???)
720     __tag_log ("INFO TAG-ABSPAGE: " .. abspage,3)
721     __tag_log ("INFO TAG-ARGS: pagencnt".. mcpagencnt..
722               " prev "..mcntprev ..
723               " type prev "..mctypeprev,4)
724     __tag_log ("INFO TAG-TRAVERSING-BOX: ".. tostring(name)..
725               " TYPE ".. node.type(node.getid(box)),3)
726     local head = box.head -- ShipoutBox is a vlist?
727     if head then
728         mcnthead, mctypehead,taghead = __tag_get_mc_cnt_type_tag (head)
729         __tag_log ("INFO TAG-HEAD: " ..
730                   node.type(node.getid(head))..
731                   " MC"..tostring(mcnthead)..
732                   " => TAG " .. tostring(mctypehead)..
733                   " => ".. tostring(taghead),3)
734     else
735         __tag_log ("INFO TAG-NO-HEAD: head is "..
736                   tostring(head),3)
737     end
738     for n in node.traverse(head) do
739         local mcnt, mctype, tag = __tag_get_mc_cnt_type_tag (n)
740         local spaceattr = nodegetattribute(n,iwspaceattributeid) or -1
741         __tag_log ("INFO TAG-NODE: "..
742                   node.type(node.getid(n))..
743                   " MC".. tostring(mcnt)..
744                   " => TAG " .. tostring(mctype)..
745                   " => " .. tostring(tag),3)
746         if n.id == HLIST
747         then -- enter the hlist
748             mcopy,mcpagencnt,mcntprev,mctypeprev=
749             ltx.__tag.func.mark_page_elements (n,mcpagencnt,mcntprev,mcopy,"INTERNAL HLIST",mctypeprev)
750         elseif n.id == VLIST then -- enter the vlist
751             mcopy,mcpagencnt,mcntprev,mctypeprev=
752             ltx.__tag.func.mark_page_elements (n,mcpagencnt,mcntprev,mcopy,"INTERNAL VLIST",mctypeprev)
753         elseif n.id == GLUE and not n.leader then -- at glue real space chars are inserted, but t
754                                           -- been done if the previous shipout wandering, so here it
755         elseif n.id == LOCAL_PAR then -- local_par is ignored
756         elseif n.id == PENALTY then -- penalty is ignored
757         elseif n.id == KERN then -- kern is ignored

```

```

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

```

```

812         tostring(ltx.__tag.mc[mccnt]["lang"]),3)
813     dict= dict .. " " .. ltx.__tag.mc[mccnt]["lang"]
814 end
815 if ltx.__tag.mc[mccnt]["actualtext"] then
816     __tag_log("INFO TAG-USE-ACTUALTEXT: "..
817         tostring(ltx.__tag.mc[mccnt]["actualtext"]),3)
818     dict= dict .. " " .. ltx.__tag.mc[mccnt]["actualtext"]
819 end
820 box.list = __tag_insert_bdc_node (box.list,n,tag, dict)
821 ltx.__tag.func.store_mc_kid (mccnt,mcpagecnt,abspage)
822 ltx.__tag.func.store_mc_in_page(mccnt,mcpagecnt,abspage)
823 ltx.__tag.trace.show_mc_data (mccnt,3)
824 end
825 mcopen = mcopen + 1
826 else
827     if tagunmarkedbool.mode == truebool.mode then
828         __tag_log("INFO TAG-NOT-TAGGED: this has not been tagged, using artifact",2)
829         box.list = __tag_insert_bmc_node (box.list,n,"Artifact")
830         mcopen = mcopen + 1
831     else
832         __tag_log("WARN TAG-NOT-TAGGED: this has not been tagged",1)
833     end
834 end
835 mcntprev = mccnt
836 end
837 end -- end if
838 end -- end for
839 if head then
840     mcntthead, mctypehead,taghead = __tag_get_mc_cnt_type_tag (head)
841     __tag_log ("INFO TAG-ENDHEAD: " ..
842         node.type(node.getid(head))..
843         " MC"..tostring(mcntthead)..
844         " => TAG "..tostring(mctypehead)..
845         " => "..tostring(taghead),4)
846 else
847     __tag_log ("INFO TAG-ENDHEAD: ".. tostring(head),4)
848 end
849 __tag_log ("INFO TAG-QUITTING-BOX "..
850     tostring(name)..
851     " TYPE ".. node.type(node.getid(box)),4)
852 return mcopen,mcpagecnt,mcntprev,mctypeprev
853 end
854
(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.

```

855 function ltx.__tag.func.mark_shipout (box)
856     mcopen = ltx.__tag.func.mark_page_elements (box,-1,-100,0,"Shipout",-1)
857     if mcopen~=0 then -- there is a chunk open, close it (hope there is only one ...
858         local emcnode = __tag_backend_create_emc_node ()
859         local list = box.list
860         if list then

```

```

861     list = node.insert_after (list,node.tail(list),emcnode)
862     mcopen = mcopen - 1
863     __tag_log ("INFO SHIPOUT-INSERT-LAST-EMC: MCOPEN " .. mcopen,3)
864 else
865     __tag_log ("WARN SHIPOUT-UPS: this shouldn't happen",0)
866 end
867 if mcopen ~=0 then
868     __tag_log ("WARN SHIPOUT-MC-OPEN: " .. mcopen,1)
869 end
870 end
871 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.

```

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

```

```

908     numsentry = numsentry .. "]" "
909     __tag_log("INFO PARENTTREE-NUMENTRY: page " ..
910     page.. " num entry = ".. numsentry,3)
911     end
912 else
913     __tag_log ("INFO PARENTTREE-NO-DATA: page "..page,3)
914     numsentry = pdfpage.." ["
915     end
916     return numsentry
917 end
918
919 function ltx.__tag.func.output_parenttree (abspage)
920 for i=1,abspage do
921     line = ltx.__tag.func.fill_parent_tree_line (i) .. "^^J"
922     tex.sprint(catlatex,line)
923 end
924 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.

```

925 do
926     local properties = node.get_properties_table()
927     local is_soft_hyphen_prop = 'tagpdf.rewrite-softhyphen.is_soft_hyphen'
928     local hyphen_char = 0x2D
929     local soft_hyphen_char = 0xAD

```

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

```

930     local softhyphen_fonts = setmetatable({}, {__index = function(t, fid)
931         local fdir = identifiers[fid]
932         local format = fdir and fdir.format
933         local result = (format == 'opentype' or format == 'truetype')
934         local characters = fdir and fdir.characters
935         result = result and (characters and characters[soft_hyphen_char]) ~= nil
936         t[fid] = result
937         return result
938     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.

```

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

```

```

953     return true
954 end
955

```

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

```

956 local function process_softhyphen_post(head, _context, _dir)
957   if softhyphenbool.mode ~= truebool.mode then return true end
958   for disc, sub in node.traverse_id(DISC, head) do
959     for n, ch, fid in node.traverse_glyph(disc.pre) do
960       local props = properties[n]
961       if softhyphen_fonts[fid] and ch == hyphen_char and props and props[is_soft_hyphen_prop]
962         if nodegetattribute(n,softhyphenattribute) then
963           n.char = soft_hyphen_char
964           props.glyph_info = nil
965         else
966           nodesetattribute(n,mctypeattributeid,-2147483647)
967           nodesetattribute(n,mccntattributeid,-2147483647)
968         end
969       end
970     end
971   end
972   return true
973 end
974
975 luatexbase.add_to_callback('pre_shaping_filter', process_softhyphen_pre, 'tagpdf.rewrite-
softhyphen')
976 luatexbase.add_to_callback('post_shaping_filter', process_softhyphen_post, 'tagpdf.rewrite-
softhyphen')
977 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

```

978 local function role_get_parent_child_rule (parent,child)
979   local state=
980   ltx.__tag.role.matrix[ltx.__tag.role.index[parent]]
981   and ltx.__tag.role.matrix[ltx.__tag.role.index[parent]][ltx.__tag.role.index[child]] or 0
982   return state
983 end
984 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.

```

985 function check_update_stashed (struct,loglevel,loop)

```



```

986 loop = (loop or 0) + 1
987 if loop > 10 then
988   __tag_log ('Warning: Too deeply nested stashed structures',0)
989   return
990 end
991 __tag_log ('updating parentrole for stashed structure '..struct,loglevel)
992 local parent = ltx.__tag.tables['g__tag_struct_ '..struct..'__prop']['parentnum']
993 if parent then
994   local ptag =
995     string.match(ltx.__tag.tables['g__tag_struct_ '..parent..'__prop']['parentrole'], "{(.-)}{(.-)}{(.-)}")
996   if ptag == 'STASHED' then
997     -- look at the parent and update it first
998     check_update_stashed (parent,loglevel,loop)
999   end
1000   -- now copy the parent role from the parent
1001   ltx.__tag.tables['g__tag_struct_ '..struct..'__prop']['parentrole']
1002   =
1003   ltx.__tag.tables['g__tag_struct_ '..parent..'__prop']['parentrole']
1004   __tag_log
1005   ('new parentrole: ' .. ltx.__tag.tables['g__tag_struct_ '..struct..'__prop']['parentrole'],0)
1006 else
1007   __tag_log ('Warning: structure '..struct..' has no parent.',0)
1008 end
1009 end
1010
1011 function check_parent_child_rules (loglevel)
1012   texio.write_nl('\n')
1013   __tag_log ('checking parent-child rules ...' ,0)
1014   for i=2,ltx.tag.get_struct_counter() do
1015     local t,tNS=
1016     string.match(ltx.__tag.tables['g__tag_struct_ '..i..'__prop']['tag'], "{(.-)}{(.-)}{(.-)}")
1017     local r,rNS=
1018     string.match(ltx.__tag.tables['g__tag_struct_ '..i..'__prop']['rolemap'], "{(.-)}{(.-)}{(.-)}")
1019     local p,pNS=
1020     string.match(ltx.__tag.tables['g__tag_struct_ '..i..'__prop']['parentrole'], "{(.-)}{(.-)}{(.-)}")
1021     local parent=ltx.__tag.tables['g__tag_struct_ '..i..'__prop']['parentnum']
1022     if parent then
1023       __tag_log (i..'': '.. t..'': '..tNS,loglevel)
1024       __tag_log (i..'': '.. r..'': '..rNS,loglevel)
1025       __tag_log (i..'': '.. p..'': '..pNS,loglevel)
1026       __tag_log ('parent of ' ..i..'': '.. parent,loglevel )
1027       if p == 'STASHED' then
1028         check_update_stashed (i,loglevel,0)
1029       p,pNS=
1030       string.match(ltx.__tag.tables['g__tag_struct_ '..i..'__prop']['parentrole'], "{(.-)}{(.-)}{(.-)}")
1031     end
1032     local pt,ptNS=
1033     string.match(ltx.__tag.tables['g__tag_struct_ '..parent..'__prop']['tag'], "{(.-)}{(.-)}{(.-)}")

```

```

1034     local pr,prNS=
1035     string.match(ltx.__tag.tables['g__tag_struct'..'parent..'__prop']['rolemap'], "{(.-
    )}{(.-)}")
1036     local pp,ppNS=
1037     string.match(ltx.__tag.tables['g__tag_struct'..'parent..'__prop']['parentrole'], "{(.-
    )}{(.-)}")
1038     if pp == 'STASHED' then
1039         check_update_stashed (parent,loglevel,0)
1040         pp,ppNS=
1041         string.match(ltx.__tag.tables['g__tag_struct'..'parent..'__prop']['parentrole'], "{(.-
    )}{(.-)}")
1042     end
1043     __tag_log (parent..'': '.. pt..'': '..ptNS,loglevel)
1044     __tag_log (parent..'': '.. pr..'': '..prNS,loglevel)
1045     __tag_log (parent..'': '.. pp..'': '..ppNS,loglevel)
1046     -- now check the rule.
1047     -- at first rolemap of child against rolemap of parent.
1048     local state=ltx.__tag.func.role_get_parent_child_rule (pr,r)
1049     __tag_log ('rule of '..pr..' -> '..r..' is '..state,loglevel)
1050     -- if the state is 7 we check against parentrole of the parent
1051     if state == 7 then
1052         state=ltx.__tag.func.role_get_parent_child_rule (pp,r)
1053         __tag_log ('Parent-Child relation '..pp..' -> '..r..' is '..state,loglevel)
1054     end
1055     if state == 0 then
1056         __tag_log
1057         ('Warning: Parent-Child relation '
1058         '..ptNS..'': '..pt..' -> '..tNS..'': '..t..' is unknown',0)
1059         __tag_log
1060         ('Structure ' ..parent..' -> '..i,0)
1061     end
1062     if state == -1 then
1063         __tag_log
1064         ('Warning: Parent-Child relation '
1065         '..ptNS..'': '..pt..' -> '..tNS..'': '..t..' is not allowed',0)
1066         __tag_log
1067         ('Structure ' ..parent..' -> '..i,0)
1068     end
1069     -- check also for MC
1070     state =ltx.__tag.func.role_get_parent_child_rule ( r ,'MC')
1071     local curtag=r
1072     if state == 7 then
1073         state =ltx.__tag.func.role_get_parent_child_rule ( p ,'MC')
1074         local curtag=p
1075     end
1076     if state == -1 then
1077         if ltx.__tag.struct[i] and NEXT(ltx.__tag.struct[i]) then
1078             __tag_log
1079             ('Warning: Real content (MC) is not allowed in ' ..curtag,0)
1080         end
1081     end
1082     __tag_log('=====',loglevel)
1083 end
1084 end -- end for

```

```

1085 end
1086
1087 ltx.__tag.func.check_parent_child_rules=check_parent_child_rules
1088
(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.

```

1089 if luatexbase.callbacktypes['linksplit'] then
1090   luatexbase.add_to_callback('linksplit', function(start_link, position)
1091     if start_link == nil then return end
1092     local structnum =
1093       node.get_attribute(start_link,luatexbase.attributes.g__tag_structnum_attr)
1094     if structnum and structnum > -1 then
1095       local s = ltx.__tag.tables['g__tag_struct_'..structnum..'__prop']['rolemap']
1096       if s and (string.find(s,'Link') or string.find(s,'Reference')) then
1097         local struct_insert_annot_shipout = token.create'__tag_struct_insert_annot_shipout'
1098         local parentnum = tex.count['c@g__tag_parenttree_obj_int']
1099         start_link.link_attr =
1100           start_link.link_attr ..
1101           ' /LTEX_position /' .. position ..
1102           '/StructParent ' .. parentnum
1103         tex.sprint(catlatex,struct_insert_annot_shipout,'{'..
1104           structnum..''}{'..
1105           start_link.objnum..' 0 R'}{'..
1106           parentnum ..'}')
1107         -- the counter must be set explicitly as struct_insert_annot_shipout doesn't do it
1108         tex.setcount('global','c@g__tag_parenttree_obj_int',parentnum +1)
1109         __tag_log(position .. " link part has object id " .. start_link.objnum .. " and s
1110       else
1111         __tag_log('Warning: Link not in Link or Reference structure element',0)
1112         __tag_log('OBJR not created',0)
1113         __tag_log('',0)
1114       end
1115     end
1116   end, 'tagpdf')
1117 end
1118 </lua>

```

The tagpdf-roles module

Tags, roles and namespace code

Part of the tagpdf package

Ulrike Fischer

Version 0.99y, released 2026-01-29

Part X

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

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

The key-value list knows the following keys:

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

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

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

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

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

`\tag_check_child:nnTF` `\tag_check_child:nnTF {<tag>} {<namespace>} {<true code>} {<false code>}`

This checks if the tag `<tag>` from the name space `<namespace>` can be used at the current position. In tagpdf-base it is always true.

```
1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-roles-code} {2026-01-29} {0.99y}
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/pdf/ssn>

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

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

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

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

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

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_tmpa_seq`

¹⁶ `\tl_new:N \l__tag_role_role_namespace_tmpa_tl`

¹⁷ `\seq_new:N \l__tag_role_tmpa_seq`

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

1.2 Namespaces

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

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

```
18 \pdfdict_new:n {g__tag_role/RoleMap_dict}
19 \__tag_prop_new:N \g__tag_role_rolemap_prop
(End of definition for g__tag_role/RoleMap_dict and \g__tag_role_rolemap_prop.)
```

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

```
\__tag_role_NS_new:nnn
```

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

```

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

```

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

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

`\c__tag_role_userNS_id_str`

```

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

```

(End of definition for `\c__tag_role_userNS_id_str`.)

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

```

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

```

1.3 Adding a new tag

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

`__tag_role_alloctag:nnn`

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

```

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

```



```

84 {
85   \cs_new_protected:Npn \__tag_role_alloctag:nnn #1 #2 #3 % #1 tagname, ns, type
86   {
87     \lua_now:e { ltx.__tag.func.alloctag ('#1') }
88     \prop_gput:Nnn \g__tag_role_tags_NS_prop {#1}{#2}
89     \__tag_prop_gput:cnn {g__tag_role_NS_#2_prop} {#1}{{}{}}
90     \prop_gput:Nnn \g__tag_role_tags_class_prop {#1}{#3}
91     \prop_gput:cnn {g__tag_role_NS_#2_class_prop} {#1}{--UNUSED--}
92   }
93 }
94 {
95   \cs_new_protected:Npn \__tag_role_alloctag:nnn #1 #2 #3
96   {
97     \prop_gput:Nnn \g__tag_role_tags_NS_prop {#1}{#2}
98     \__tag_prop_gput:cnn {g__tag_role_NS_#2_prop} {#1}{{}{}}
99     \prop_gput:Nnn \g__tag_role_tags_class_prop {#1}{#3}
100    \prop_gput:cnn {g__tag_role_NS_#2_class_prop} {#1}{--UNUSED--}
101  }
102 }
103 }
104 {
105   \sys_if_engine luatex:TF
106   {
107     \cs_new_protected:Npn \__tag_role_alloctag:nnn #1 #2 #3 % #1 tagname, ns, type
108     {
109       \lua_now:e { ltx.__tag.func.alloctag ('#1') }
110       \prop_gput:Nnn \g__tag_role_tags_NS_prop {#1}{#2}
111       \__tag_prop_gput:cnn {g__tag_role_NS_#2_prop} {#1}{{}{}}
112       \prop_gput:Nnn \g__tag_role_tags_class_prop {#1}{--UNUSED--}
113       \prop_gput:cnn {g__tag_role_NS_#2_class_prop} {#1}{#3}
114     }
115   }
116   {
117     \cs_new_protected:Npn \__tag_role_alloctag:nnn #1 #2 #3
118     {
119       \prop_gput:Nnn \g__tag_role_tags_NS_prop {#1}{#2}
120       \__tag_prop_gput:cnn {g__tag_role_NS_#2_prop} {#1}{{}{}}
121       \prop_gput:Nnn \g__tag_role_tags_class_prop {#1}{--UNUSED--}
122       \prop_gput:cnn {g__tag_role_NS_#2_class_prop} {#1}{#3}
123     }
124   }
125 }
126 \cs_generate_variant:Nn \__tag_role_alloctag:nnn {nno}

```

(End of definition for __tag_role_alloctag:nnn.)

1.3.1 pdf 1.7 and earlier

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

```

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

```

checks and messages

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

now the addition

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

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

```
142   \tl_if_empty:nF { #2 }
143   {
144     \prop_get:NnNTF \g__tag_role_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:Npn \__tag_role_get:nnNN #1#2#3#4 {%#1 tag, #2 NS, #3 tlvar which hold the role tag
157   {
158     \prop_get:NnNF \g__tag_role_rolemap_prop {#1}#3
159     {
160       \tl_set:Nn #3 {#1}
161     }
162     \tl_set:Nn #4 {}
163   }
164   \cs_generate_variant:Nn \__tag_role_get:nnNN {ooNN}
165 }
166
(End of definition for \__tag_role_get:nnNN.)
```

1.3.2 The pdf 2.0 version

_tag_role_add_tag:nnnn

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

```

167 \cs_new_protected:Nn \_tag_role_add_tag:nnnn %tag/namespace/role/namespace
168 {
169   \_tag_check_add_tag_role:nnn {#1/#2}{#3}{#4}
170   \int_compare:nNnT {\_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}\_tag_tmpa_tl
178     \quark_if_no_value:NT \_tag_tmpa_tl
179     {
180       \tl_set:Nn\_tag_tmpa_tl{--UNKNOWN--}
181     }
182   }
183   { \tl_set:Nn\_tag_tmpa_tl{--UNKNOWN--} }
184   \_tag_role_alloctag:nno {#1}{#2}{ \_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}\_tag_tmpa_tl
199     \quark_if_no_value:NTF \_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}{\_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 \_tag_role_update_bool
209   {
210     \tl_if_empty:nF { #3 }

```

```

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

```

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

```

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

```

1.4 Helper command to read the data from files

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

`__tag_role_read_namespace_line:nw`

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

```

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

```

```

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

```

(End of definition for __tag_role_read_namespace_line:nw.)

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

```

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

```

(End of definition for __tag_role_read_namespace:nn.)

__tag_role_read_namespace:n This command reads the default namespace file.

```

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

```

(End of definition for __tag_role_read_namespace:n.)

1.5 Reading the default data

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

```

341 \__tag_role_read_namespace:n {pdf}

```

```

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

```

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

```

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

```

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

```

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

```

1.6 Parent-child rules

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

`\g__tag_role_parent_child_intarray`

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

```
390 \intarray_new:Nn \g__tag_role_parent_child_intarray {6000}
```

(End of definition for `\g__tag_role_parent_child_intarray`.)

`\c__tag_role_rules_prop`

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

`\c__tag_role_rules_num_prop`

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

`__tag_store_parent_child_rule:nnn`

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

```
391 \sys_if_engine luatex:TF
```

```
392 {
```

```
393 \cs_new_protected:Npn \__tag_store_parent_child_rule:nnn #1 #2 #3 % num parent, num child,
```

```
394 {
```

```
395 \prop_get:NeNTF \c__tag_role_rules_prop{#3} \l__tag_tmp_unused_tl
```

```
396 {
```

```
397 \intarray_gset:Nnn \g__tag_role_parent_child_intarray
```

```
398 { #1#2 }{0\l__tag_tmp_unused_tl}
```

```
399 \lua_now:e
```

```
400 {
```

```
401 ltx.__tag.role.matrix[#1] = ltx.__tag.role.matrix[#1] or {}
```

```
402 ltx.__tag.role.matrix[#1][#2] = 0\l__tag_tmp_unused_tl
```

```
403 }
```

```
404 }
```

```
405 {
```

```
406 \intarray_gset:Nnn \g__tag_role_parent_child_intarray
```

```
407 { #1#2 }{0}
```

```
408 \lua_now:e
```

```
409 {
```

```
410 ltx.__tag.role.matrix[#1] = ltx.__tag.role.matrix[#1] or {}
```

```
411 ltx.__tag.role.matrix[#1][#2] = 0
```

```
412 }
```

```
413 }
```

```
414 }
```

```
415 }
```

```
416 {
```

```
417 \cs_new_protected:Npn \__tag_store_parent_child_rule:nnn #1 #2 #3 % num parent, num child,
```

```
418 {
```

```
419 \prop_get:NeNTF \c__tag_role_rules_prop{#3} \l__tag_tmp_unused_tl
```

```
420 {
```

```
421 \intarray_gset:Nnn \g__tag_role_parent_child_intarray
```

```
422 { #1#2 }{0\l__tag_tmp_unused_tl}
```

```
423 }
```

```
424 {
```

```
425 \intarray_gset:Nnn \g__tag_role_parent_child_intarray
```

```
426 { #1#2 }{0}
```

```
427 }
```

```
428 }
```

```
429 }
```


(End of definition for _tag_store_parent_child_rule:nnn.)

1.6.1 Reading in the csv-files

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

```
430 \int_zero:N \l__tag_tmpa_int
```

Open the file depending on the PDF version

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

Now the main loop over the file

```
438 \ior_map_inline:Nn \g_tmpa_ior
439 {
```

ignore lines containing only comments

```
440   \tl_if_empty:nF{#1}
441   {
```

count the lines ...

```
442     \int_incr:N\l__tag_tmpa_int
```

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

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

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

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

now the data lines.

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

get the name of the child tag from the first column

```
455       \seq_pop_left:NN\l__tag_tmpa_seq\l__tag_tmpa_tl
```

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

```
456       \prop_get:NoN \g__tag_role_index_prop { \l__tag_tmpa_tl } \l__tag_tmppb_tl
```

remove column 2+3

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

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

```

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

```

close the read handle.

```

467 \ior_close:N\g_tmpa_ior

```

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

```

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

```

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

```

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

```

1.6.2 Retrieving the parent-child rule

`__tag_role_get_parent_child_rule:nnN`

This command retrieves the rule (as a number) and stores it in the `tl`-var. It assumes that the tags in `#1` and `#2` are standard tags after role mapping for which a rule exist.

If the parent is one of Part, Div, NonStruct the result can be state 7, which means that a check must be repeated for the “real parent”.

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

```

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

```

Debugging messages, this can perhaps go into debug mode.

```

512   \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
513   {
514     \prop_get:NnF\c__tag_role_rules_num_prop {#3} \l__tag_tmpa_tl
515     {
516       \tl_set:Nn \l__tag_tmpa_tl {unknown}
517     }
518     \tl_set:Nn \l__tag_tmpb_tl {#1}
519     \msg_note:nneee
520     { tag }
521     { role-parent-child-result }
522     { #1 }
523     { #2 }
524     {
525       #3~(=\l__tag_tmpa_tl')
526     }
527   }
528   \int_compare:nNnT {#3} = { 0 }
529   {
530     \msg_warning:nneee
531     { tag }
532     {role-parent-child-result}
533     { #1 }
534     { #2 }
535     { unknown! }
536   }
537 }
538 }
539 }
540 {
541   \cs_new_protected:Npn \__tag_role_get_parent_child_rule:nnN #1 #2 #3
542     % #1 parent (string, standard tag after rolemapping)
543     % #2 child (string, standard tag after rolemapping)
544     % #3 tl for state
545   {
546     \prop_get:NnN \g__tag_role_index_prop{#1}\l__tag_tmpa_tl
547     \prop_get:NnN \g__tag_role_index_prop{#2}\l__tag_tmpb_tl
548     \bool_lazy_and:nnTF

```

```

549         { ! \quark_if_no_value_p:N \l__tag_tmpa_tl }
550         { ! \quark_if_no_value_p:N \l__tag_tmpb_tl }
551         {

```

Get the rule from the intarray

```

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

```

Debugging messages, this can perhaps go into debug mode.

```

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

```

(End of definition for __tag_role_get_parent_child_rule:nnN.)

__tag_role_check_parent_child:nnnnN

This command rolemaps its arguments and then calls __tag_role_get_parent-child_rule:nnN to retrieve the parent-child rule between both. It does not try to resolve inheritance rules of Part, Div and NonStruct but instead gives back the state 7. It is then the task of the caller command to find the real parent and run the check again. In pdf 2.0 the name spaces of the tags are relevant, so we have arguments for them, but in pdf <2.0 they are ignored and can be left empty.

```

590 \pdf_version_compare:NnTF < {2.0}

```

```

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

```

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

```

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

```

now the child

```

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

```

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

```

619     \bool_lazy_and:nnTF
620     { ! \quark_if_no_value_p:N \l__tag_tmpa_tl }
621     { ! \quark_if_no_value_p:N \l__tag_tmpb_tl }
622     {
623       \__tag_role_get_parent_child_rule:ooN
624       { \l__tag_tmpa_tl }
625       { \l__tag_tmpb_tl }
626       #5
627     }
628     {
629       \tl_set:Nn #5 {0}
630       \msg_warning:nneee
631       { tag }
632       {role-parent-child-result}
633       { #1 }
634       { #3 }
635       { unknown! }
636     }
637   }
638 }

```

and now the pdf 2.0 version

```

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

```

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

```

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

```

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

```

670   \tl_if_empty:nTF {#4}
671   {
672     \tl_set:Nn \l__tag_tmpb_tl {#3}
673   }
674   {
675     \prop_if_exist:cTF { g__tag_role_NS_#4_prop }
676     {
677       \prop_get:cnNTF
678       { g__tag_role_NS_#4_prop }
679       {#3}
680       \l__tag_tmpb_tl
681       {
682         \tl_set:Ne \l__tag_tmpb_tl { \tl_head:N\l__tag_tmpb_tl }
683         \tl_if_empty:NT\l__tag_tmpb_tl
684         {
685           \tl_set:Nn \l__tag_tmpb_tl {#3}

```

```

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

```

and now get the relation

```

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

```

(End of definition for __tag_role_check_parent_child:nnnnN.)

\tag_check_child:nnTF

```

719 <base>\prg_new_protected_conditional:Npnn \tag_check_child:nn #1 #2 {T,F,TF}{\prg_return_true}
720 <*package>
721 \prg_set_protected_conditional:Npnn \tag_check_child:nn #1 #2 {T,F,TF} {%#1 tag, #2 NS
722 {
723     \seq_get:NN\g__tag_struct_stack_seq\l__tag_tmpa_tl
724     \__tag_struct_get_role:enNN
725     {\l__tag_tmpa_tl}
726     {rolemap}
727     \l__tag_get_parent_tmpa_tl
728     \l__tag_get_parent_tmpb_tl
729     \__tag_role_check_parent_child:oonnN
730     { \l__tag_get_parent_tmpa_tl }
731     { \l__tag_get_parent_tmpb_tl }
732     {#1}{#2}
733     \l__tag_parent_child_check_tl
734     \int_compare:nNnT {\l__tag_parent_child_check_tl} = { \c__tag_role_rule_checkparent_tl }

```

```

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

```

(End of definition for \tag_check_child:nnTF. This function is documented on page 180.)

1.7 Key-val user interface

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

```

tag (rolemap-key)
tag-namespace (rolemap-key)
role (rolemap-key)
role-namespace (rolemap-key)
role/new-tag (setup-key)
add-new-tag (deprecated)
752 \keys_define:nn { __tag / tag-role }
753 {
754     ,tag .tl_set:N = \l__tag_role_tag_tmpa_tl
755     ,tag-namespace .tl_set:N = \l__tag_role_tag_namespace_tmpa_tl
756     ,role .tl_set:N = \l__tag_role_role_tmpa_tl
757     ,role-namespace .tl_set:N = \l__tag_role_role_namespace_tmpa_tl
758 }
759
760 \keys_define:nn { __tag / setup }
761 {
762     role/mathml-tags .bool_gset:N = \g__tag_role_add_mathml_bool
763     ,role/new-tag .code:n =
764     {
765         \keys_set_known:nnnN
766             {__tag/tag-role}
767             {
768                 tag-namespace=user,
769                 role-namespace=, %so that we can test for it.
770                 #1
771             }{__tag/tag-role}\l__tag_tmpa_tl
772         \tl_if_empty:NF \l__tag_tmpa_tl
773         {
774             \exp_args:NNno \seq_set_split:Nnn \l__tag_tmpa_seq { / } {\l__tag_tmpa_tl/}
775             \tl_set:Ne \l__tag_role_tag_tmpa_tl { \seq_item:Nn \l__tag_tmpa_seq {1} }
776             \tl_set:Ne \l__tag_role_role_tmpa_tl { \seq_item:Nn \l__tag_tmpa_seq {2} }
777         }
778         \tl_if_empty:NT \l__tag_role_role_namespace_tmpa_tl
779         {
780             \prop_get:NoNTF

```



```

781         \g__tag_role_tags_NS_prop
782         { \l__tag_role_role_tmpa_tl }
783         \l__tag_role_role_namespace_tmpa_tl
784         {
785             \prop_get:NnF
786             \g__tag_role_NS_prop
787             { \l__tag_role_role_namespace_tmpa_tl }
788             \l__tag_tmp_unused_tl
789             {
790                 \tl_set:Nn \l__tag_role_role_namespace_tmpa_tl {user}
791             }
792         }
793         {
794             \tl_set:Nn \l__tag_role_role_namespace_tmpa_tl {user}
795         }
796     }
797     \pdf_version_compare:NnTF < {2.0}
798     {
799         %TODO add check for emptiness?
800         \__tag_role_add_tag:oo
801         { \l__tag_role_tag_tmpa_tl }
802         { \l__tag_role_role_tmpa_tl }
803     }
804     {
805         \__tag_role_add_tag:oooo
806         { \l__tag_role_tag_tmpa_tl }
807         { \l__tag_role_tag_namespace_tmpa_tl }
808         { \l__tag_role_role_tmpa_tl }
809         { \l__tag_role_role_namespace_tmpa_tl }
810     }
811 }
812 ,role/map-tags .choice:
813 ,role/map-tags/false .code:n = { \socket_assign_plug:nn { tag/struct/tag } {latex-
tags} }
814 ,role/map-tags/pdf .code:n = { \socket_assign_plug:nn { tag/struct/tag } {pdf-
tags} }

815 ,role/user-NS .code:n =
816 {
817     \pdf_version_compare:NnF < {2.0}
818     {
819         \pdf_string_from_unicode:nnN{utf8/string}{https://www.latex-project.org/ns/local/#1}
820         \tl_if_empty:NF \l__tag_tmpa_str
821         {
822             \pdfdict_gput:nne
823             {g__tag_role/Namespace_user_dict}
824             {NS}
825             {\l__tag_tmpa_str}
826         }
827     }
828 }

```

deprecated names

```

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

```

```
831 }  
832 </package>
```

(End of definition for tag (rolemap-key) and others. These functions are documented on page 180.)

The tagpdf-space module

Code related to real space chars

Part of the tagpdf package

Ulrike Fischer

Version 0.99y, released 2026-01-29

Part XI

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

This key allows to activate/deactivate the real space chars if the engine supports it. The allowed values are `true`, `on`, `false`, `off`. The old name of the key `interwordspace` is still supported but deprecated.

`show-spaces` (deprecated)

This key is deprecated. Use `debug/show=spaces` instead. This key works only with `luatex` and shows with small red bars where spaces have been inserted. This is only for debugging and is not completely reliable (and change affect other literals and tagging), so it should be used with care.

```
1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-space-code} {2026-01-29} {0.99y}
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{!tx.__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 pdf_{tex} this can be done with `\pdfinterwordspaceoff` and `\pdfinterwordspaceon`. These commands insert what-sits and this mean they act globally. In lua_{tex} a attribute is used to this effect, for consistency this is also set globally.

The `off` command sets the attributes in lua_{tex}.

```

\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, 557, 620, 628
_	437, 448
A	
activate_ (setup-key)	41, <u>285</u>
activate-all (deprecated) (key)	<u>1</u>
activate-mc (deprecated) (key)	<u>1</u>
activate-struct (deprecated) (key)	<u>1</u>
activate-tree (deprecated) (key)	<u>1</u>
activate/all (key)	<u>1</u> , <u>220</u>
activate/mc (key)	<u>1</u> , <u>220</u>
activate/socket_ (setup-key)	<u>285</u>
activate/softhyphen (key)	<u>1</u> , <u>254</u>
activate/space_ (setup-key)	<u>203</u>
activate/spaces (key)	<u>1</u>
activate/spaces_ (setup-key)	<u>6</u>
activate/struct (key)	<u>1</u> , <u>220</u>
activate/struct-dest (key)	<u>1</u> , <u>220</u>
activate/tagunmarked (key)	<u>1</u> , <u>251</u>
activate/tree (key)	<u>1</u> , <u>220</u>
actualtext (key)	<u>1</u> , <u>722</u>
actualtext_ (mc-key)	<u>82</u> , <u>238</u> , <u>384</u>
add-new-tag_ (deprecated)	<u>752</u>
add-new-tag_ (setup-key)	<u>180</u>
\AddToHook	13, 16, 31, 34, 44, 57, 58, 238, 303, 395, 528, 529, 530, 534, 538, 545, 585
\AddToHookNext	750
AF (key)	<u>1</u> , <u>920</u>
AFinline (key)	<u>1</u> , <u>920</u>
AFinline-o (key)	<u>1</u> , <u>920</u>
AFref (key)	<u>1</u> , <u>920</u>
alt (key)	<u>1</u> , <u>722</u>
alt_ (mc-key)	<u>82</u> , <u>238</u> , <u>384</u>
artifact_ (mc-key)	<u>82</u> , <u>238</u> , <u>384</u>
artifact-bool internal commands:	
__artifact-bool	<u>182</u>
artifact-type internal commands:	
__artifact-type	<u>182</u>
\AssignTaggingSocketPlug	<u>526</u> , <u>527</u> , <u>602</u> , <u>609</u> , <u>715</u> , <u>716</u> , <u>778</u> , <u>787</u>
\AtBeginDocument	695
attr-unknown	<u>22</u> , <u>84</u>
attribute (key)	<u>1</u> , <u>1532</u>
attribute-class (key)	<u>1</u> , <u>1498</u>
B	
benchmark commands:	
\benchmark_tic:	637, 639
\benchmark_toc:	640
bool commands:	
\bool_gset_eq:NN	619, 634, 646, 664, 723, 737
\bool_gset_false:N	62, 221, 255, 256, 372, 620, 647, 724
\bool_gset_true:N	36, 57, 88, 96, 175, 259, 274, 289, 304, 313, 999
\bool_if:NTF	9, 13, 18, 31, 40, 40, 68, 75, 80, 85, 91, 96, 111, 135, 146, 196, 203, 208, 228, 248, 264, 265, 314, 317, 323, 336, 339, 345, 351, 389, 434, 445, 459, 461, 466, 478, 486, 511, 518, 614, 629, 641, 659, 718, 732, 1169, 1201, 1232
\bool_if:nTF	507
\bool_lazy_all:nTF	258
\bool_lazy_and:nnTF	292, 302, 305, 309, 355, 373, 548, 619, 697, 735, 790, 1002
\bool_new:N	7, 16, 20, 21, 35, 73, 83, 84, 85, 85, 86, 87, 89, 91, 93, 94, 95, 255, 323, 324, 610, 998
\bool_set_false:N	17, 165, 166, 167, 176, 189, 190, 191, 222, 344, 406, 574, 613, 640, 717
\bool_set_true:N	16, 90, 92, 175, 176, 177, 200, 201, 202, 256, 346, 405, 573
box commands:	
\box_dp:N	180, 184
\box_ht:N	170
\box_new:N	78, 79
\box_set_dp:Nn	178, 180
\box_set_eq:NN	193
\box_set_ht:Nn	177, 179
\box_use_drop:N	182, 186
\boxmaxdepth	95, 181
C	
c@g internal commands:	
\c@g__tag_MCID_abs_int	11, 15, 28, 37, 50, 57, 60, 68, 74, 134, 138, 178, 242, 245, 288, 295, 346

\c@g__tag_parenttree_obj_int	155 , 500	83 , 84 , 85 , 86 , 93 , 94 , 95 , 107 , 108 ,
\c@g__tag_struct_abs_int	.. 6 , 18 ,	109 , 117 , 120 , 122 , 126 , 137 , 142 ,
	40 , 58 , 91 , 114 , 115 , 118 , 125 , 128 ,	147 , 153 , 161 , 163 , 167 , 169 , 171 ,
	149 , 166 , 259 , 393 , 550 , 727 , 740 ,	172 , 175 , 189 , 195 , 211 , 212 , 213 ,
	785 , 797 , 811 , 827 , 842 , 850 , 904 ,	214 , 215 , 216 , 227 , 233 , 246 , 248 ,
	915 , 934 , 937 , 942 , 978 , 980 , 985 ,	255 , 259 , 260 , 263 , 264 , 266 , 277 ,
	997 , 999 , 1004 , 1095 , 1106 , 1107 ,	280 , 281 , 285 , 286 , 292 , 295 , 305 ,
	1108 , 1109 , 1110 , 1112 , 1114 , 1120 ,	310 , 315 , 318 , 320 , 322 , 327 , 336 ,
	1125 , 1132 , 1135 , 1145 , 1153 , 1157 ,	337 , 340 , 349 , 349 , 350 , 353 , 357 ,
	1172 , 1185 , 1195 , 1208 , 1211 , 1226 ,	361 , 365 , 369 , 384 , 388 , 393 , 395 ,
	1227 , 1229 , 1240 , 1525 , 1528 , 1576	396 , 403 , 415 , 417 , 428 , 429 , 431 ,
catalog-supplemental-file (key)	... 1075	434 , 442 , 453 , 462 , 503 , 503 , 527 ,
cctab commands:		534 , 541 , 541 , 548 , 554 , 568 , 576 ,
\c_document_cctab	49 , 54 , 75 , 155 , 510	578 , 581 , 582 , 583 , 583 , 584 , 591 ,
\chapter 191 , 357 , 375	592 , 598 , 611 , 625 , 633 , 634 , 634 ,
check commands:		635 , 638 , 640 , 654 , 713 , 729 , 855 ,
check_parent_child_rules 985	863 , 876 , 889 , 922 , 950 , 1095 , 1096 ,
check_update_stashed 985	1097 , 1293 , 1334 , 1386 , 1399 , 1419 ,
clist commands:		1423 , 1427 , 1431 , 1437 , 1456 , 1480
\clist_const:Nn 80 , 81	\cs_set:Nn 679 , 680 , 744 , 745
\clist_if_empty:NTF 1537	\cs_set:Npn 47 , 52 , 89 , 109
\clist_map_inline:nn	... 107 , 419 , 901	\cs_set_eq:NN 14 , 20 ,
\clist_new:N 76	66 , 80 , 81 , 82 , 140 , 141 , 142 , 143 ,
\clist_set:Nn 1502 , 1536	144 , 145 , 146 , 147 , 148 , 149 , 182 ,
color commands:		183 , 194 , 208 , 217 , 218 , 225 , 230 ,
\color_select:n 437 , 448	235 , 236 , 237 , 238 , 386 , 387 , 388 ,
cs commands:		389 , 639 , 640 , 672 , 673 , 674 , 675 ,
\cs:w 756 , 1408 , 1412	681 , 682 , 686 , 687 , 688 , 689 , 746 , 747
\cs_end: 756 , 1408 , 1412	\cs_set_protected:Nn
\cs_generate_variant:Nn 169 , 216 , 243 , 359 , 365 , 1250 , 1251
...	44 , 79 , 99 , 100 , 101 , 101 , 102 ,	\cs_set_protected:Npn 9 ,
	103 , 104 , 105 , 106 , 107 , 107 , 107 ,	15 , 16 , 22 , 29 , 35 , 38 , 40 , 48 , 49 , 52 ,
	108 , 116 , 117 , 118 , 126 , 134 , 151 ,	58 , 63 , 65 , 71 , 73 , 78 , 83 , 83 , 87 , 94 ,
	152 , 153 , 153 , 154 , 155 , 156 , 157 ,	95 , 99 , 102 , 116 , 120 , 143 , 161 , 170 ,
	163 , 164 , 168 , 181 , 194 , 214 , 226 ,	184 , 195 , 220 , 228 , 228 , 240 , 249 ,
	247 , 253 , 263 , 265 , 276 , 279 , 304 ,	265 , 283 , 299 , 305 , 367 , 371 , 375 ,
	314 , 335 , 589 , 633 , 681 , 717 , 921 ,	379 , 1099 , 1100 , 1287 , 1295 , 1336 , 1388
	949 , 970 , 1384 , 1396 , 1436 , 1465 , 1486	\cs_to_str:N
\cs_gset_eq:NN 432	.. 12 , 18 , 25 , 32 , 38 , 43 , 61 , 62 , 68 , 69
\cs_if_exist:NTF	249 , 587 , 637 , 752 , 759	\cs_undefine:N 57
\cs_if_exist_p:N	.. 357 , 360 , 375 , 378	
\cs_if_exist_use:NTF 400 , 1390	
\cs_if_free:NTF 48	
\cs_new:Nn	
.	83 , 109 , 131 , 136 , 293 , 409 , 410 , 411	
\cs_new:Npn 9 , 15 , 23 , 27 , 105 ,	
	119 , 149 , 156 , 215 , 229 , 235 , 237 ,	
	391 , 528 , 536 , 542 , 548 , 754 , 1380 , 1466	
\cs_new_eq:NN 37	
\cs_new_protected:Nn	
 74 , 127 , 167 , 296 , 412 , 416	
\cs_new_protected:Npn 13 ,	
	17 , 20 , 22 , 23 , 30 , 31 , 36 , 42 , 43 , 45 ,	
	60 , 61 , 63 , 65 , 67 , 78 , 79 , 80 , 81 , 82 ,	

D

debug/log (key) 1 , 238
debug/show (key) 237
debug/structures_ (show-key)	... 42 , 254
debug/uncompress (key) 238
\DebugSocketsOn 44
\DeclareOption 37 , 38
dim commands:	
\c_max_dim 169 , 194
\c_zero_dim 177 , 178 , 179
\documentclass 16
\DocumentMetadata 15

E	
E (key)	1 , 722 , 897
\endinput	22
\ERRORusetaggingsocket	106 , 121
exclude-header-footer _□ (deprecated)	667
exp commands:	
\exp_args:Ne	122 , 530
\exp_args:NNe	86 , 89 , 195 , 215
\exp_args:Nne	79 , 337 , 341 , 427 , 461
\exp_args:NNno	774
\exp_args:No	291 , 326
\exp_last_unbraced:Ne	99 , 102 , 109
\exp_last_unbraced:No	135 , 138 , 152 , 154 , 157 , 159 , 206 , 207 , 238 , 239 , 594 , 597 , 605 , 606 , 608 , 610 , 1276
\exp_not:n	187 , 206
F	
file commands:	
\file_if_exist:nTF	322
\file_input:n	332
firstkid (key)	1 , 722
flag commands:	
\flag_clear:n	239
\flag_height:n	138 , 251
\flag_new:n	136
\flag_raise:n	252
\fontencoding	6
\fontfamily	6
\fontseries	6
\fontshape	6
\fontsize	6
G	
group commands:	
\group_begin:	67 , 76 , 173 , 311 , 927 , 1019 , 1027 , 1062 , 1079 , 1105
\group_end:	74 , 79 , 213 , 350 , 945 , 1023 , 1033 , 1072 , 1090 , 1246
H	
\halign	44
hbox commands:	
\hbox_set:Nn	171 , 172
hook commands:	
\hook_gput_code:nnn	7 , 11 , 33 , 57 , 66 , 80 , 156 , 239 , 288 , 289 , 353 , 371 , 387 , 391 , 802 , 809 , 816 , 823 , 830 , 837 , 843 , 850 , 856 , 863 , 871 , 884 , 895 , 908 , 919 , 932 , 943 , 956 , 966 , 979
\hook_new:n	348
\hook_use:n	353
I	
\IfFormatAtLeastTF	336
\IfPDFManagementActiveF	6
\ignorespaces	41
int commands:	
\int_abs:n	154
\int_case:nnTF	99 , 114 , 329
\int_compare:nNnTF	22 , 58 , 70 , 98 , 116 , 124 , 125 , 132 , 142 , 148 , 157 , 170 , 173 , 173 , 279 , 329 , 359 , 378 , 405 , 408 , 436 , 442 , 444 , 450 , 512 , 528 , 529 , 536 , 543 , 550 , 551 , 560 , 562 , 570 , 578 , 578 , 583 , 585 , 593 , 600 , 619 , 734 , 748 , 1136
\int_compare:nTF	180 , 476 , 1518 , 1520 , 1522 , 1546 , 1572
\int_compare_p:nNn	740
\int_decr:N	172 , 197
\int_eval:n	118 , 138 , 166 , 197 , 396 , 621 , 629 , 737 , 742 , 745 , 942 , 985 , 1004 , 1107 , 1108 , 1109 , 1110 , 1226 , 1227 , 1229 , 1240 , 1528
\int_gincr:N	178 , 242 , 288 , 295 , 351 , 355 , 359 , 363 , 369 , 373 , 377 , 381 , 500 , 928 , 1064 , 1081 , 1095 , 1106
\int_gset:Nn	7 , 82 , 158
\int_if_zero:nTF	172 , 173 , 197 , 198 , 617 , 625
\int_incr:N	93 , 164 , 188 , 442
\int_new:N	6 , 77 , 78 , 82 , 97 , 155 , 160 , 326 , 327 , 328 , 329 , 920
\int_rand:n	61 , 62 , 64 , 66 , 68 , 70 , 71
\int_set:Nn	239 , 242 , 245 , 246 , 247
\int_step_inline:nn	473 , 479
\int_step_inline:nnn	25 , 91 , 259
\int_step_inline:nnnn	149 , 174 , 177 , 200 , 461 , 467
\int_to_arabic:n	154 , 156
\int_to_Hex:n	61 , 62 , 64 , 66 , 68 , 70 , 71
\int_use:N	11 , 15 , 18 , 28 , 37 , 40 , 49 , 50 , 54 , 57 , 58 , 60 , 68 , 74 , 75 , 100 , 115 , 125 , 132 , 134 , 155 , 163 , 180 , 187 , 206 , 234 , 241 , 245 , 277 , 279 , 346 , 393 , 437 , 448 , 510 , 515 , 550 , 556 , 557 , 565 , 566 , 727 , 785 , 797 , 811 , 827 , 842 , 850 , 904 , 915 , 931 , 934 , 937 , 978 , 980 , 997 , 999 , 1068 , 1071 , 1085 , 1089 , 1114 , 1120 , 1125 , 1132 , 1135 , 1157 , 1172 , 1185 , 1195 , 1208 , 1211 , 1466 , 1525 , 1576
\int_zero:N	90 , 105 , 430
intarray commands:	
\intarray_gset:Nnn	397 , 406 , 421 , 425 , 439
\intarray_item:Nn	441 , 444 , 554
\intarray_new:Nn	390 , 431

msg commands:	\nointerlineskip 185
\msg_error:nn 317, 338, 473, 1142	
\msg_error:nnn 354, 365, 373, 384, 459, 1512, 1552	P
\msg_error:nnnn 242	\PackageError 8
\msg_error:nnnnn 553, 562	\PackageWarning 22
\msg_info:nnn 134, 172, 325, 331, 333, 407, 411	page/exclude-header-footer_␣(setup-key) 43, 667
\msg_info:nnnn 361, 380, 420	page/tabsorder (key) 1, 318
\msg_line_context: .. 93, 97, 107, 114, 524, 525, 557, 561, 565, 621, 629	para-flattened_␣(deprecated) 402
\g_msg_module_name_prop 24, 28	para-hook-count-wrong 22, 231
\g_msg_module_type_prop 27	para/flattened_␣(tool-key) 402
\msg_new:nnn 7, 8, 9, 12, 13, 14, 15, 16, 22, 24, 25, 32, 35, 36, 38, 40, 42, 47, 54, 65, 74, 85, 86, 87, 88, 89, 90, 92, 94, 104, 111, 164, 213, 215, 216, 217, 218, 219, 220, 226, 228, 524, 525, 555, 559, 563, 615, 623	para/maintag_␣(setup-key) 402
\msg_new:nnnn 231	para/maintag_␣(tool-key) 402
\msg_note:nn 29, 199	para/tag_␣(setup-key) 402
\msg_note:nnn 163, 180, 545, 552, 587, 595	para/tag_␣(tool-key) 402
\msg_note:nnnn 128, 186, 205, 531, 538, 572, 580, 587	para/tagging_␣(setup-key) 43, 402
\msg_note:nnnnn 519, 569	para/tagging_␣(tool-key) 402
\msg_redirect_name:nnn 549	\PARALABEL 504
\msg_show_item_unbraced:n 276	paratag_␣(deprecated) 402
\msg_show_item_unbraced:nn 267	paratagging_␣(deprecated) 43, 402
\msg_term:nnnnnn 261, 270	paratagging-show_␣(deprecated) .. 43, 402
\msg_warning:nn 24, 222	parent (key) 1, 722
\msg_warning:nnn 12, 14, 43, 45, 54, 242, 248, 321, 324, 347, 392, 400, 425, 449, 666, 693, 873, 886, 1329, 1348, 1374	pdf commands:
\msg_warning:nnnn 440, 608, 744	\pdf_activate_indexed_structure_destination: 311
\msg_warning:nnnnn 126, 175, 530, 580, 630, 708	\pdf_bdc:nn 237
\msg_warning:nnnnnn 146	\pdf_bdc_shipout:nn 238
	\pdf_bmc:n 235
N	\l_pdf_current_structure_destination_tl 309
\n 1012	\pdf_emc: 236
namespace_␣(rolemap-key) 180	\pdf_name_from_unicode_e:n 105, 114, 119, 167, 180, 190, 278, 1031, 1483, 1506, 1542
new-tag 22, 215	\pdf_object_if_exist:n 98
newattribute_␣(deprecated) 115, 1480	\pdf_object_if_exist:nTF .. 975, 1040
\newcommand 570, 571	\pdf_object_new:n 114, 34, 36, 154, 262, 310, 321
\newcounter 8	\pdf_object_new_indexed:nn 31, 1111
\NewDocumentCommand 6, 23, 29, 34, 40, 46, 51, 56, 126, 316, 575	\pdf_object_ref:n 114, 56, 99, 131, 135, 141, 192, 318, 335, 978, 1042, 1089
\newmarks 13	\pdf_object_ref_indexed:nn 57, 74, 96, 128, 211, 255, 271, 414, 435, 496, 524, 1382
\NewTaggingSocket 454, 455, 767, 768	\pdf_object_ref_last: 114, 104, 118, 124, 294, 1447, 1453, 1561
\NewTaggingSocketPlug 457, 476, 509, 595, 603, 699, 705, 770, 780	\pdf_object_unnamed_write:nn 100, 111, 120, 244, 286, 1439, 1556
no-struct-dest (deprecated) (key) 1	\pdf_object_write:nnn 257, 281, 311, 330, 337, 342
	\pdf_object_write_indexed:nnnn 139, 449
	\pdf_pageobject_ref:n .. 221, 486, 514
	\pdf_string_from_unicode:nnN 42, 819

\pdf_uncompress: 248, 250
 \pdf_version: 237, 240, 242
 \pdf_version_compare:NnTF
 20, 81, 136, 154, 159, 227,
 257, 324, 351, 431, 471, 590, 797, 817
 \pdf_version_gset:n 243
 pdfannot commands:
 \pdfannot_dict_put:nnn
 100, 877, 901, 925, 949, 972
 \pdfannot_link_ref_last:
 891, 915, 939, 963, 986
 pdfdict commands:
 \pdfdict_gput:nnn
 38, 45, 53, 187, 276, 334, 822
 \pdfdict_if_empty:nTF 328
 \pdfdict_new:n 18, 35, 37
 \pdfdict_put:nnn 1020,
 1021, 1028, 1029, 1030, 1063, 1080
 \pdfdict_use:n 283, 332, 339
 \pdffakespace 42, 314
 pdffile commands:
 \pdffile_embed_file:nnn
 108, 1065, 1082
 \pdffile_embed_stream:nnN . 921, 929
 \pdffile_embed_stream:nnn 101
 \pdfglyphtounicode 30
 \pdfinterwordspaceoff 205, 118
 \pdfinterwordspaceon 205, 34, 122
 pdfmanagement commands:
 \pdfmanagement_add:nnn
 .. 52, 70, 71, 320, 322, 324, 393, 1086
 \pdfmanagement_remove:nn 326
 phoneme (key) 722
 prg commands:
 \prg_do_nothing:
 37, 82, 102, 117, 386,
 387, 388, 389, 432, 686, 687, 688, 689
 \prg_generate_conditional_
 variant:Nnn 98
 \prg_new_conditional:Nnn ... 68, 226
 \prg_new_conditional:Npnn
 251, 275, 290, 300, 499, 505, 516
 \prg_new_eq_conditional:NNn . 82, 233
 \prg_new_protected_conditional:Npnn
 719
 \prg_replicate:nn 153
 \prg_return_false: 78, 230, 252, 270,
 281, 284, 297, 307, 502, 514, 520, 749
 \prg_return_true: .. 79, 229, 267,
 280, 294, 304, 503, 513, 519, 719, 750
 \prg_set_conditional:Npnn 256
 \prg_set_protected_conditional:Npnn
 721
 process commands:
 process_softhyphen_preprocess_
 softhyphen_post 925
 \ProcessOptions 39
 prop commands:
 \prop_clear:N 176
 \prop_count:N 203
 \prop_gclear:N 1017
 \prop_get:NnN 127, 144, 145,
 177, 198, 214, 268, 299, 456, 468,
 470, 483, 484, 485, 546, 547, 585, 586
 \prop_get:NnNTF 44, 96, 130,
 137, 144, 158, 181, 183, 201, 205,
 236, 295, 312, 322, 342, 357, 376,
 395, 419, 423, 432, 514, 564, 599,
 604, 609, 614, 650, 677, 685, 701,
 752, 780, 785, 865, 878, 1179, 1277,
 1343, 1402, 1440, 1510, 1550, 1554
 \prop_gput:Nnn
 . 24, 26, 27, 28, 31, 56, 88, 90, 91,
 97, 98, 99, 99, 100, 101, 102, 103,
 110, 112, 113, 119, 121, 122, 143,
 145, 269, 272, 286, 291, 383, 434,
 436, 437, 448, 469, 475, 481, 488,
 490, 491, 726, 1018, 1020, 1228,
 1239, 1314, 1359, 1482, 1514, 1561
 \prop_gremove:Nn 137, 147, 1021
 \prop_gset_eq:NN 146, 1225
 \prop_gset_from_keyval:Nn 991
 \prop_if_exist:NnTF 174,
 209, 234, 320, 430, 648, 675, 1299, 1340
 \prop_if_exist_p:N 737
 \prop_item:Nn 41, 99, 102, 104, 109,
 115, 147, 244, 533, 1236, 1559, 1566
 \prop_map_function:NN 265
 \prop_map_inline:Nn 267, 272,
 293, 326, 363, 381, 398, 486, 495, 1004
 \prop_map_tokens:Nn 344
 \prop_new:N 8, 9, 10, 11, 11, 25,
 33, 73, 140, 144, 990, 1108, 1475, 1478
 \prop_new_linked:N
 7, 17, 84, 89, 91, 141, 1476
 \prop_put:Nnn 103, 188
 \prop_show:N
 .. 67, 95, 149, 1222, 1243, 1528, 1555
 property commands:
 \property_new:nnnn
 123, 126, 130, 133, 137
 \property_record:nn 59, 112
 \property_ref:nn 113, 117
 \property_ref:nnn
 42, 116, 121, 181, 190,
 221, 222, 343, 478, 487, 489, 1300, 1304
 \providecommand 62, 63, 64, 65, 66, 69, 70, 321

\ProvidesExplFile 3
 \ProvidesExplPackage 3, 3,
 3, 3, 3, 3, 3, 3, 3, 3, 7, 7, 20, 31, 1471

Q

\quad 233, 234
 quark commands:
 \q_no_value 606, 616, 662, 667, 689, 694
 \quark_if_no_value:NTF
 132, 178, 199, 215, 269, 300, 591, 602
 \quark_if_no_value_p:N
 549, 550, 620, 621, 698, 699
 \q_stop 259, 292, 328

R

raw_␣(mc-key) 82, 238, 384
 ref (key) 1, 722, 897
 \RemoveFromHook 35, 532, 533
 \renewcommand 573, 574
 \RenewDocumentCommand 8
 \RequirePackage ... 40, 338, 341, 347, 350
 \rlap 448
 role_␣(rolemap-key) 180, 752
 role commands:
 role_get_parent_child_rule 978
 role-MC-child-forbidden 104
 role-missing 22, 86
 role-namespace_␣(rolemap-key) . 180, 752
 role-parent-child-check 90
 role-parent-child-forbidden 111
 role-parent-child-result 22, 92
 role-parent-child-unresolved 164
 role-remapping 22, 213
 role-struct-parent-child-forbidden . 94
 role-tag 22, 215
 role-unknown 22, 86
 role-unknown-NS 22, 86
 role-unknown-tag 22, 86
 role/new-attribute_␣(setup-key) 115, 1480
 role/new-tag_␣(setup-key) 752
 root-AF (key) 1, 1036
 root-supplemental-file (key) 1058

S

\selectfont 6
 seq commands:
 \seq_clear:N 319, 466
 \seq_const_from_clist:Nn 39, 52
 \seq_count:N 22, 25, 58,
 331, 444, 1518, 1520, 1522, 1546, 1572
 \seq_get:NN 723, 736
 \seq_get:NNTF 469, 602, 1138, 1265, 1273
 \seq_gpop:NN 1258
 \seq_gpop:NNTF 106, 1259

\seq_gpop_left:NN 307
 \seq_gpush:Nn 13, 15, 89, 96, 1145, 1151
 \seq_gput_left:Nn .. 42, 145, 273, 311
 \seq_gput_right:Nn
 . 37, 144, 146, 152, 236, 257, 296, 486
 \seq_gset_eq:NN 159, 221, 326
 \seq_if_empty:NTF 200, 438
 \seq_item:Nn
 . 59, 116, 118, 125, 129, 136, 140,
 146, 348, 355, 368, 509, 511, 518,
 694, 695, 710, 711, 761, 762, 775, 776
 \seq_log:N 175, 199, 249, 412, 573, 588
 \seq_map_function:NN 274
 \seq_map_indexed_inline:Nn 446, 459
 \seq_map_inline:Nn 289, 320, 1508, 1548
 \seq_new:N
 12, 14, 14, 15, 16, 17, 18, 19,
 21, 22, 24, 74, 75, 142, 145, 1110, 1479
 \seq_pop_left:NN 455, 457, 458
 \seq_put_right:Nn 321
 \seq_remove_all:Nn 324
 \seq_set_eq:NN 207, 208
 \seq_set_from_clist:NN ... 1503, 1539
 \seq_set_from_clist:Nn
 87, 90, 196, 216, 443, 454
 \seq_set_map_e:NNn 1504, 1540
 \seq_set_split:Nnn 51,
 105, 687, 691, 703, 707, 754, 758, 774
 \seq_show:N
 60, 148, 216, 217, 250, 322,
 323, 325, 496, 1155, 1223, 1244, 1254
 \seq_use:Nn
 50, 110, 111, 205, 233, 234, 383, 1519

Setup keys:

activate-all (deprecated) 1
 activate-mc (deprecated) 1
 activate-struct (deprecated) 1
 activate-tree (deprecated) 1
 activate/all 1, 220
 activate/mc 1, 220
 activate/softhyphen 1, 254
 activate/spaces 1
 activate/struct 1, 220
 activate/struct-dest 1, 220
 activate/tagunmarked 1, 251
 activate/tree 1, 220
 catalog-supplemental-file 1075
 debug/log 1, 238
 debug/show 237
 debug/uncompress 238
 log (deprecated) 238
 no-struct-dest (deprecated) 1
 page/tabsorder 1, 318
 root-AF 1, 1036

`\tag_mc_begin:n` ... [11](#), [81](#), [25](#), [66](#),
[114](#), [169](#), [169](#), [295](#), [295](#), [299](#), [305](#),
[436](#), [447](#), [473](#), [505](#), [622](#), [650](#), [702](#), [775](#)
`\tag_mc_begin_pop:n` [81](#),
[76](#), [80](#), [81](#), [102](#), [631](#), [661](#), [734](#), [785](#)
`\tag_mc_end:` [81](#),
[31](#), [75](#), [93](#), [216](#), [216](#), [295](#), [296](#), [359](#),
[365](#), [438](#), [449](#), [515](#), [628](#), [657](#), [707](#), [783](#)
`\tag_mc_end_push:`
. [81](#), [65](#), [80](#), [80](#), [83](#), [616](#), [643](#), [720](#), [773](#)
`\tag_mc_if_in:` [82](#), [233](#)
`\tag_mc_if_in:TF` [81](#), [42](#), [68](#), [226](#)
`\tag_mc_if_in_p:` [81](#), [68](#), [226](#)
`\tag_mc_new_stream:n` [82](#), [17](#), [17](#), [67](#), [67](#)
`\tag_mc_reset_box:N` [81](#), [79](#), [79](#), [228](#), [228](#)
`\tag_mc_use:n` [81](#), [36](#), [36](#), [36](#), [38](#)
`\l_tag_para_attr_class_tl` . [397](#), [399](#)
`\tag_resume:n`
... [7](#), [73](#), [159](#), [195](#), [208](#), [218](#), [627](#), [656](#)
`\tag_socket_use:n`
..... [44](#), [45](#), [62](#), [72](#), [73](#), [528](#), [529](#)
`\tag_socket_use:nn` . [44](#), [45](#), [63](#), [72](#), [78](#)
`\tag_socket_use:nnn` . [44](#), [45](#), [64](#), [72](#), [83](#)
`\tag_socket_use_expandable:n` ...
..... [44](#), [45](#), [65](#), [72](#), [89](#)
`\tag_spacechar_off:` ... [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](#), [54](#), [159](#), [161](#), [182](#), [211](#)
`\tag_stop:n` [7](#), [159](#), [194](#), [215](#), [217](#)
`\tag_struct_begin:n`
. [112](#), [48](#), [464](#), [471](#), [489](#), [499](#), [649](#),
[701](#), [726](#), [774](#), [1095](#), [1095](#), [1099](#), [1100](#)
`\tag_struct_end:`
.. [112](#), [26](#), [53](#), [517](#), [521](#), [658](#), [708](#),
[731](#), [784](#), [1095](#), [1096](#), [1250](#), [1251](#), [1290](#)
`\tag_struct_end:n` ... [112](#), [1097](#), [1287](#)
`\tag_struct_gput:nnn`
..... [112](#), [903](#), [1386](#), [1386](#), [1388](#), [1396](#)
`\tag_struct_gput_ref:nnn` [113](#)
`\tag_struct_insert_annot:nn`
..... [112](#), [150](#), [890](#),
[914](#), [938](#), [962](#), [985](#), [1456](#), [1456](#), [1465](#)
`\tag_struct_object_ref:n`
. [112](#), [869](#), [882](#), [893](#), [1379](#), [1380](#), [1384](#)
`\tag_struct_parent_int:`
..... [112](#), [150](#), [880](#), [891](#), [904](#), [915](#),
[928](#), [939](#), [952](#), [963](#), [975](#), [986](#), [1456](#), [1466](#)
`\tag_struct_use:n`
..... [112](#), [113](#), [58](#), [1293](#), [1293](#), [1295](#)
`\tag_struct_use_num:n`
..... [112](#), [1334](#), [1334](#), [1336](#)

`\tag_suspend:n`
... [7](#), [68](#), [159](#), [184](#), [194](#), [217](#), [623](#), [651](#)
`\tag_tool:n` [41](#), [13](#), [13](#), [14](#), [16](#), [20](#)
tag internal commands:
`__tag_activate_mark_space` [563](#)
`\g__tag_active_mc_bool`
..... [40](#), [83](#), [223](#), [230](#), [261](#), [292](#)
`\l__tag_active_mc_bool`
..... [89](#), [166](#), [176](#), [190](#), [201](#), [264](#), [292](#)
`\l__tag_active_socket_bool`
..... [75](#), [80](#), [85](#),
[89](#), [91](#), [96](#), [111](#), [167](#), [177](#), [191](#), [202](#), [293](#)
`\g__tag_active_space_bool`
..... [13](#), [57](#), [62](#), [83](#)
`\g__tag_active_struct_bool`
..... [83](#), [225](#), [232](#), [260](#), [302](#), [307](#), [466](#)
`\l__tag_active_struct_bool`
..... [89](#), [165](#), [175](#), [189](#), [200](#), [263](#), [302](#)
`\g__tag_active_struct_dest_bool`
..... [83](#), [229](#), [236](#), [306](#)
`\g__tag_active_tree_bool`
..... [9](#), [68](#), [83](#), [224](#), [231](#), [262](#), [351](#), [389](#)
`__tag_add_missing_mcs:Nn`
..... [96](#), [167](#), [167](#), [219](#)
`__tag_add_missing_mcs_to_-`
stream:Nn . [65](#), [65](#), [66](#), [189](#), [189](#), [225](#)
`\g__tag_attr_class_used_prop` ...
..... [291](#), [293](#), [1474](#), [1514](#)
`\g__tag_attr_class_used_seq` [289](#), [1479](#)
`\g__tag_attr_entries_prop`
[295](#), [1474](#), [1482](#), [1510](#), [1550](#), [1555](#), [1559](#)
`__tag_attr_new_entry:nn`
... [637](#), [1480](#), [1480](#), [1486](#), [1491](#), [1495](#)
`\g__tag_attr_objref_prop`
..... [1474](#), [1554](#), [1561](#), [1566](#)
`\l__tag_attr_value_tl` [1474](#),
[1544](#), [1563](#), [1568](#), [1570](#), [1574](#), [1578](#)
`__tag_backend_create_bdc_node` .. [437](#)
`__tag_backend_create_bmc_node` .. [408](#)
`__tag_backend_create_emc_node` .. [379](#)
`__tag_check_add_tag_role:nn` ...
..... [129](#), [350](#), [350](#)
`__tag_check_add_tag_role:nnn` ..
..... [169](#), [369](#)
`__tag_check_benchmark_tic:` . [356](#),
[360](#), [364](#), [368](#), [372](#), [376](#), [380](#), [633](#), [639](#)
`__tag_check_benchmark_toc:` . [358](#),
[362](#), [366](#), [370](#), [374](#), [378](#), [382](#), [634](#), [640](#)
`__tag_check_forbidden_parent_-`
child:nnnn [120](#), [120](#), [134](#), [171](#)
`__tag_check_if_active_mc:` [290](#)
`__tag_check_if_active_mc:TF` ...
..... [85](#), [104](#),
[171](#), [191](#), [218](#), [289](#), [301](#), [307](#), [361](#), [367](#)

```

__tag_check_if_active_struct: . 300
__tag_check_if_active_struct:TF
..... 40, 289, 1102, 1103,
1255, 1256, 1289, 1297, 1338, 1459
__tag_check_if_mc_in_galley: .. 499
__tag_check_if_mc_in_galley:TF
..... 209, 230
__tag_check_if_mc_tmb_missing: 505
__tag_check_if_mc_tmb_missing:TF
..... 112, 218, 235, 505
__tag_check_if_mc_tmb_missing_-
p: ..... 505
__tag_check_if_mc_tme_missing: 516
__tag_check_if_mc_tme_missing:TF
..... 155, 222, 239, 516
__tag_check_if_mc_tme_missing_-
p: ..... 516
__tag_check_info_closing_-
struct:n ..... 327, 327, 335, 1261
__tag_check_init_mc_used: .....
..... 429, 429, 432, 438
__tag_check_mc_if_nested: .....
..... 174, 312, 388, 388
__tag_check_mc_if_open: .....
..... 220, 371, 388, 396
__tag_check_mc_in_galley:TF ... 499
__tag_check_mc_in_galley_p: ... 499
__tag_check_mc_pushed_popped:nn
..... 90, 97, 110, 113, 118, 403, 403
__tag_check_mc_tag:N .....
..... 193, 330, 415, 415
__tag_check_mc_used:n .....
..... 145, 268, 434, 434
g__tag_check_mc_used_intarray .
..... 429, 439, 441, 444
__tag_check_no_open_struct: ...
..... 336, 336, 1263, 1271
__tag_check_para_begin_show:nn
..... 431, 472, 504
__tag_check_para_end_show:nn ..
..... 442, 516
c__tag_check_pdfversion_tl ....
..... 237, 240, 242, 243
__tag_check_show_MCID_by_page:
..... 453, 453
__tag_check_struct_forbidden_-
parent_child:nnn ... 137, 163, 628
__tag_check_struct_used:n ....
..... 340, 340, 1302
__tag_check_structure_has_tag:n
..... 310, 310, 1135
__tag_check_structure_tag:N ...
..... 320, 320, 696, 719, 770

__tag_check_typeout_v:n .....
..... 110, 111, 114, 149, 157, 164,
202, 211, 230, 230, 243, 482, 498, 514
__tag_check_unresolved_parent_-
child:nnnn ..... 169, 169
g__tag_css_bool . 998, 999, 1002, 1013
g__tag_css_prop .....
990, 991, 1004, 1017, 1018, 1020, 1021
__tag_debug_mc_begin_ignore:n .
..... 354, 534
__tag_debug_mc_begin_insert:n .
..... 309, 527
__tag_debug_mc_end_ignore: 379, 548
__tag_debug_mc_end_insert: 369, 541
__tag_debug_struct_begin_-
ignore:n ..... 576, 1248
__tag_debug_struct_begin_-
insert:n ..... 568, 1245
__tag_debug_struct_end_check:n
..... 598, 1289
__tag_debug_struct_end_ignore:
..... 591, 1284
__tag_debug_struct_end_insert:
..... 583, 1282
__tag_exclude_headfoot_begin: .
..... 611, 672, 673
__tag_exclude_headfoot_end: ...
..... 625, 674, 675
__tag_exclude_struct_headfoot_-
begin:n ..... 638, 679, 680
__tag_exclude_struct_headfoot_-
end: ..... 654, 681, 682
__tag_fakespace ..... 492
__tag_fakespace: ..... 72, 74, 318
__tag_finish_structure: .....
..... 13, 16, 348, 349
l__tag_get_child_tmpa_tl .....
..... 60, 569, 574, 641, 643, 653,
656, 667, 1303, 1307, 1309, 1315, 1325
l__tag_get_child_tmpb_tl .....
..... 60, 570, 575, 642, 654
l__tag_get_child_tmpc_tl .....
..... 60, 145, 157, 159
__tag_get_data_mc_counter: .... 9, 9
__tag_get_data_mc_tag: .....
..... 237, 237, 293, 293
__tag_get_data_struct_counter:
..... 547, 548
__tag_get_data_struct_id: 536, 536
__tag_get_data_struct_num: 541, 542
__tag_get_data_struct_tag: 528, 528
__tag_get_mathsubtype ..... 303
__tag_get_mc_abs_cnt: .....
..... 14, 15, 19, 20, 102,

```


137, 166, 177, 183, 210, 246, 254,
 272, 286, 307, 321, 331, 392, 400, 420
 __tag_get_mc_cnt_type_tag [297](#)
 __tag_get_num_from [322](#)
 \l__tag_get_parent_tmpa_tl
 . [60](#), 127, 132, 136, 139, 149, 152,
 162, 165, 175, 564, 572, 585, 591,
 595, 598, 665, 667, 727, 730, 740, 743
 \l__tag_get_parent_tmpb_tl
 [60](#), 150,
 153, 163, 166, 175, 565, 573, 586,
 602, 606, 609, 666, 728, 731, 741, 744
 \l__tag_get_parent_tmpc_tl
 [60](#), 144, 152, 154
 __tag_get_tag_from [341](#)
 \l__tag_get_tmpc_tl [60](#),
 181, 186, 204, 206, 207, 236, 238,
 239, 1182, 1188, 1405, 1407, 1411, 1417
 __tag_gincr_para_begin_int: ...
 [349](#), 353, 371, 387, 470, 497
 __tag_gincr_para_end_int: ...
 [349](#), 361, 379, 389, 513
 __tag_gincr_para_main_begin_
 int: .. [349](#), 349, 367, 386, 463, 488
 __tag_gincr_para_main_end_int:
 [349](#), 357, 375, 388, 520
 __tag_headfoot_tagged_begin:n .
 [713](#), 744, 745
 __tag_headfoot_tagged_end:
 [729](#), 746, 747
 __tag_hook_kernel_after_foot: .
 [584](#), 592, 607, 675, 682, 689, 747
 __tag_hook_kernel_after_head: .
 [582](#), 590, 599, 674, 681, 688, 746
 __tag_hook_kernel_before_foot:
 [583](#), 591, 605, 673, 680, 687, 745
 __tag_hook_kernel_before_head:
 [581](#), 589, 597, 672, 679, 686, 744
 \g__tag_in_mc_bool [16](#),
 18, 175, 221, 228, 313, 372, 619,
 620, 634, 646, 647, 664, 723, 724, 737
 __tag_insert_bdc_node [437](#)
 __tag_insert_bmc_node [408](#)
 __tag_insert_emc_node [379](#)
 __tag_log [225](#)
 \l__tag_loglevel_int
 [82](#), 125, 132, 170, 173, 239,
 242, 245, 246, 247, 329, 359, 378,
 406, 409, 436, 512, 529, 536, 543,
 550, 562, 570, 578, 583, 585, 593, 600
 __tag_mark_spaces [497](#)
 __tag_mc_artifact_begin_marks:n
 [23](#), 45, 81, 327
 \l__tag_mc_artifact_bool
 [20](#), 176, 185, 196, 222, 323
 \l__tag_mc_artifact_type_tl
 [19](#), 189, 193, 197,
 201, 205, 209, 213, 217, 325, 327, 344
 __tag_mc_bdc:nn [234](#), 237, 283
 __tag_mc_bdc_mcid:n ... [123](#), [239](#), 255
 __tag_mc_bdc_mcid:nn
 [239](#), 240, 257, 262
 __tag_mc_bdc_shipout:nn .. [238](#), 248
 __tag_mc_begin_marks:nn
 [23](#), 23, 44, 80, 334
 __tag_mc_bmc:n [234](#), 235, 279
 __tag_mc_bmc_artifact: [277](#), 277, 290
 __tag_mc_bmc_artifact:n [277](#), 281, 291
 \l__tag_mc_botmarks_seq
 [96](#), 21, 90, 111,
 161, 208, 216, 217, 221, 234, 501, 518
 __tag_mc_check_parent_child:n .
 [122](#), 122, 181, 207, 343
 __tag_mc_disable_marks: [78](#), 78
 __tag_mc_emc: [158](#), [234](#), 236, 374
 __tag_mc_end_marks: .. [23](#), 63, 82, 375
 \l__tag_mc_firstmarks_seq
 [95](#), 21, 87, 110, 196, 199,
 200, 207, 208, 216, 233, 501, 509, 511
 \g__tag_mc_footnote_marks_seq ... [14](#)
 __tag_mc_get_marks: . [84](#), 84, 208, 229
 __tag_mc_handle_artifact:N
 [119](#), [277](#), 285, 325
 __tag_mc_handle_mc_label:n
 [27](#), 27, 200, 337
 __tag_mc_handle_mcid:nn
 [239](#), 260, 265, 331
 __tag_mc_handle_stash:n [50](#), [140](#),
 [142](#), 143, 168, 210, [266](#), 266, 276, 346
 __tag_mc_if_in: [68](#), 82, 226, 233
 __tag_mc_if_in:TF [68](#), 87, [226](#), 390, 398
 __tag_mc_if_in:p: [68](#), [226](#)
 __tag_mc_insert_extra_tmb:n ...
 [108](#), 108, 171
 __tag_mc_insert_extra_tme:n ...
 [108](#), 153, 172
 __tag_mc_insert_mcid_kids:n ...
 [131](#), 131, 150, 309
 __tag_mc_insert_mcid_single_
 kids:n [131](#), 136, 310
 \l__tag_mc_key_label_tl
 . [23](#), 198, 200, 316, 334, 335, 337, 424
 \l__tag_mc_key_properties_tl ...
 . [23](#), 177, 251, 266, 267, 281, 301,
 302, 333, 394, 403, 404, 409, 420, 421
 \l__tag_mc_key_stash_bool
 [20](#), 31, 40, 184, 203, 339

\g__tag_mc_key_tag_tl 19, [23](#),
 180, [225](#), [237](#), [243](#), [293](#), [315](#), [373](#), [390](#)
 \l__tag_mc_key_tag_tl [23](#), [179](#), [193](#),
 195, [224](#), [242](#), [314](#), [330](#), [332](#), [334](#), [389](#)
 \l__tag_mc_lang_tl
 [22](#), [185](#), [190](#), [316](#), [321](#)
 __tag_mc_lua_set_mc_type_attr:n
 [83](#), [83](#), [107](#), [195](#)
 __tag_mc_lua_unset_mc_type_-
 attr: [83](#), [109](#), [223](#)
 \g__tag_mc_main_marks_seq [14](#)
 \g__tag_mc_marks [13](#),
 25, [34](#), [47](#), [54](#), [65](#), [71](#), [88](#), [91](#), [197](#), [217](#)
 \g__tag_mc_multicol_marks_seq . . . [14](#)
 \g__tag_mc_parenttree_prop
 [17](#), [18](#), [103](#), [184](#), [272](#)
 \l__tag_mc_ref_abspage_tl [11](#)
 __tag_mc_set_label_used:n [31](#), [31](#), [51](#)
 \g__tag_mc_stack_seq
 [18](#), [89](#), [96](#), [106](#), [412](#)
 __tag_mc_store:nnn . . . [93](#), [93](#), [107](#), [134](#)
 \l__tag_mc_tmpa_tl [12](#)
 g__tag_MCID_abs_int [7](#)
 \g__tag_mode_lua_bool
 [35](#), [36](#), [135](#), [146](#), [248](#), [314](#), [317](#),
 336, [345](#), [614](#), [629](#), [641](#), [659](#), [718](#), [732](#)
 \l__tag_name_link_tl [759](#), [761](#), [762](#)
 __tag_pairs_prop [242](#)
 \l__tag_para_attr_class_tl
 [322](#), [399](#), [502](#)
 \g__tag_para_begin_int
 [322](#), [355](#), [373](#), [437](#), [560](#), [565](#)
 \l__tag_para_bool
 [322](#), [404](#), [413](#), [420](#), [426](#),
 459, [478](#), [511](#), [573](#), [574](#), [613](#), [640](#), [717](#)
 \g__tag_para_end_int
 [322](#), [363](#), [381](#), [448](#), [560](#), [566](#)
 \l__tag_para_flattened_bool
 [322](#), [409](#), [416](#), [429](#), [461](#), [486](#), [518](#)
 \l__tag_para_main_attr_class_tl
 [322](#), [492](#)
 \g__tag_para_main_begin_int
 [322](#), [351](#), [369](#), [551](#), [556](#)
 \g__tag_para_main_end_int
 [322](#), [359](#), [377](#), [551](#), [557](#)
 __tag_para_main_store_struct: .
 [391](#), [391](#), [468](#), [494](#)
 \g__tag_para_main_struct_tl [322](#), [393](#)
 \l__tag_para_main_tag_tl
 [322](#), [408](#), [415](#), [428](#), [466](#), [491](#)
 \l__tag_para_show_bool
 [322](#), [405](#), [406](#), [421](#), [434](#), [445](#)
 \l__tag_para_tag_default_tl [322](#)

\l__tag_para_tag_tl
 [322](#), [407](#), [414](#), [422](#), [427](#), [471](#), [501](#)
 \l__tag_parent_child_check_tl . .
 [156](#), [157](#), [169](#), [172](#), [500](#),
 618, [619](#), [626](#), [629](#), [733](#), [734](#), [746](#), [748](#)
 __tag_parenttree_add_objr:nn . .
 [163](#), [163](#), [491](#), [519](#)
 \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 [105](#), [105](#)
 __tag_pdf_object_ref [467](#)
 __tag_prop_gput:Nnn
 [9](#), [29](#), [89](#), [97](#), [98](#), [111](#),
 120, [121](#), [128](#), [132](#), [140](#), [143](#), [146](#),
 149, [151](#), [201](#), [205](#), [217](#), [220](#), [282](#),
 285, [314](#), [315](#), [384](#), [1308](#), [1444](#), [1451](#)
 __tag_prop_item:Nn [9](#), [52](#), [140](#), [147](#)
 __tag_prop_new:N [9](#), [9](#),
 11, [19](#), [24](#), [32](#), [108](#), [140](#), [140](#), [154](#), [1107](#)
 __tag_prop_new_linked:N
 [15](#), [17](#), [140](#), [141](#)
 __tag_prop_show:N [9](#), [65](#), [140](#), [149](#), [157](#)
 \c__tag_property_mc_clist . . . [80](#), [247](#)
 __tag_property_record:nn
 [29](#), [109](#), [109](#), [118](#), [243](#), [477](#), [728](#)
 __tag_property_ref_lastpage:nn
 [83](#), [119](#), [119](#), [160](#), [174](#), [177](#), [457](#), [471](#)
 \c__tag_property_struct_clist [80](#), [730](#)
 \l__tag_Ref_tmpa_tl [64](#)
 g__tag_role/RoleMap_dict [18](#)
 \g__tag_role_add_mathml_bool
 [73](#), [265](#), [762](#), [829](#)
 __tag_role_add_tag:nn
 [127](#), [127](#), [153](#), [280](#), [365](#), [800](#)
 __tag_role_add_tag:nnnn
 [167](#), [167](#), [226](#), [312](#), [805](#)
 __tag_role_alloctag:nnn [81](#),
 85, [95](#), [107](#), [117](#), [126](#), [141](#), [184](#), [277](#), [308](#)
 __tag_role_check_parent_-
 child:nnnnN [151](#),
 164, [571](#), [590](#), [592](#), [640](#), [717](#), [729](#), [742](#)
 \l__tag_role_debug_prop [11](#)
 __tag_role_get:nnNN [154](#),
 156, [164](#), [227](#), [229](#), [253](#), [712](#), [763](#), [1146](#)
 __tag_role_get_parent_child_-
 rule:nnN
 [196](#), [500](#), [503](#), [541](#), [589](#), [623](#), [701](#)
 \g__tag_role_index_prop
 [181](#), [10](#), [448](#), [456](#), [468](#),
 469, [470](#), [475](#), [481](#), [483](#), [484](#), [485](#),
 488, [490](#), [491](#), [495](#), [546](#), [547](#), [599](#), [609](#)
 \g__tag_role_NS_<ns>_class_prop [181](#)

<code>\g__tag_role_NS<ns>_prop</code>	181	<code>__tag_seq_new:N</code>	
<code>\g__tag_role_NS_mathml_prop</code>		267 , 486	<code>....</code>	9 , 9 , 22 , 109 , 140 , 142 , 155 , 1109	
<code>__tag_role_NS_new:nnn</code>		<code>__tag_seq_show:N</code>	9 , 58 , 140 , 148 , 156	
<code>....</code>	183 , 20 , 22 , 30 , 74 , 75 , 76 , 77 , 78 , 80		<code>__tag_show_spacemark</code>	478
<code>\g__tag_role_NS_prop</code>		<code>\l__tag_showspace_bool</code>	...	7 , 16 , 17
<code>....</code>	181 , 9 , 26 , 56 , 181 , 326 , 344 , 786		<code>\g__tag_softthyphen_bool</code>	
<code>\g__tag_role_parent_child-</code>			<code>....</code>	95 , 255 , 256 , 259 , 274 , 289 , 304	
<code>intarray</code>	390 , 397 , 406 , 421 , 425 , 555		<code>__tag_space_chars_shipout</code>	595
<code>__tag_role_read_namespace:n</code>	337 ,		<code>__tag_start_para_ints:</code>	
<code>337</code> , 341 , 342 , 343 , 345 , 347 , 349 , 350			<code>.....</code>	178 , 203 , 365 , 365	
<code>__tag_role_read_namespace:nn</code>	..		<code>__tag_stop_para_ints:</code>	
<code>.....</code>	318 , 318 , 339 , 348		<code>.....</code>	168 , 192 , 365 , 384	
<code>__tag_role_read_namespace-</code>			<code>__tag_store_parent_child-</code>		
<code>line:nw</code>	255 , 259 , 292 , 328	<code>rule:nnn</code>	391 , 393 , 417 , 462
<code>\l__tag_role_role_namespace-</code>			<code>g__tag_struct_1_prop</code>	107
<code>tmpa_tl</code>	12 ,	<code>__tag_struct_add_AF:nn</code>	
<code>757</code> , 778 , 783 , 787 , 790 , 794 , 809			<code>.....</code>	933 , 950 , 970 , 977 , 997 , 1042	
<code>\l__tag_role_role_tmpa_tl</code>		<code>__tag_struct_add_inline_AF:nn</code>	..	
<code>.....</code>	12 , 756 , 776 , 782 , 802 , 808		<code>....</code>	922 , 949 , 1011 , 1015 , 1022 , 1032	
<code>\g__tag_role_rolemap_prop</code>		<code>\l__tag_struct_addkid_tl</code>	86 , 772 , 1219	
<code>....</code>	181 , 18 , 144 , 146 , 149 , 158 ,		<code>\g__tag_struct_AFobj_int</code>	920 , 928 , 931	
<code>214</code> , 217 , 220 , 269 , 272 , 385 , 604 , 614			<code>__tag_struct_check_parent-</code>		
<code>\c__tag_role_rule_checkparent_tl</code>	157 , 173 , 619 , 734	<code>child:nn</code>	578 , 578 , 633 , 669 , 678 , 1206	
<code>\c__tag_role_rules_num_prop</code>		<code>__tag_struct_check_parent-</code>		
<code>.....</code>	391 , 514 , 564		<code>child_aux:nnnnN</code>	..	553 , 554 , 613 , 621
<code>\c__tag_role_rules_prop</code>	391 , 395 , 419		<code>\g__tag_struct_cont_mc_prop</code>	
<code>\l__tag_role_tag_namespace_tmpa-</code>			<code>.....</code>	11 , 95 , 96 , 98 , 101 , 244	
<code>tl</code>	12 , 755 , 807	<code>\g__tag_struct_dest_num_prop</code>	88 , 878	
<code>\l__tag_role_tag_namespace_tmpb-</code>			<code>\l__tag_struct_elem_stash_bool</code>	..	
<code>tl</code>	14	<code>.....</code>	85 , 732 , 1169 , 1202 , 1232	
<code>\l__tag_role_tag_namespace_tmpb-</code>			<code>__tag_struct_exchange_kid-</code>		
<code>tluuuuu%</code>	12	<code>command:N</code>	305 , 305 , 314 , 345
<code>\l__tag_role_tag_tmpa_tl</code>		<code>__tag_struct_fill_kid_key:n</code>	...	
<code>.....</code>	12 , 754 , 775 , 801 , 806		<code>.....</code>	136 , 315 , 315 , 447	
<code>\g__tag_role_tags_class_prop</code>	...		<code>__tag_struct_format_P:nnN</code>	409
<code>....</code>	181 , 8 , 90 , 99 , 112 , 121 , 137 , 268		<code>__tag_struct_format_parentnum:nnN</code>	412 , 412
<code>\g__tag_role_tags_NS_prop</code>		<code>__tag_struct_format_parentrole:nnN</code>	409 , 410
<code>181</code> , 7 , 88 , 97 , 110 , 119 , 130 , 322 ,			<code>__tag_struct_format_Ref</code>	137
357 , 383 , 423 , 685 , 701 , 752 , 781 , 1277			<code>__tag_struct_format_Ref:nnN</code>	416 , 416	
<code>\l__tag_role_tmpa_seq</code>	12	<code>__tag_struct_format_rolemap:nnN</code>	409 , 409
<code>\l__tag_role_update_bool</code>		<code>.....</code>	409 , 411	
<code>.....</code>	208 , 255 , 256 , 264 , 344 , 346		<code>__tag_struct_get_dict_content:nN</code>	138 , 395 , 395 , 448
<code>\c__tag_role_userNS_id_str</code>		<code>__tag_struct_get_id:n</code>	
<code>.....</code>	182 , 59 , 80		<code>..</code>	96 , 101 , 114 , 115 , 148 , 149 , 454 , 538	
<code>\g__tag_root_default_tl</code>	285	<code>__tag_struct_get_role:nnNN</code>	
<code>\g__tag_saved_in_mc_bool</code>		<code>.....</code>	146 , 159 , 195 , 195 ,	
<code>....</code>	610 , 619 , 634 , 646 , 664 , 723 , 737		<code>214</code> , 561 , 566 , 638 , 650 , 662 , 724 , 737		
<code>__tag_seq_gput_left:Nn</code>		<code>__tag_struct_gput_data_attribute:nn</code>	1437 , 1437
<code>.....</code>	9 , 40 , 145 , 153 , 268				
<code>__tag_seq_gput_right:Nn</code>	9 ,			
<code>35</code> , 140 , 144 , 152 , 231 , 241 , 252 , 291					
<code>__tag_seq_item:Nn</code>	...	9 , 47 , 140 , 146			

__tag_struct_gput_data_ref:nn .	__tag_struct_set_attribute: ...
..... 1419, 1436 23, 37, 1154, 1268
__tag_struct_gput_data_ref-	__tag_struct_set_tag_info:nnn .
aux:nnn 159, 161, 175, 194, 1131
.. 1398, 1399, 1421, 1425, 1429, 1433	\g__tag_struct_stack_current_tl
__tag_struct_gput_data_ref-	.. 16, 29, 31, 38, 69, 75, 104, 148,
dest:nn	154, 162, 208, 270, 274, 310, 344,
__tag_struct_gput_data_ref-	533, 538, 544, 1153, 1217, 1221,
label:nn	1222, 1243, 1261, 1267, 1306, 1312,
__tag_struct_gput_data_ref-	1318, 1324, 1351, 1357, 1363, 1369
num:nn	\l__tag_struct_stack_parent-
__tag_struct_insert_annot:nn ..	tmpa_tl .. 16, 471, 480, 497, 742,
..... 462, 462, 1461	1129, 1136, 1140, 1180, 1207, 1214,
__tag_struct_insert_annot-	1218, 1220, 1223, 1235, 1236, 1244
shipout:nnn	\g__tag_struct_stack_seq
__tag_struct_kid_mc_gput- 12, 22, 25, 470, 723,
right:nn ... 215, 227, 228, 247, 269	736, 1139, 1145, 1155, 1254, 1259, 1265
__tag_struct_kid_OBJR_gput-	\c__tag_struct_StructElem-
right:nnn 280, 280, 283, 304, 478, 506	entries_seq
__tag_struct_kid_struct_gput- 39
left:nn	\c__tag_struct_StructTreeRoot-
264, 264, 265, 279	entries_seq
__tag_struct_kid_struct_gput- 39
right:nn	\g__tag_struct_tag_NS_tl 76, 695,
..... 248, 248, 249, 263, 1305, 1350	711, 714, 718, 1134, 1148, 1242, 1279
g__tag_struct_kids_l_seq	\g__tag_struct_tag_stack_seq ...
..... 107 14, 50, 249,
\g__tag_struct_label_num_prop ..	250, 573, 588, 602, 1151, 1258, 1273
..... 84, 726, 865	\g__tag_struct_tag_tl
\l__tag_struct_lang_tl 76,
..... 579, 1093, 1117, 1122	179, 180, 183, 314, 315, 419, 420,
__tag_struct_mcid_dict:n	694, 696, 710, 713, 717, 719, 1133,
..... 98, 101, 215, 234	1147, 1152, 1275, 1277, 1319, 1364
\c__tag_struct_null_tl	__tag_struct_use_check_parent-
..... 10, 349	child:nn . 634, 634, 681, 1323, 1368
\g__tag_struct_objR_seq	__tag_struct_write_obj
..... 8 137
\l__tag_struct_parenttag_NS_tl .	__tag_struct_write_obj:n
..... 76, 762, 765, 769, 1175 151, 428, 428
\l__tag_struct_parenttag_tl	\l__tag_tag_stop_int 159, 163, 164,
..... 76, 761, 764, 768, 770, 1175	172, 173, 180, 187, 188, 197, 198, 206
__tag_struct_prop_gput:nnn .. 93,	\g__tag_tagunmarked_bool 94, 251, 253
94, 95, 101, 111, 116, 121, 126,	\l__tag_tmp_unused_tl 63, 130, 315,
131, 138, 164, 168, 177, 183, 188,	322, 395, 398, 402, 419, 422, 423,
351, 364, 378, 784, 796, 810, 826,	685, 688, 701, 704, 752, 755, 788, 1550
841, 849, 914, 936, 979, 998, 1043,	\l__tag_tmp_unused_tl\l__-
1113, 1119, 1124, 1156, 1171, 1184,	tag_Ref_tmpa_tl
1194, 1210, 1353, 1414, 1524, 1575 60
\g__tag_struct_ref_by_dest_prop . 91	\l__tag_tmpa_box
__tag_struct_Ref_dest:nN . 855, 876 60, 171, 177, 178, 182, 193, 194
__tag_struct_Ref_label:nN 855, 863	\l__tag_tmpa_clist
__tag_struct_Ref_num:nN .. 855, 889 60, 1502, 1503, 1536, 1537, 1539
__tag_struct_Ref_obj:nN .. 855, 855	\l__tag_tmpa_int
\g__tag_struct_roletag_NS_tl 76 60,
\l__tag_struct_roletag_NS_tl ...	90, 93, 98, 101, 105, 114, 430, 442, 444
..... 79, 1150, 1160, 1198	\l__tag_tmpa_prop 60, 176, 189, 203, 205
\l__tag_struct_roletag_tl	\l__tag_tmpa_seq 51, 58, 59, 60, 319,
..... 76, 1149, 1152, 1160, 1162, 1198	321, 323, 324, 325, 326, 443, 446,
	454, 455, 457, 458, 459, 466, 486,
	496, 687, 691, 694, 695, 703, 707,

710, 711, 754, 758, 761, 762, 774, 775, 776, 1504, 1508, 1518, 1519, 1520, 1522, 1540, 1546, 1548, 1572	_tag_tree_write_idtree: .. 86, 361
\l__tag_tmpa_str	_tag_tree_write_namespaces: 322, 322, 373
.... 42, 43, 48, 60, 262, 267, 272, 297, 302, 309, 399, 404, 416, 421, 780, 787, 792, 799, 806, 813, 819, 820, 822, 825, 829, 837, 844, 910, 917	_tag_tree_write_parenttree: 246, 246, 357
\l__tag_tmpa_tl	_tag_tree_write_rolemap: 263, 263, 365
42, 43, 47, 49, 50, 51, 56, 60, 86, 88, 93, 94, 96, 98, 102, 106, 106, 108, 109, 113, 114, 116, 118, 119, 137, 138, 139, 141, 143, 144, 146, 177, 178, 180, 183, 184, 186, 191, 198, 199, 205, 205, 206, 209, 211, 214, 215, 220, 268, 269, 271, 275, 277, 288, 297, 299, 300, 302, 306, 307, 308, 308, 308, 311, 314, 345, 347, 349, 357, 376, 448, 453, 455, 455, 456, 457, 458, 463, 468, 469, 470, 475, 481, 483, 488, 514, 516, 525, 546, 549, 556, 564, 566, 575, 599, 601, 602, 604, 606, 606, 610, 620, 624, 645, 653, 655, 656, 658, 662, 667, 698, 702, 715, 717, 723, 725, 736, 738, 766, 768, 771, 772, 774, 932, 935, 1258, 1259, 1265, 1267, 1273, 1276, 1277, 1279, 1346, 1440, 1442, 1443, 1447, 1510, 1516, 1527, 1554	_tag_tree_write_structelements: 147, 147, 377
\l__tag_tmpb_box	_tag_tree_write_structtreeroot: 126, 126, 381
..... 60, 172, 179, 180, 184, 186	\g__tag_unique_cnt_int
\l__tag_tmpb_seq	97, 1064, 1068, 1071, 1081, 1085, 1089
..... 60, 1503, 1504, 1539, 1540	_tag_whatsits:
\l__tag_tmpb_tl 193, 60, 89, 104, 118, 120, 295, 301, 432, 456, 462, 484, 490, 518, 547, 550, 556, 568, 609, 611, 614, 616, 621, 625, 672, 680, 682, 683, 685, 689, 694, 699, 703, 716, 718, 767, 769, 865, 869, 878, 882 36, 43, 48, 49, 52, 295, 296
\l__tag_tmpc_tl	tag-namespace_(rolemap-key) 752
60, 485, 491	tag/check/parent-child 183
__tag_tree_fill_parenttree: ...	tag/check/parent-child-end 183
..... 171, 172, 253	tag/struct/1 internal commands:
__tag_tree_final_checks: 20, 20, 354	__tag/struct/1 31
\g__tag_tree_id_pad_int .. 78, 82, 154	tag/tree/namespaces internal commands:
__tag_tree_lua_fill_parenttree:	__tag/tree/namespaces 321
..... 233, 233, 250	tag/tree/parenttree internal commands:
\g__tag_tree_openaction_struct_-	__tag/tree/parenttree 154
tl 32, 38, 57	tag/tree/rolemap internal commands:
__tag_tree_parenttree_rerun_-	__tag/tree/rolemap 262
msg: 171, 220, 255	tagabspace 8, 123
__tag_tree_update_openaction: .	tagmcabs 8, 123
..... 42, 75	\tagmcbegin 41, 180, 22
__tag_tree_write_classmap:	\tagmcend 41, 22
..... 286, 286, 369	tagmcid 8, 123
	\tagmcifinTF 41, 39
	\tagmcuse 41, 22
	\tagpdfparaOff 43, 570
	\tagpdfparaOn 43, 570
	\tagpdfsetup .. 8, 41, 68, 114, 115, 180, 6
	\tagpdfsuppressmarks 43, 575
	\tagstart 7, 183, 214
	\tagstop 7, 182, 213
	tagstruct 8, 123
	\tagstructbegin 41, 148, 180, 45, 288
	\tagstructend 41, 45, 289
	tagstructobj 8, 123
	\tagstructuse 41, 45
	tagtag@LastPage internal commands:
	\r__tagtag@LastPage 57
	\tagtool 41, 13
	tagunmarked (deprecated) (key) ... 1, 251
	test/lang_(setup-key) 577
	TeX and L ^A T _E X 2 _ε commands:
	\@M 168
	\@bsphack 111
	\@esphack 113
	\@gobble 31, 55

\UseExpandableTaggingSocket .	44, 70, <u>72</u>	vbox commands:	
\UseSocket	<u>44</u>	\vbox_set_split_to_ht:NNn	194
\UseStructureName .	338, 339, 752, 754, 774	\vbox_set_to_ht:Nnn	170
\UseTaggingSocket	44,	\vbox_unpack_drop:N	183
	45, 69, <u>72</u> , 806, 813, 820, 827, 834,	\vfuzz	169
	841, 847, 854, 860, 867, 875, 888,	viewer/startstructure_(setup-key) . .	<u>34</u>
	899, 912, 923, 936, 947, 960, 970, 983		
V		W	
\vbadness	168, 192	wrong-pdfversion	<u>220</u>