

# tagpdf – L<sup>A</sup>T<sub>E</sub>X kernel code for PDF tagging<sup>\*</sup>

Ulrike Fischer<sup>†</sup>

Released 2026-04-24

## Contents

<b>I</b>	<b>The tagpdf main module</b>	<b>7</b>
1	Initialization and test if pdfmanagement is active.	8
2	base package	9
3	Package options	9
4	Packages	10
	4.1 a LastPage label . . . . .	10
5	Variables	10
6	Variants of l3 commands	12
7	Label and Reference commands	12
8	Setup label attributes	13
9	Commands to fill seq and prop	13
10	General tagging commands	14
11	Handling link artifacts	15
12	Keys for tagpdfsetup	16
13	loading of engine/more dependent code	18
<b>II</b>	<b>The tagpdf-checks module</b>	
	Messages and check code	<b>20</b>
1	Commands	<b>20</b>

---

<sup>\*</sup>This file describes v1.0b, last revised 2026-04-24.

<sup>†</sup>E-mail: [fischer@troubleshooting-tex.de](mailto:fischer@troubleshooting-tex.de)

<b>2</b>	<b>Description of log messages</b>	<b>20</b>
2.1	\ShowTagging command . . . . .	20
2.2	Messages in checks and commands . . . . .	21
2.3	Messages from the ptagging code . . . . .	21
2.4	Warning messages from the lua-code . . . . .	21
2.5	Info messages from the lua-code . . . . .	21
2.6	Debug mode messages and code . . . . .	22
2.7	Messages . . . . .	22
<b>3</b>	<b>Messages</b>	<b>24</b>
3.1	Messages related to mc-chunks . . . . .	24
3.2	Messages related to structures . . . . .	25
3.3	Attributes . . . . .	27
3.4	Roles . . . . .	27
3.5	Miscellaneous . . . . .	30
<b>4</b>	<b>Retrieving data</b>	<b>31</b>
<b>5</b>	<b>PDF version check</b>	<b>31</b>
<b>6</b>	<b>User conditionals</b>	<b>32</b>
<b>7</b>	<b>Internal checks</b>	<b>33</b>
7.1	checks for active tagging . . . . .	33
7.2	Checks related to structures . . . . .	33
7.3	Checks related to roles . . . . .	35
7.4	Check related to mc-chunks . . . . .	35
7.5	Checks related to the state of MC on a page or in a split stream . . . . .	38
7.6	Benchmarks . . . . .	41
<b>III The tagpdf-user module</b>		
<b>Code related to L<sup>A</sup>T<sub>E</sub>X2e user commands and document commands</b>		<b>42</b>
<b>1</b>	<b>Setup commands</b>	<b>42</b>
<b>2</b>	<b>Commands related to mc-chunks</b>	<b>42</b>
<b>3</b>	<b>Commands related to structures</b>	<b>42</b>
<b>4</b>	<b>Debugging</b>	<b>43</b>
<b>5</b>	<b>Extension commands</b>	<b>43</b>
5.1	Fake space . . . . .	43
5.2	Tagging of paragraphs . . . . .	44
5.3	Header and footer . . . . .	44
5.4	Link tagging . . . . .	45
<b>6</b>	<b>Socket support</b>	<b>45</b>
<b>7</b>	<b>User commands and extensions of document commands</b>	<b>46</b>

<b>8</b>	<b>Setup and preamble commands</b>	<b>46</b>
<b>9</b>	<b>Commands for the mc-chunks</b>	<b>46</b>
<b>10</b>	<b>Commands for the structure</b>	<b>47</b>
<b>11</b>	<b>Tagging Socket support</b>	<b>48</b>
<b>12</b>	<b>Debugging</b>	<b>49</b>
<b>13</b>	<b>Commands to extend document commands</b>	<b>52</b>
13.1	Document structure . . . . .	52
13.2	Structure destinations . . . . .	53
13.3	Fake space . . . . .	53
13.4	Paratagging . . . . .	54
13.5	Language support . . . . .	58
13.6	Header and footer . . . . .	58
13.7	Links . . . . .	61
13.8	Attaching css-files for derivation . . . . .	66
<b>IV</b>	<b>The <code>tagpdf-tree</code> module</b>	
	<b>Commands trees and main dictionaries</b>	<b>68</b>
<b>1</b>	<b>Trees, pdfmanagement and finalization code</b>	<b>68</b>
1.1	Check structure . . . . .	68
1.2	Catalog: MarkInfo and StructTreeRoot and OpenAction . . . . .	69
1.3	Writing the IDtree . . . . .	70
1.4	Writing structure elements . . . . .	71
1.5	ParentTree . . . . .	72
1.6	Rolemap dictionary . . . . .	75
1.7	Classmap dictionary . . . . .	76
1.8	Namespaces . . . . .	76
1.9	Finishing the structure . . . . .	77
1.10	StructParents entry for Page . . . . .	78
<b>V</b>	<b>The <code>tagpdf-mc-shared</code> module</b>	
	<b>Code related to Marked Content (mc-chunks), code shared by all modes</b>	<b>79</b>
<b>1</b>	<b>Public Commands</b>	<b>79</b>
<b>2</b>	<b>Public keys</b>	<b>80</b>
<b>3</b>	<b>Marked content code – shared</b>	<b>81</b>
3.1	Variables and counters . . . . .	81
3.2	Functions . . . . .	82
3.3	Keys . . . . .	86

<b>VI</b>	<b>The <code>tagpdf-mc-generic</code> module</b>	
	<b>Code related to Marked Content (<code>mc-chunks</code>), generic mode</b>	<b>88</b>
<b>1</b>	<b>Marked content code – generic mode</b>	<b>88</b>
1.1	Variables . . . . .	88
1.2	Functions . . . . .	89
1.3	Looking at MC marks in boxes . . . . .	92
1.4	Keys . . . . .	98
<b>VII</b>	<b>The <code>tagpdf-mc-luacode</code> module</b>	
	<b>Code related to Marked Content (<code>mc-chunks</code>), luamode-specific</b>	<b>101</b>
<b>1</b>	<b>Marked content code – luamode code</b>	<b>101</b>
1.1	Commands . . . . .	103
1.2	Key definitions . . . . .	107
<b>VIII</b>	<b>The <code>tagpdf-struct</code> module</b>	
	<b>Commands to create the structure</b>	<b>110</b>
<b>1</b>	<b>Public Commands</b>	<b>110</b>
<b>2</b>	<b>Public keys</b>	<b>111</b>
2.1	Keys for the structure commands . . . . .	111
2.2	Setup keys . . . . .	113
<b>3</b>	<b>Variables</b>	<b>113</b>
3.1	Variables used by the keys . . . . .	116
3.2	Variables used by tagging code of basic elements . . . . .	116
<b>4</b>	<b>Commands</b>	<b>117</b>
4.1	Initialization of the <code>StructTreeRoot</code> . . . . .	117
4.2	Adding the <code>/ID</code> key . . . . .	118
4.3	Filling in the tag info . . . . .	119
4.4	Handlings kids . . . . .	120
4.5	Output of the object . . . . .	124
4.6	Commands for the parent-child checks . . . . .	129
<b>5</b>	<b>Keys</b>	<b>132</b>
<b>6</b>	<b>User commands</b>	<b>140</b>
<b>7</b>	<b>Attributes and attribute classes</b>	<b>149</b>
7.1	Variables . . . . .	149
7.2	Commands and keys . . . . .	150
<b>IX</b>	<b>The <code>tagpdf</code> driver for <code>luatex</code></b>	<b>153</b>
<b>1</b>	<b>Loading the lua</b>	<b>153</b>

<b>2</b>	<b>User commands to access data</b>	<b>157</b>
<b>3</b>	<b>Logging functions</b>	<b>158</b>
<b>4</b>	<b>Helper functions</b>	<b>160</b>
4.1	Retrieve data functions . . . . .	160
4.2	Functions to insert the pdf literals . . . . .	162
<b>5</b>	<b>Function for the real space chars</b>	<b>164</b>
<b>6</b>	<b>Function for the tagging</b>	<b>168</b>
<b>7</b>	<b>Parenttree</b>	<b>173</b>
<b>8</b>	<b>parent-child rules</b>	<b>175</b>
<b>9</b>	<b>Link annotations</b>	<b>178</b>
<b>X</b>	<b>The tagpdf-roles module</b>	
	<b>Tags, roles and namespace code</b>	<b>180</b>
<b>1</b>	<b>Code related to roles and structure names</b>	<b>180</b>
1.1	Variables . . . . .	181
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
<b>XI</b>	<b>The tagpdf-space module</b>	
	<b>Code related to real space chars</b>	<b>203</b>
<b>1</b>	<b>Code for interword spaces</b>	<b>203</b>
	<b>Index</b>	<b>207</b>

Ulrike Fischer  
Version 1.0b, released 2026-04-24

## Part I

# The tagpdf main module

---

<code>\tag_suspend:n</code>	<code>\tag_suspend:n {&lt;label&gt;}</code>
<code>\tag_resume:n</code>	<code>\tag_resume:n {&lt;label&gt;}</code>
<code>\tag_stop:n</code>	<code>\tag_stop:n {&lt;label&gt;} (deprecated)</code>
<code>\tag_start:n</code>	<code>\tag_start:n {&lt;label&gt;} (deprecated)</code>

---

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 special cases, e.g. for debugging. `activate-tree (deprecated) (setup key)`

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

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

`activate/struct-dest (setup key)` 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 lead to disappearing hyphens.
- **artifact** (default value) This keeps the hard hyphen U+002D but surrounds it by an Artifact-MC. This only works if the document is tagged.
- **true**, on do the same as **artifact**.

If tagging is not 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.

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

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

---

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

---

These are attributes used by the label/ref system.

## 1 Initialization and test if pdfmanagement is active.

```
1 <@@=tag>
2 <*package>
3 \ProvidesExplPackage {tagpdf} {2026-04-24} {1.0b}
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} {2026-04-24} {1.0b}
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} {2026-04-24} {1.0b}
32      {part of tagpdf - provide base, no-op versions of the user commands }
33      \</base>

```

## 3 Package options

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

```

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

```

## 4 Packages

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

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

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

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

### 4.1 a LastPage label

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

```
56 \end{package}
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

A few temporary variables

```

\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
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

```

(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 {tagabspace,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. \l\_\_tag\_active\_struct\_bool TODO: check if they are used everywhere as needed and as wanted. \l\_\_tag\_active\_socket\_bool

```

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 used 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_softthyphen_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_softthyphen_bool
96 \bool_gset_true:N \g__tag_softthyphen_bool

```

(End of definition for `\g__tag_softthyphen_bool`.)

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

```

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   \@bsphack
112   \property_record:nn{#1}{#2}
113   \@esphack
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   { tagabspace } { 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
__tag_seq_new:N
__tag_prop_gput:Nnn
__tag_seq_gput_right:Nn
__tag_seq_item:cn
__tag_prop_item:cn
__tag_seq_show:N
__tag_prop_show:N
140 \cs_set_eq:NN __tag_prop_new:N \prop_new:N
141 \cs_set_eq:NN __tag_prop_new_linked:N \prop_new_linked:N
142 \cs_set_eq:NN __tag_seq_new:N \seq_new:N
143 \cs_set_eq:NN __tag_prop_gput:Nnn \prop_gput:Nnn
144 \cs_set_eq:NN __tag_seq_gput_right:Nn \seq_gput_right:Nn
145 \cs_set_eq:NN __tag_seq_gput_left:Nn \seq_gput_left:Nn
146 \cs_set_eq:NN __tag_seq_item:cn \seq_item:cn
147 \cs_set_eq:NN __tag_prop_item:cn \prop_item:cn
148 \cs_set_eq:NN __tag_seq_show:N \seq_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 \endpackage

```

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

## 10 General tagging commands

`\tag_suspend:n` We need commands to stop tagging in some places. They switch local booleans and also stop the counting of paragraphs. The commands keep track of the nesting with a local counter. Tagging only is only restarted at the outer level, if the current level is 1. The `\tag_start:` commands with argument allow to give a label. This is only used in debugging messages to allow to follow the nesting. The label is not expand so can e.g. be a single command token.

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

```

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 Handling link artifacts

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

```

220 <*package>
221 \cs_new_protected:Npn \__tag_activate_linkbin:
222 {
223   \sys_if_engine luatex:T
224   {
225     \tagstructbegin{tag=\UseStructureName{link},stash}
226     \tl_const:Ne \c__tag_struct_link_bin_tl {\int_use:N\c@g__tag_struct_abs_int}%
227     \tagstructend
228     \AddToHook{tagpdf/finish/before}[tagpdf/linkbin]
229     {
230       \par

```

```

231         \int_compare:nNtT {\directlua{tex.sprint(1tx.__tag.func.linkbin())}}>{0}
232         {
233             \tagstructbegin{tag=Artifact,title=Ignored-link-annotations}
234             \tag_struct_use_num:o{\c__tag_struct_link_bin_tl}
235             \tagstructend
236         }
237     }
238 }
239 }
240 \AddToHook{begindocument/end}{tagpdf/linkbin}{\__tag_activate_linkbin:}
241 \end{package}

```

## 12 Keys for tagpdfsetup

TODO: the log-levels must be sorted

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

```

242 <*\package>
243 \keys_define:nn { __tag / setup }
244 {
245     activate/mc      .bool_gset:N = \g__tag_active_mc_bool,
246     activate/tree    .bool_gset:N = \g__tag_active_tree_bool,
247     activate/struct  .bool_gset:N = \g__tag_active_struct_bool,
248     activate/all     .meta:n =
249         {activate/mc={#1},activate/tree={#1},activate/struct={#1}},
250     activate/all     .default:n = true,
251     activate/struct-dest .bool_gset:N = \g__tag_active_struct_dest_bool,

```

old, deprecated names

```

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

```

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

```

259     debug/show      .choice:,

```

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

```

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

```



```

269     debug/log / all      .code:n = {\int_set:Nn \l__tag_loglevel_int { 10 }},
270     debug/uncompress .code:n = { \pdf_uncompress: },

```

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

```

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

```

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

```

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

```

deprecated name

```

275     tagunmarked .bool_gset:N = \g__tag_tagunmarked_bool,

```

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

```

276     activate/softhyphen .choice:,
277     activate/softhyphen/false .code:n = {\bool_gset_false:N \g__tag_softhyphen_bool},
278     activate/softhyphen/off .code:n = {\bool_gset_false:N \g__tag_softhyphen_bool},
279     activate/softhyphen/char .code:n =
280     {
281         \bool_gset_true:N \g__tag_softhyphen_bool
282         \sys_if_engine luatex:T
283         {
284             \lua_now:e
285             {
286                 tex.setattribute
287                 (
288                     luatexbase.attributes.l__tag_softhyphen_attr,
289                     1
290                 )
291             }
292         }
293     },
294     activate/softhyphen/artifact .code:n =
295     {
296         \bool_gset_true:N \g__tag_softhyphen_bool
297         \sys_if_engine luatex:T
298         {
299             \lua_now:e
300             {
301                 tex.setattribute
302                 (
303                     luatexbase.attributes.l__tag_softhyphen_attr,
304                     -2147483647
305                 )
306             }
307         }
308     },
309     activate/softhyphen/true .code:n =
310     {
311         \bool_gset_true:N \g__tag_softhyphen_bool
312         \sys_if_engine luatex:T
313         {

```

```

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

```

collect-link-artifacts (*setup key*)

```

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

```

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

```

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

```

deprecated name

```

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

```

## 13 loading of engine/more dependent code

```

354 \sys_if_engine luatex:T
355 {

```

```

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

```

Ulrike Fischer  
 Version 1.0b, released 2026-04-24

## Part II

# The tagpdf-checks module

## Messages and check code

### 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

---

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

---

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

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

## 2.2 Messages in checks and commands

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

## 2.3 Messages from the ptagging code

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

## 2.4 Warning messages from the lua-code

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

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

## 2.5 Info messages from the lua-code

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

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

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

## 2.6 Debug mode messages and code

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

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

## 2.7 Messages

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

<u>struct-unknown</u> <u>struct-no-objnum</u> <u>struct-orphan</u> <u>struct-faulty-nesting</u> <u>struct-missing-tag</u> <u>struct-used-twice</u> <u>struct-label-unknown</u> <u>struct-show-closing</u>	Various messages related to structure. Check the definition in the code for their meaning and the arguments they take.
<u>tree-struct-still-open</u>	Message issued at the end of the compilation if there are (beside Root) other open structures on the stack.
<u>tree-statistic</u>	Message issued at the end of the compilation showing the number of objects to write
<u>show-struct</u> <u>show-kids</u>	These two messages are used in debug mode to show the current structures in the log and terminal.
<u>attr-unknown</u>	Message if an attribute is unknown.
<u>role-missing</u> <u>role-unknown</u> <u>role-unknown-tag</u> <u>role-unknown-NS</u> <u>role-tag</u> <u>new-tag</u> <u>role-parent-child-result</u> <u>role-remapping</u>	Messages related to role mapping.
<u>tree-mcid-index-wrong</u>	Used in the tree code, typically indicates the document must be rerun.
<u>sys-no-interwordspace</u>	Message if an engine doesn't support inter word spaces
<u>para-hook-count-wrong</u>	<p>Message if the number of begin paragraph and end paragraph differ. This normally means faulty structure.</p> <pre> 1 &lt;@@=tag&gt; 2 &lt;*header&gt; 3 \ProvidesExplPackage {tagpdf-checks-code} {2026-04-24} {1.0b} 4 {part of tagpdf - code related to checks, conditionals, debugging and messages} 5 &lt;/header&gt; </pre>

## 3 Messages

### 3.1 Messages related to mc-chunks

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

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

*(End of definition for mc-nested. This function is documented on page 22.)*

**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 22.)*

**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 22.)*

**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 22.)*

**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 22.)*

**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 22.)*

**mc-current** Informational messages about current mc state.

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

*(End of definition for mc-current. This function is documented on page 22.)*



## 3.2 Messages related to structures

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

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

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

**struct-no-objnum** Should not happen ...

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

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

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

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

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

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

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

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

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

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

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

**struct-used-twice**

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

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

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

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

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

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

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

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

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

```

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

```

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

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

```

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

```

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

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

```

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

```

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

The following messages are only needed in debug mode.

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

**show-kids**

```

64 <*>debug>
65 \msg_new:nnn { tag/debug } { show-struct }
66 {
67     =====\\
68     The~structure~#1~
69     \tl_if_empty:nTF {#2}
70     { is~empty \>~ . }
71     { contains: #2 }
72     \\
73 }
74 \msg_new:nnn { tag/debug } { show-kids }
75 {
76     The~structure~has~the~following~kids:
77     \tl_if_empty:nTF {#2}
78     { \>~ NONE }
79     { #2 }
80     \\
81     =====

```

```

82     }
83   </debug>

```

(End of definition for `show-struct` and `show-kids`. These functions are documented on page 23.)

### 3.3 Attributes

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

`attr-unknown`

```

84 <*package>
85 \msg_new:nnn { tag } {attr-unknown} { attribute-#1-is-unknown}

```

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

### 3.4 Roles

`role-missing` Warning message if either the tag or the role is missing

`role-unknown` `\msg_new:nnn { tag } {role-missing} { tag~#1-has-no-role-assigned }`

`role-unknown-tag` `\msg_new:nnn { tag } {role-unknown} { role~#1-is-not-known }`

`role-unknown-NS` `\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 23.)

`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 23.)

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

```

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

```

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

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

```

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

```

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

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

```

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

```

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

\\_tag\_check\_forbidden\_parent\_child:nnnn

```

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

```

```

148     {role-struct-parent-child-forbidden}
149     { #2 }
150     { #3 }
151     {
152         \exp_last_unbraced:No \use_ii:nn { \l__tag_get_parent_tmpc_tl }
153         :
154         \exp_last_unbraced:No \use_i:nn { \l__tag_get_parent_tmpc_tl }
155     }
156     {
157         \exp_last_unbraced:No \use_ii:nn { \l__tag_get_child_tmpc_tl }
158         :
159         \exp_last_unbraced:No \use_i:nn { \l__tag_get_child_tmpc_tl }
160     }
161 }
162 }
163 \cs_generate_variant:Nn\__tag_check_struct_forbidden_parent_child:nnn{onn}
(End of definition for \__tag_check_forbidden_parent_child:nnnn.)

```

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

```

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

```

**\\_\_tag\_check\_unresolved\_parent\_child:nnnn**

```

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

```

**tag/check/parent-child** **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 23.)*

**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 23.)*

### 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 23.)

**sys-no-interwordspace** Currently only pdf<sub>l</sub>atex and lua<sub>l</sub>atex 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 23.)

**\\_\_tag\_check\_typeout\_v:n** A simple logging function. By default is gobbles its argument, but the log-keys sets it to typeout.

```

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

```

(End of definition for \\_\_tag\_check\_typeout\_v:n.)

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

```

231 \msg_new:nnnn { tag } {para-hook-count-wrong}
232   {The~number~of~automatic~begin~(#1)~and~end~(#2)~#3~para-hooks~differ!}
233   {This~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 23.)

## 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 20.)

## 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 20.)

`\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.  
`\tag_if_box_tagged:` *NTF*

```

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\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 20.)



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

```

289 <*package>
290 \prg_new_conditional:Npnn\tag_if_in:n #1 {T,F,TF}
291 {
292   \seq_if_in:NnTF \g__tag_struct_roletag_stack_seq {#1}
293   {\prg_return_true:}
294   {\prg_return_false:}
295 }
296 </package>

```

*(End of definition for \tag\_if\_in:n. This function is documented on page ??.)*

## 7 Internal checks

These are checks used in various places in the code.

### 7.1 checks for active tagging

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

```

\__tag_check_if_active_struct:TF
297 <*package>
298 \prg_new_conditional:Npnn \__tag_check_if_active_mc: {T,F,TF}
299 {
300   \bool_lazy_and:nnTF { \g__tag_active_mc_bool } { \l__tag_active_mc_bool }
301   {
302     \prg_return_true:
303   }
304   {
305     \prg_return_false:
306   }
307 }
308 \prg_new_conditional:Npnn \__tag_check_if_active_struct: {T,F,TF}
309 {
310   \bool_lazy_and:nnTF { \g__tag_active_struct_bool } { \l__tag_active_struct_bool }
311   {
312     \prg_return_true:
313   }
314   {
315     \prg_return_false:
316   }
317 }

```

*(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.

```

318 \cs_new_protected:Npn \__tag_check_structure_has_tag:n #1 %#1 struct num
319 {
320   \prop_get:cnNF

```

```

321     { g__tag_struct_#1_prop }
322     {S}
323     \l__tag_tmp_unused_t1
324     {
325         \msg_error:nn { tag } {struct-missing-tag}
326     }
327 }

```

(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.

```

328 \cs_new_protected:Npn \__tag_check_structure_tag:N #1
329 {
330     \prop_get:NoNF \g__tag_role_tags_NS_prop {#1}\l__tag_tmp_unused_t1
331     {
332         \msg_warning:nne { tag } {role-unknown-tag} {#1}
333     }
334 }

```

(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.

```

335 \cs_new_protected:Npn \__tag_check_info_closing_struct:n #1 %#1 struct num
336 {
337     \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
338     {
339         \msg_info:nnn { tag } {struct-show-closing} {#1}
340     }
341 }

```

```

342
343 \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.

```

344 \cs_new_protected:Npn \__tag_check_no_open_struct:
345 {
346     \msg_error:nn { tag } {struct-faulty-nesting}
347 }

```

(End of definition for \\_\_tag\_check\_no\_open\_struct:.)

\\_\_tag\_check\_struct\_used:n This checks if a stashed structure has already been used.

```

348 \cs_new_protected:Npn \__tag_check_struct_used:n #1 %#1 label
349 {
350     \prop_get:cnNT
351     {g__tag_struct_ \property_ref:enn{tagpdfstruct-#1}{tagstruct}{unknown}_prop}
352     {parentnum}
353     \l__tag_tmpa_t1
354     {
355         \msg_warning:nnn { tag } {struct-used-twice} {#1}
356     }
357 }

```

(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.

```

358 \\cs_new_protected:Npn \\_tag\_check\_add\_tag\_role:nn #1 #2 % #1 tag, #2 role
359 {
360   \\tl\_if\_empty:nTF {#2}
361   {
362     \\msg\_error:nnn { tag } {role-missing} {#1}
363   }
364   {
365     \\prop\_get:NnNTF \\g\_tag\_role\_tags\_NS\_prop {#2} \\l\_tag\_tmpa\_tl
366     {
367       \\int\_compare:nNnT {\\l\_tag\_loglevel\_int} > { 0 }
368       {
369         \\msg\_info:nnnn { tag } {role-tag} {#1} {#2}
370       }
371     }
372     {
373       \\msg\_error:nnn { tag } {role-unknown} {#2}
374     }
375   }
376 }

```

Similar with a namespace

```

377 \\cs_new_protected:Npn \\_tag\_check\_add\_tag\_role:nnn #1 #2 #3 % #1 tag/NS, #2 role #3 namespace
378 {
379   \\tl\_if\_empty:nTF {#2}
380   {
381     \\msg\_error:nnn { tag } {role-missing} {#1}
382   }
383   {
384     \\prop\_get:cnNTF { g\_tag\_role\_NS\_#3\_prop } {#2} \\l\_tag\_tmpa\_tl
385     {
386       \\int\_compare:nNnT {\\l\_tag\_loglevel\_int} > { 0 }
387       {
388         \\msg\_info:nnnn { tag } {role-tag} {#1} {#2/#3}
389       }
390     }
391     {
392       \\msg\_error:nnn { tag } {role-unknown} {#2/#3}
393     }
394   }
395 }

```

*(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).

```

396 \\cs_new_protected:Npn \\_tag\_check\_mc\_if\_nested:
397 {
398   \\_tag\_mc\_if\_in:T
399   {
400     \\msg\_warning:nne { tag } {mc-nested} { \\_tag\_get\_mc\_abs\_cnt: }

```

```

401     }
402   }
403
404   \cs_new_protected:Npn \__tag_check_mc_if_open:
405   {
406     \__tag_mc_if_in:F
407     {
408       \msg_warning:nne { tag } {mc-not-open} { \__tag_get_mc_abs_cnt: }
409     }
410   }

```

(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.

```

411   \cs_new_protected:Npn \__tag_check_mc_pushed_popped:nn #1 #2
412   {
413     \int_compare:nNnT
414     { \l__tag_loglevel_int } = { 2 }
415     { \msg_info:nne {tag}{mc-#1}{#2} }
416     \int_compare:nNnT
417     { \l__tag_loglevel_int } > { 2 }
418     {
419       \msg_info:nne {tag}{mc-#1}{#2}
420       \seq_log:N \g__tag_mc_stack_seq
421     }
422   }

```

(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.

```

423   \cs_new_protected:Npn \__tag_check_mc_tag:N #1 % #1 is var with a tag name in it
424   {
425     \tl_if_empty:NTF #1
426     {
427       \tl_set:No #1 { \g__tag_struct_tag_tl }
428       \msg_info:nnee { tag } {mc-tag-missing} { \g__tag_struct_tag_tl } { \__tag_get_mc_abs_
429     }
430     {
431       \prop_get:NoNF \g__tag_role_tags_NS_prop {#1}\l__tag_tmp_unused_tl
432       {
433         \msg_warning:nne { tag } {role-unknown-tag} {#1}
434       }
435     }
436   }

```

(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??

```

437 \cs_new_protected:Npn \__tag_check_init_mc_used:
438 {
439   \intarray_new:Nn \g__tag_check_mc_used_intarray { 65536 }
440   \cs_gset_eq:NN \__tag_check_init_mc_used: \prg_do_nothing:
441 }

```

(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.

```

442 \cs_new_protected:Npn \__tag_check_mc_used:n #1 %#1 mcid absnt
443 {
444   \int_compare:nNnT {\l__tag_loglevel_int} > { 2 }
445   {
446     \__tag_check_init_mc_used:
447     \intarray_gset:Nnn \g__tag_check_mc_used_intarray
448       {#1}
449     { \intarray_item:Nn \g__tag_check_mc_used_intarray {#1} + 1 }
450     \int_compare:nNnT
451     {
452       \intarray_item:Nn \g__tag_check_mc_used_intarray {#1}
453     }
454     >
455     { 1 }
456     {
457       \msg_warning:nnn { tag } {mc-used-twice} {#1}
458     }
459   }
460 }

```

(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.

```

461 \cs_new_protected:Npn \__tag_check_show_MCID_by_page:
462 {
463   \tl_set:Ne \l__tag_tmpa_tl
464   {
465     \__tag_property_ref_lastpage:nn
466       {abspage}
467       {-1}
468   }
469   \int_step_inline:nnnn {1}{1}
470   {
471     \l__tag_tmpa_tl
472   }
473   {
474     \seq_clear:N \l__tag_tmpa_seq
475     \int_step_inline:nnnn
476       {1}
477       {1}
478       {
479         \__tag_property_ref_lastpage:nn
480           {tagmcabs}
481           {-1}

```

```

482     }
483     {
484         \int_compare:nT
485         {
486             \property_ref:enn
487             {mcid-###1}
488             {tagabspage}
489             {-1}
490             =
491             ##1
492         }
493         {
494             \seq_gput_right:Ne \l__tag_tmpa_seq
495             {
496                 Page##1-###1-
497                 \property_ref:enn
498                 {mcid-###1}
499                 {tagmcid}
500                 {-1}
501             }
502         }
503     }
504     \seq_show:N \l__tag_tmpa_seq
505 }
506 }

```

*(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.

```

507 \prg_new_conditional:Npnn \__tag_check_if_mc_in_galley: { T,F,TF }
508 {
509     \tl_if_eq:NNTF \l__tag_mc_firstmarks_seq \l__tag_mc_botmarks_seq
510     { \prg_return_false: }
511     { \prg_return_true: }
512 }

```

*(End of definition for \\_\_tag\_check\_mc\_in\_galley:TF.)*

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

```

513 \prg_new_conditional:Npnn \__tag_check_if_mc_tmb_missing: { T,F,TF }
514 {

```

```

515 \bool_if:nTF
516 {
517   \str_if_eq_p:ee {\seq_item:Nn \l__tag_mc_firstmarks_seq {1}}{e-}
518   ||
519   \str_if_eq_p:ee {\seq_item:Nn \l__tag_mc_firstmarks_seq {1}}{b+}
520 }
521 { \prg_return_true: }
522 { \prg_return_false: }
523 }

```

(End of definition for \\_\_tag\_check\_if\_mc\_tmb\_missing:TF.)

\\_\_tag\_check\_if\_mc\_tme\_missing\_p: This checks if a extra bottom mark (“extra-tme”) is needed. According to the analysis this the case if the botmarks starts with b+. Like above we assume that the marks content is already in the seq’s.

\\_\_tag\_check\_if\_mc\_tme\_missing:TF

```

524 \prg_new_conditional:Npnn \__tag_check_if_mc_tme_missing: { T,F,TF }
525 {
526   \str_if_eq:eeTF {\seq_item:Nn \l__tag_mc_botmarks_seq {1}}{b+}
527   { \prg_return_true: }
528   { \prg_return_false: }
529 }

```

(End of definition for \\_\_tag\_check\_if\_mc\_tme\_missing:TF.)

530 </package>

531 <\*debug>

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

```

532 \msg_new:nnn { tag / debug } {mc-begin} { MC-begin~#1-with-options:~\tl_to_str:n{#2}~[\msg_line_context:] }
533 \msg_new:nnn { tag / debug } {mc-end} { MC-end~#1~[\msg_line_context:] }
534
535 \cs_new_protected:Npn \__tag_debug_mc_begin_insert:n #1
536 {
537   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
538   {
539     \msg_note:nnnn { tag / debug } {mc-begin} {inserted} { #1 }
540   }
541 }
542 \cs_new_protected:Npn \__tag_debug_mc_begin_ignore:n #1
543 {
544   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
545   {
546     \msg_note:nnnn { tag / debug } {mc-begin } {ignored} { #1 }
547   }
548 }
549 \cs_new_protected:Npn \__tag_debug_mc_end_insert:
550 {
551   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
552   {
553     \msg_note:nnn { tag / debug } {mc-end} {inserted}
554   }
555 }
556 \cs_new_protected:Npn \__tag_debug_mc_end_ignore:
557 {

```

```

558 \int_compare:nNnT { \l__tag_loglevel_int } > {0}
559 {
560     \msg_note:nnn { tag / debug } {mc-end } {ignored}
561 }
562 }

```

And now something for the structures

```

563 \msg_new:nnn { tag / debug } {struct-begin}
564 {
565     Struct~\tag_get:n{struct_num}~begin~#1~with~options:~\tl_to_str:n{#2}~\[\msg_line_context
566 }
567 \msg_new:nnn { tag / debug } {struct-end}
568 {
569     Struct~end~#1~[\msg_line_context:]
570 }
571 \msg_new:nnn { tag / debug } {struct-end-wrong}
572 {
573     Struct~end~'#1'~doesn't~fit~start~'#2'~[\msg_line_context:]
574 }
575
576 \cs_new_protected:Npn \__tag_debug_struct_begin_insert:n #1
577 {
578     \int_compare:nNnT { \l__tag_loglevel_int } > {0}
579     {
580         \msg_note:nnnn { tag / debug } {struct-begin} {inserted} { #1 }
581         \seq_log:N \g__tag_struct_tag_stack_seq
582     }
583 }
584 \cs_new_protected:Npn \__tag_debug_struct_begin_ignore:n #1
585 {
586     \int_compare:nNnT { \l__tag_loglevel_int } > {0}
587     {
588         \msg_note:nnnn { tag / debug } {struct-begin } {ignored} { #1 }
589     }
590 }
591 \cs_new_protected:Npn \__tag_debug_struct_end_insert:
592 {
593     \int_compare:nNnT { \l__tag_loglevel_int } > {0}
594     {
595         \msg_note:nnn { tag / debug } {struct-end} {inserted}
596         \seq_log:N \g__tag_struct_tag_stack_seq
597     }
598 }
599 \cs_new_protected:Npn \__tag_debug_struct_end_ignore:
600 {
601     \int_compare:nNnT { \l__tag_loglevel_int } > {0}
602     {
603         \msg_note:nnn { tag / debug } {struct-end } {ignored}
604     }
605 }
606 \cs_new_protected:Npn \__tag_debug_struct_end_check:n #1
607 {
608     \int_compare:nNnT { \l__tag_loglevel_int } > {0}
609     {
610         \seq_get:NNT \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl

```



```

611     {
612       \str_if_eq:eeF
613       {#1}
614       {\exp_last_unbraced:No \use_i:nn { \l__tag_tmpa_tl }}
615       {
616         \msg_warning:nnee { tag/debug }{ struct-end-wrong }
617         {#1}
618         {\exp_last_unbraced:No \use_i:nn { \l__tag_tmpa_tl }}
619       }
620     }
621   }
622 }

```

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.

```

623 \msg_new:nnn { tag / debug } {tag-suspend}
624 {
625   \int_if_zero:nTF
626   {#1}
627   {Tagging~suspended}
628   {Tagging~(not)~suspended~(already~inactive)}\
629   level:~#1~==>~\int_eval:n{#1+1}\tl_if_empty:nF{#2}{,~label:~#2}~[\msg_line_context:]
630 }
631 \msg_new:nnn { tag / debug } {tag-resume}
632 {
633   \int_if_zero:nTF
634   {#1}
635   {Tagging~resumed}
636   {Tagging~(not)~resumed}\
637   level:~\int_eval:n{#1+1}~==>~#1\tl_if_empty:nF{#2}{,~label:~#2}~[\msg_line_context:]
638 }
639 </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?

```

640 <*package>
641 \cs_new_protected:Npn \__tag_check_benchmark_tic:{
642 \cs_new_protected:Npn \__tag_check_benchmark_toc:{
643 \cs_new_protected:Npn \tag_check_benchmark_on:
644 {
645   \cs_if_exist:NT \benchmark_tic:
646   {
647     \cs_set_eq:NN \__tag_check_benchmark_tic: \benchmark_tic:
648     \cs_set_eq:NN \__tag_check_benchmark_toc: \benchmark_toc:
649   }
650 }
651 </package>

```

Ulrike Fischer  
Version 1.0b, released 2026-04-24

## Part III

# The **tagpdf-user** module

## Code related to L<sup>A</sup>T<sub>E</sub>X2e user commands and document commands

### 1 Setup commands

---

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

---

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

---

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

---

### 2 Commands related to mc-chunks

---

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

---

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

---

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

---

This is a wrapper around `\tag_mc_if_in:TF`. and tests if an mc is open or not. It is mostly of importance for pdf<sub>l</sub>atex as lua<sub>l</sub>atex doesn't mind much if a mc tag is not correctly closed. Unlike the expl3 command it is not expandable.

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

### 3 Commands related to structures

---

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

---

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

## 4 Debugging

---

<code>\ShowTagging</code>	<code>\ShowTagging{&lt;key-val&gt;}</code>
---------------------------	--

---

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

---

<code>mc-data (show-key)</code>	<code>mc-data = &lt;number&gt;</code>
---------------------------------	---------------------------------------

---

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

---

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

---

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

---

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

---

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

---

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

---

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

---

<code>debug/structures (show-key)</code>	<code>debug/structures = &lt;structure number&gt;</code>
--	--

---

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

## 5 Extension commands

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

The commands and keys should be viewed as experimental!

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

### 5.1 Fake space

---

<code>\pdffakespace</code>	(lua-only) This provides a lua-version of the <code>\pdffakespace</code> primitive of pdftex.
----------------------------	---

---

## 5.2 Tagging of paragraphs

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

---

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

---

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

---

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

---

---

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

---

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

## 5.3 Header and footer

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

---

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

---

## 5.4 Link tagging

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

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

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

## 6 Socket support

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

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

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

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

Usually, these sockets have (beside the default plug defined for every socket) one additional plug defined and directly assigned. This plug is used when tagging is active. There may be more plugs, e.g., tagging with special debugging or special behaviour depending on the class or PDF version etc., but right now it is usually just on or off.

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

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

For special cases like in some `\halign` contexts we need a fully expandable version of the command. For these cases, `\UseExpandableTaggingSocket` can be used. To allow being expandable, it does not output any debugging information if `\DebugSocketsOn` is in effect and therefore should be avoided whenever possible.

The L3 programming layer versions `\tag_socket_use_expandable:n`, `\tag_socket_use:n`, and `\tag_socket_use:nn`, `\tag_socket_use:nnn` are slightly more efficient than `\UseTaggingSocket` because they do not have to determine how many arguments the socket takes when disabling it.

## 7 User commands and extensions of document commands

```

1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-user} {2026-04-24} {1.0b}
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 42.)*

`\tag_tool:n`

This command is deprecated and will be removed.

`\tagtool` (deprecated)

```

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

```

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

## 9 Commands for the mc-chunks

`\tagmcbegin`

`\tagmcend`

`\tagmcuse`

```

22 <*base>
23 \NewDocumentCommand \tagmcbegin { m }
24   {
25     \tag_mc_begin:n {#1}
26   }
27
28

```

```

29 \NewDocumentCommand \tagmcend { }
30 {
31   \tag_mc_end:
32 }
33
34 \NewDocumentCommand \tagmcuse { m }
35 {
36   \tag_mc_use:n {#1}
37 }
38 \</base>

```

*(End of definition for \tagmcbegin, \tagmcend, and \tagmcuse. These functions are documented on page 42.)*

**\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<sub>l</sub>atex 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 42.)*

## 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 42.)*

## 11 Tagging Socket support

```

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

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

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

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

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

83 \cs_set_protected:Npn \UseTaggingSocket #1
84 {
85   \bool_if:NTF \l__tag_active_socket_bool
86   { \socket_use:nw {tagsupport/#1} }
87   {
88     \int_case:nnF
89     { \int_use:c { c__socket_tagsupport/#1_args_int } }
90     {
91       0 \prg_do_nothing:
92       1 \use_none:n
93       2 \use_ii:nn
94     }
95     \ERRORusetaggingsocket
96   }
97 }

98 \cs_set:Npn \UseExpandableTaggingSocket #1
99 {
100   \bool_if:NTF \l__tag_active_socket_bool
101   { \socket_use_expandable:nw {tagsupport/#1} }
102   {
103     \int_case:nnF
104     { \int_use:c { c__socket_tagsupport/#1_args_int } }
105     {
106       0 \prg_do_nothing:

```



```

107             1 \use_none:n
108             2 \use_ii:nn

```

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

```

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

```

*(End of definition for \tag\_socket\_use:n and others. These functions are documented on page 45.)*

## 12 Debugging

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

```

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

```

*(End of definition for \ShowTagging. This function is documented on page 43.)*

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

```

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

```

*(End of definition for mc-data (show-key). This function is documented on page 43.)*

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

```

132 \keys_define:nn { __tag / show }
133 { mc-current .code:n =
134     {
135         \bool_if:NTF \g__tag_mode_lua_bool
136         {
137             \int_compare:nNnTF
138             { -2147483647 }
139             =
140             {
141                 \lua_now:e

```

```

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

```

*(End of definition for mc-current (show-key). This function is documented on page 43.)*

**mc-marks (show-key)** It maps the mc-marks into the sequences and then shows them. This allows to inspect the

first and last mc-Mark on a page. It should only be used in the shipout (header/footer).

```

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

```

*(End of definition for mc-marks (show-key). This function is documented on page 43.)*

#### struct-stack (show-key)

```

235 \keys_define:nn { __tag / show }
236 {
237   struct-stack .choice:
238   ,struct-stack / log .code:n = \seq_log:N \g__tag_struct_tag_stack_seq
239   ,struct-stack / show .code:n = \seq_show:N \g__tag_struct_tag_stack_seq
240   ,struct-stack .default:n = show
241 }

```

242 `\package`

(End of definition for `struct-stack (show-key)`. This function is documented on page 43.)

`debug/structures (show-key)`

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

243 `\*debug`

244 `\keys_define:nn { __tag / show }`

245 `{`

246 `,debug/structures .code:n =`

247 `{`

248 `\int_step_inline:nnn{#1}{\c@g__tag_struct_abs_int}`

249 `{`

250 `\msg_term:nneeee`

251 `{ tag/debug } { show-struct }`

252 `{ ##1 }`

253 `{`

254 `\prop_map_function:cN`

255 `{g__tag_struct_debug_##1_prop}`

256 `\msg_show_item_unbraced:nn`

257 `}`

258 `{ } { }`

259 `\msg_term:nneeee`

260 `{ tag/debug } { show-kids }`

261 `{ ##1 }`

262 `{`

263 `\seq_map_function:cN`

264 `{g__tag_struct_debug_kids_##1_seq}`

265 `\msg_show_item_unbraced:n`

266 `}`

267 `{ } { }`

268 `}`

269 `}`

270 `,debug/structures .default:n = 1`

271 `}`

272 `\debug`

(End of definition for `debug/structures (show-key)`. This function is documented on page 43.)

## 13 Commands to extend document commands

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

273 `\*package`

### 13.1 Document structure

`\g__tag_root_default_tl`

`activate (setup-key)`

`activate/socket (setup-key)`

274 `\tl_new:N\g__tag_root_default_tl`

275 `\tl_gset:Nn\g__tag_root_default_tl {Document}`

276

```

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

```

(End of definition for `\g__tag_root_default_tl`, `activate (setup-key)`, and `activate/socket (setup-key)`. These functions are documented on page 42.)

## 13.2 Structure destinations

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

```

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

```

## 13.3 Fake space

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

```

303 \bool_if:NT \g__tag_mode_lua_bool
304 {
305   \NewDocumentCommand\pdffakespace { }
306   {
307     \__tag_fakespace:
308   }
309 }
310 \providecommand\pdffakespace{}

```

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

## 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.

At first some variables.

```

\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
311 </package>
312 <base>\bool_new:N \l__tag_para_flattened_bool
313 <base>\bool_new:N \l__tag_para_bool
314 <*package>
315 \int_new:N \g__tag_para_begin_int
316 \int_new:N \g__tag_para_end_int
317 \int_new:N \g__tag_para_main_begin_int
318 \int_new:N \g__tag_para_main_end_int
this will hold the structure number of the current text-unit.
319 \tl_new:N \g__tag_para_main_struct_tl
320 \tl_gset:Nn \g__tag_para_main_struct_tl {1}
321 \tl_new:N \l__tag_para_tag_default_tl
322 \tl_new:N \l__tag_para_tag_tl
323 \tl_set:Nn \l__tag_para_tag_tl { \l__tag_para_tag_default_tl }
324 \tl_new:N \l__tag_para_main_tag_tl
325 \tl_set:Nn \l__tag_para_tag_default_tl { \UseStructureName {para/textblock} }
326 \tl_set:Nn \l__tag_para_main_tag_tl { \UseStructureName {para/semantic} }
327 %\tl_new:N \l__tag_para_attr_class_tl % defined in lttagging
328 \tl_new:N \l__tag_para_main_attr_class_tl

```

(End of definition for \l\_\_tag\_para\_bool and others.)

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

```

\__tag_gincr_para_main_begin_int:
\__tag_gincr_para_main_end_int:
\__tag_gincr_para_begin_int:
\__tag_gincr_para_end_int:
329 \cs_new_protected:Npn \__tag_gincr_para_main_begin_int:
330 {
331   \int_gincr:N \g__tag_para_main_begin_int
332 }
333 \cs_new_protected:Npn \__tag_gincr_para_begin_int:
334 {
335   \int_gincr:N \g__tag_para_begin_int
336 }
337 \cs_new_protected:Npn \__tag_gincr_para_main_end_int:
338 {
339   \int_gincr:N \g__tag_para_main_end_int
340 }
341 \cs_new_protected:Npn \__tag_gincr_para_end_int:
342 {
343   \int_gincr:N \g__tag_para_end_int
344 }

```

(End of definition for \\_\_tag\_gincr\_para\_main\_begin\_int: and others.)

```

\__tag_start_para_ints:
\__tag_stop_para_ints:
345 \cs_new_protected:Npn \__tag_start_para_ints:
346 {
347   \cs_set_protected:Npn \__tag_gincr_para_main_begin_int:
348   {
349     \int_gincr:N \g__tag_para_main_begin_int

```

```

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

```

(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:

```

371 \cs_new:Npn \__tag_para_main_store_struct:
372 {
373     \tl_gset:Ne \g__tag_para_main_struct_tl {\int_use:N \c@g__tag_struct_abs_int }
374 }

```

(End of definition for \\_\_tag\_para\_main\_store\_struct:.)

**para/tagging (setup-key)**

para/tag (setup-key)

para/maintag (setup-key)

para/tagging (tool-key)

para/tag (tool-key)

para/maintag (tool-key)

para/flattened (tool-key)

unittag (deprecated)

para-flattened (deprecated)

**paratagging (deprecated)**

paratagging-show (deprecated)

paratag (deprecated)

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

```

375 \keys_define:nn { __tag / setup }
376 {
377     para/tagging      .bool_set:N = \l__tag_para_bool,
378     debug/show/para   .code:n = {\bool_set_true:N \l__tag_para_show_bool},
379     debug/show/paraOff .code:n = {\bool_set_false:N \l__tag_para_show_bool},
380     para/tag          .tl_set:N   = \l__tag_para_tag_tl,
381     para/maintag      .tl_set:N   = \l__tag_para_main_tag_tl,
382     para/flattened    .bool_set:N = \l__tag_para_flattened_bool
383 }

```

deprecated key definitions:

```

384 \keys_define:nn { tag / tool }
385 {
386     para/tagging      .bool_set:N = \l__tag_para_bool,

```

```

387   para/tag      .tl_set:N = \l__tag_para_tag_tl,
388   para/maintag .tl_set:N = \l__tag_para_main_tag_tl,
389   para/flattened .bool_set:N = \l__tag_para_flattened_bool
390 }

```

the deprecated names

```

391 \keys_define:nn { __tag / setup }
392 {
393   paratagging      .bool_set:N = \l__tag_para_bool,
394   paratagging-show .bool_set:N = \l__tag_para_show_bool,
395   paratag          .tl_set:N   = \l__tag_para_tag_tl
396 }

```

still needed as used in block code ...

```

397 \keys_define:nn { tag / tool}
398 {
399   para      .bool_set:N = \l__tag_para_bool,
400   paratag .tl_set:N = \l__tag_para_tag_tl,
401   unittag .tl_set:N = \l__tag_para_main_tag_tl,
402   para-flattened .bool_set:N = \l__tag_para_flattened_bool
403 }

```

(End of definition for para/tagging (setup-key) and others. These functions are documented on page 44.)

Helper command for debugging:

```

404 \cs_new_protected:Npn \__tag_check_para_begin_show:nn #1 #2
405   {%#1 color, #2 prefix
406   {
407     \bool_if:NT \l__tag_para_show_bool
408     {
409       \tag_mc_begin:n{artifact}
410       \llap{\color_select:n{#1}\tiny#2\int_use:N\g__tag_para_begin_int\ }
411       \tag_mc_end:
412     }
413   }
414
415 \cs_new_protected:Npn \__tag_check_para_end_show:nn #1 #2
416   {%#1 color, #2 prefix
417   {
418     \bool_if:NT \l__tag_para_show_bool
419     {
420       \tag_mc_begin:n{artifact}
421       \rlap{\color_select:n{#1}\tiny\ #2\int_use:N\g__tag_para_end_int}
422       \tag_mc_end:
423     }
424   }
425 \AddToHook{begindocument/before}[tagpdf/para]
426 {
427   \AddToHook{para/begin}[tagpdf]{ \tag_socket_use:n { para/begin } }
428   \AddToHook{para/end} [tagpdf] { \tag_socket_use:n { para/end } }
429 }

```



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

```

430 \AddToHook{enddocument/info}
431 {
432   \tag_if_active:F
433   {
434     \msg_redirect_name:nnn { tag } { para-hook-count-wrong } { warning }
435   }
436   \int_compare:nNnF {\g__tag_para_main_begin_int}={\g__tag_para_main_end_int}
437   {
438     \msg_error:nneee
439     {tag}
440     {para-hook-count-wrong}
441     {\int_use:N\g__tag_para_main_begin_int}
442     {\int_use:N\g__tag_para_main_end_int}
443     {text-unit}
444   }
445   \int_compare:nNnF {\g__tag_para_begin_int}={\g__tag_para_end_int}
446   {
447     \msg_error:nneee
448     {tag}
449     {para-hook-count-wrong}
450     {\int_use:N\g__tag_para_begin_int}
451     {\int_use:N\g__tag_para_end_int}
452     {text}
453   }
454 }
</package>

```

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

```

455 <base>\newcommand\tagpdfparaOn {}
456 <base>\newcommand\tagpdfparaOff{}
457 <*package>
458 \renewcommand\tagpdfparaOn {\bool_set_true:N \l__tag_para_bool}
459 \renewcommand\tagpdfparaOff{\bool_set_false:N \l__tag_para_bool}

```

(End of definition for **\tagpdfparaOn** and **\tagpdfparaOff**. These functions are documented on page 44.)

**\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}%

```

```

460 \NewDocumentCommand\tagpdfsuppressmarks{m}
461   {{\use:c{__tag_mc_disable_marks:} #1}}

```

(End of definition for \tagpdfsuppressmarks. This function is documented on page 44.)

## 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)

```

462 \keys_define:nn { __tag / setup }
463   {
464     text / lang .tl_set:N = \l__tag_struct_lang_tl
465   }

```

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

## 13.6 Header and footer

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

```

466 \cs_new_protected:Npn \__tag_hook_kernel_before_head:{}
467 \cs_new_protected:Npn \__tag_hook_kernel_after_head:{}
468 \cs_new_protected:Npn \__tag_hook_kernel_before_foot:{}
469 \cs_new_protected:Npn \__tag_hook_kernel_after_foot:{}

```

We use the page sockets.

```

470 \NewTaggingSocketPlug{build/page/header}{tagpdf}
471   {
472     \__tag_hook_kernel_before_head:
473     #2
474     \__tag_hook_kernel_after_head:
475   }
476
477 \AssignTaggingSocketPlug{build/page/header}{tagpdf}
478 \NewTaggingSocketPlug{build/page/footer}{tagpdf}
479   {
480     \__tag_hook_kernel_before_foot:
481     #2
482     \__tag_hook_kernel_after_foot:
483   }
484 \AssignTaggingSocketPlug{build/page/footer}{tagpdf}
485
486 \bool_new:N \g__tag_saved_in_mc_bool
487 \cs_new_protected:Npn \__tag_exclude_headfoot_begin:
488   {
489     \bool_set_false:N \l__tag_para_bool
490     \bool_if:NTF \g__tag_mode_lua_bool
491     {
492       \tag_mc_end_push:
493     }
494     {
495       \bool_gset_eq:NN \g__tag_saved_in_mc_bool \g__tag_in_mc_bool
496       \bool_gset_false:N \g__tag_in_mc_bool

```

```

496     }
497     \tag_mc_begin:n {artifact}
498     \tag_suspend:n{headfoot}
499   }
500   \cs_new_protected:Npn \__tag_exclude_headfoot_end:
501   {
502     \tag_resume:n{headfoot}
503     \tag_mc_end:
504     \bool_if:NTF \g__tag_mode_lua_bool
505     {
506       \tag_mc_begin_pop:n{ }
507     }
508     {
509       \bool_gset_eq:NN \g__tag_in_mc_bool \g__tag_saved_in_mc_bool
510     }
511   }

```

This version allows to use an Artifact structure

```

512   \__tag_attr_new_entry:nn {__tag/attr/pagination}{/0/Artifact/Type/Pagination}
513   \cs_new_protected:Npn \__tag_exclude_struct_headfoot_begin:n #1
514   {
515     \bool_set_false:N \l__tag_para_bool
516     \bool_if:NTF \g__tag_mode_lua_bool
517     {
518       \tag_mc_end_push:
519     }
520     {
521       \bool_gset_eq:NN \g__tag_saved_in_mc_bool \g__tag_in_mc_bool
522       \bool_gset_false:N \g__tag_in_mc_bool
523     }
524     \tag_struct_begin:n{tag=Artifact,attribute-class=__tag/attr/#1}
525     \tag_mc_begin:n {artifact=#1}
526     \tag_suspend:n{headfoot}
527   }
528
529   \cs_new_protected:Npn \__tag_exclude_struct_headfoot_end:
530   {
531     \tag_resume:n{headfoot}
532     \tag_mc_end:
533     \tag_struct_end:
534     \bool_if:NTF \g__tag_mode_lua_bool
535     {
536       \tag_mc_begin_pop:n{ }
537     }
538     {
539       \bool_gset_eq:NN \g__tag_in_mc_bool \g__tag_saved_in_mc_bool
540     }
541   }

```

And now the keys

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

exclude-header-footer (deprecated)

```

542   \keys_define:nn { __tag / setup }
543   {
544     page/exclude-header-footer .choice:,

```

```

545 page/exclude-header-footer / true .code:n =
546 {
547   \cs_set_eq:NN \__tag_hook_kernel_before_head: \__tag_exclude_headfoot_begin:
548   \cs_set_eq:NN \__tag_hook_kernel_before_foot: \__tag_exclude_headfoot_begin:
549   \cs_set_eq:NN \__tag_hook_kernel_after_head: \__tag_exclude_headfoot_end:
550   \cs_set_eq:NN \__tag_hook_kernel_after_foot: \__tag_exclude_headfoot_end:
551 },
552 page/exclude-header-footer / pagination .code:n =
553 {
554   \cs_set:Nn \__tag_hook_kernel_before_head: { \__tag_exclude_struct_headfoot_begin:n {p
555   \cs_set:Nn \__tag_hook_kernel_before_foot: { \__tag_exclude_struct_headfoot_begin:n {p
556   \cs_set_eq:NN \__tag_hook_kernel_after_head: \__tag_exclude_struct_headfoot_end:
557   \cs_set_eq:NN \__tag_hook_kernel_after_foot: \__tag_exclude_struct_headfoot_end:
558 },
559 page/exclude-header-footer / false .code:n =
560 {
561   \cs_set_eq:NN \__tag_hook_kernel_before_head: \prg_do_nothing:
562   \cs_set_eq:NN \__tag_hook_kernel_before_foot: \prg_do_nothing:
563   \cs_set_eq:NN \__tag_hook_kernel_after_head: \prg_do_nothing:
564   \cs_set_eq:NN \__tag_hook_kernel_after_foot: \prg_do_nothing:
565 },
566 page/exclude-header-footer .default:n = true,
567 page/exclude-header-footer .initial:n = true,

```

deprecated name

```

568   exclude-header-footer .meta:n = { page/exclude-header-footer = {#1} }
569 }

```

(End of definition for page/exclude-header-footer (setup-key) and exclude-header-footer (deprecated).  
These functions are documented on page 44.)

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

```

570 \AtBeginDocument
571 {
572   \NewTaggingSocketPlug{parbox/before}{tag/footer}
573   {
574     \tag_struct_begin:n{tag=Span}
575     \tag_mc_begin:n{ }
576   }
577
578   \NewTaggingSocketPlug{parbox/after}{tag/footer}
579   {
580     \tag_mc_end:
581     \tag_struct_end:
582   }
583 }
584 \cs_new_protected:Npn \__tag_headfoot_tagged_begin:n #1
585 {
586   \AssignTaggingSocketPlug{parbox/before}{tag/footer}
587   \AssignTaggingSocketPlug{parbox/after}{tag/footer}
588   \bool_set_false:N \l__tag_para_bool
589   \bool_if:NTF \g__tag_mode_lua_bool
590   {

```

```

591     \tag_mc_end_push:
592   }
593   {
594     \bool_gset_eq:NN \g__tag_saved_in_mc_bool \g__tag_in_mc_bool
595     \bool_gset_false:N \g__tag_in_mc_bool
596   }
597   \tag_struct_begin:n{tag=Artifact,attribute-class=__tag/attr/#1,parent=\tag_get:n{currentt
598 }
599
600 \cs_new_protected:Npn \__tag_headfoot_tagged_end:
601 {
602   \tag_struct_end:
603   \bool_if:NTF \g__tag_mode_lua_bool
604   {
605     \tag_mc_begin_pop:n{ }
606   }
607   {
608     \bool_gset_eq:NN \g__tag_in_mc_bool \g__tag_saved_in_mc_bool
609   }
610 }
611 \keys_define:nn { __tag / setup }
612 {
613   page/tag-header-footer .choice:,
614   page/tag-header-footer/artifact .code:n =
615   {
616     \cs_set:Nn \__tag_hook_kernel_before_head: { \__tag_headfoot_tagged_begin:n {pagination
617     \cs_set:Nn \__tag_hook_kernel_before_foot: { \__tag_headfoot_tagged_begin:n {pagination
618     \cs_set_eq:NN \__tag_hook_kernel_after_head: \__tag_headfoot_tagged_end:
619     \cs_set_eq:NN \__tag_hook_kernel_after_foot: \__tag_headfoot_tagged_end:
620   },
621   page/tag-header-footer .default:n = artifact,
622 }

```

## 13.7 Links

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

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

```

623 \AddToHookNext{class/before}
624 {
625   \cs_if_exist:NF \l__tag_name_link_tl
626   {
627     \tl_new:N \l__tag_name_link_tl
628     \tl_set:Nn \l__tag_name_link_tl{Link}
629   }
630 }

```

Tagging sockets for links

```

631 \socket_if_exist:nF {tagsupport/link/before}

```

```

632 {
633   \NewTaggingSocket{link/before}{1}
634   \NewTaggingSocket{link/after}{1}
635 }
636 \NewTaggingSocketPlug{link/before}{kernel}
637 {
638   \mode_leave_vertical:
639   \tag_mc_end_push:
640   \tag_struct_begin:n { tag=\UseStructureName{link} }
641   \tag_mc_begin:n {}
642   #1
643 }
644 \AssignTaggingSocketPlug{link/before}{kernel}
645
646 \NewTaggingSocketPlug{link/after}{kernel}
647 {
648   #1
649   \tag_mc_end:
650   \tag_struct_end:
651   \tag_mc_begin_pop:n{ }
652 }
653 \AssignTaggingSocketPlug{link/after}{kernel}
654
655
656 \bool_lazy_and:nnTF
657 { \sys_if_engine luatex_p: }
658 {
659   \tl_if_empty_p:e
660   {
661     \lua_now:e
662     { if~ luatexbase.in_callback('pre_shipout_filter','linksplit')~
663       then~else~tex.print('1')~end
664     }
665   }
666 }
667 {
668   \hook_gput_code:nnn
669   {pdfannot/link/URI/before}
670   {tagpdf}
671   {
672     \UseTaggingSocket{link/before}{ }
673   }
674
675   \hook_gput_code:nnn
676   {pdfannot/link/URI/after}
677   {tagpdf}
678   {
679     \UseTaggingSocket{link/after}{ }
680   }
681
682   \hook_gput_code:nnn
683   {pdfannot/link/GoTo/before}
684   {tagpdf}
685   {

```

```

686     \UseTaggingSocket{link/before}{}
```

```

687 }
688
689 \hook_gput_code:nnn
690 {pdfannot/link/GoTo/after}
691 {tagpdf}
692 {
693     \UseTaggingSocket{link/after}{}
```

```

694 }
695
696 \hook_gput_code:nnn
697 {pdfannot/link/GoToR/before}
698 {tagpdf}
699 {
700     \UseTaggingSocket{link/before}
```

```

701 }
702
703 \hook_gput_code:nnn
704 {pdfannot/link/GoToR/after}
705 {tagpdf}
706 {
707     \UseTaggingSocket{link/after}{}
```

```

708 }
709 \hook_gput_code:nnn
710 {pdfannot/link/Launch/before}
711 {tagpdf}
712 {
713     \UseTaggingSocket{link/before}{}
```

```

714 }
715
716 \hook_gput_code:nnn
717 {pdfannot/link/Launch/after}
718 {tagpdf}
719 {
720     \UseTaggingSocket{link/after}{}
```

```

721 }
722 \hook_gput_code:nnn
723 {pdfannot/link/Named/before}
724 {tagpdf}
725 {
726     \UseTaggingSocket{link/before}{}
```

```

727 }
728
729 \hook_gput_code:nnn
730 {pdfannot/link/Named/after}
731 {tagpdf}
732 {
733     \UseTaggingSocket{link/after}{}
```

```

734 }
735 }
736 {
737     \hook_gput_code:nnn
738     {pdfannot/link/URI/before}
739     {tagpdf}
```

```

740     {
741         \UseTaggingSocket{link/before}
742         {
743             \pdfannot_dict_put:nne
744             { link/URI }
745             { StructParent }
746             { \tag_struct_parent_int: }
747         }
748     }
749
750     \hook_gput_code:nnn
751     {pdfannot/link/URI/after}
752     {tagpdf}
753     {
754         \UseTaggingSocket{link/after}
755         {
756             \tag_struct_insert_annot:ee
757             {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
758         }
759     }
760
761     \hook_gput_code:nnn
762     {pdfannot/link/GoTo/before}
763     {tagpdf}
764     {
765         \UseTaggingSocket{link/before}
766         {
767             \pdfannot_dict_put:nne
768             { link/GoTo }
769             { StructParent }
770             { \tag_struct_parent_int: }
771         }
772     }
773
774     \hook_gput_code:nnn
775     {pdfannot/link/GoTo/after}
776     {tagpdf}
777     {
778         \UseTaggingSocket{link/after}
779         {
780             \tag_struct_insert_annot:ee
781             {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
782         }
783     }
784
785     \hook_gput_code:nnn
786     {pdfannot/link/GoToR/before}
787     {tagpdf}
788     {
789         \UseTaggingSocket{link/before}
790         {
791             \pdfannot_dict_put:nne
792             { link/GoToR }
793             { StructParent }

```



```

794         { \tag_struct_parent_int: }
795     }
796 }
797
798 \hook_gput_code:nnn
799 {pdfannot/link/GoToR/after}
800 {tagpdf}
801 {
802     \UseTaggingSocket{link/after}
803     {
804         \tag_struct_insert_annot:ee
805         {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
806     }
807 }
808
809 \hook_gput_code:nnn
810 {pdfannot/link/Named/before}
811 {tagpdf}
812 {
813     \UseTaggingSocket{link/before}
814     {
815         \pdfannot_dict_put:nne
816         { link/Named }
817         { StructParent }
818         { \tag_struct_parent_int: }
819     }
820 }
821
822 \hook_gput_code:nnn
823 {pdfannot/link/Named/after}
824 {tagpdf}
825 {
826     \UseTaggingSocket{link/after}
827     {
828         \tag_struct_insert_annot:ee
829         {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
830     }
831 }
832 \hook_gput_code:nnn
833 {pdfannot/link/Launch/before}
834 {tagpdf}
835 {
836     \UseTaggingSocket{link/before}
837     {
838         \pdfannot_dict_put:nne
839         { link/Launch }
840         { StructParent }
841         { \tag_struct_parent_int: }
842     }
843 }
844
845 \hook_gput_code:nnn
846 {pdfannot/link/Launch/after}
847 {tagpdf}

```

```

848     {
849       \UseTaggingSocket{link/after}
850       {
851         \tag_struct_insert_annot:ee
852         {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
853       }
854     }
855   }

```

### 13.8 Attaching css-files for derivation

Derivation to html ([https://pdfa.org/wp-content/uploads/2019/06/Deriving\\_HTML\\_from\\_PDF.pdf](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 Catalog<sup>1</sup>.

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.

```

856 \prop_new:N \g__tag_css_prop
857 \prop_gset_from_keyval:Nn \g__tag_css_prop
858 {
859   latex-list-css.html=true,
860   latex-align-css.html=true
861 }

```

---

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

```

862
863
864 \bool_new:N \g__tag_css_bool
865 \bool_gset_true:N \g__tag_css_bool

```

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

```

866 \tl_gput_left:Nn \g__kernel_pdfmanagement_end_run_code_tl
867 {
868   \bool_lazy_and:nnT { \g__tag_css_bool }{ \tag_if_active_p: }
869   {
870     \prop_map_inline:Nn \g__tag_css_prop
871     {
872       \keys_set:nn { __tag / setup }{ catalog-supplemental-file= {#1} }
873     }
874   }
875 }
876
877 \keys_define:nn { __tag / setup }
878 {
879   attach-css .bool_gset:N = \g__tag_css_bool,
880   css-list .code:n =
881   {
882     \tl_if_empty:nTF{#1}
883     { \prop_gclear:N \g__tag_css_prop }
884     { \prop_gput:Nnn \g__tag_css_prop { #1 }{true} }
885   },
886   css-list-add .code:n = { \prop_gput:Nnn \g__tag_css_prop { #1 }{true} },
887   css-list-remove .code:n = { \prop_gremove:Nn \g__tag_css_prop { #1 } },
888 }

```

</package>

Ulrike Fischer  
Version 1.0b, released 2026-04-24

## Part IV

# The tagpdf-tree module

## Commands trees and main dictionaries

```
1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-tree-code} {2026-04-24} {1.0b}
4 {part of tagpdf - code related to writing trees and dictionaries to the pdf}
5 </header>
```

### 1 Trees, pdfmanagement and finalization code

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

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

#### 1.1 Check structure

\\_\_tag\_tree\_final\_checks:

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

(End of definition for \\_\_tag\_tree\_final\_checks:.)

## 1.2 Catalog: MarkInfo and StructTreeRoot and OpenAction

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

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

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

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

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

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

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

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

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

```

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

```

### 1.3 Writing the IDtree

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

\g\_\_tag\_tree\_id\_pad\_int

```

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

```

Now we get the needed padding

```

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

```

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

```

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

```

```

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

```

## 1.4 Writing structure elements

The following commands are needed to write out the structure.

`\__tag_tree_write_structtreeroot:` This writes out the root object.

```

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

```

```

143         \l__tag_tmpa_tl
144     }

```

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

```

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

```

(End of definition for \\_\_tag\_tree\_write\_structtreeroot:.)

\\_\_tag\_tree\_write\_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@g__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@g\_\_tag\_parenttree\_obj\_int This is a counter for the real objects. It starts at the absolute last page value. It relies on l3ref.

```

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

```

(End of definition for \c@g\_\_tag\_parenttree\_obj\_int.)

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

\g\_\_tag\_parenttree\_objr\_tl

```

162 \tl_new:N \g__tag_parenttree_objr_tl

```

(End of definition for \g\_\_tag\_parenttree\_objr\_tl.)



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

```

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

```

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

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

```

170 \tl_new:N \l__tag_parenttree_content_tl

```

(End of definition for `\l__tag_parenttree_content_tl`.)

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

```

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

```

```

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

```

(End of definition for \\_\_tag\_tree\_fill\_parenttree:.)

\\_\_tag\_tree\_lua\_fill\_parenttree: This is a special variant for luatex. lua mode must/can do it differently.

```

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

```

(End of definition for \\_\_tag\_tree\_lua\_fill\_parenttree:.)

\\_\_tag\_tree\_write\_parenttree: This combines the two parts and writes out the object. TODO should the check for lua be moved into the backend code?

```

246 \cs_new_protected:Npn \__tag_tree_write_parenttree:
247 {
248   \bool_if:NTF \g__tag_mode_lua_bool
249   {
250     \__tag_tree_lua_fill_parenttree:
251   }

```

```

252     {
253       \__tag_tree_fill_parenttree:
254     }
255   \__tag_tree_parenttree_rerun_msg:
256   \tl_put_right:No \l__tag_parenttree_content_tl { \g__tag_parenttree_objr_tl }
257   \pdf_object_write:nne { __tag/tree/parenttree }{dict}
258   {
259     /Nums\c_space_tl [\l__tag_parenttree_content_tl]
260   }
261 }
(End of definition for \__tag_tree_write_parenttree:.)

```

## 1.6 Rolemap dictionary

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

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

```

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

```

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

```

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

```

## 1.7 Classmap dictionary

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

```
\__tag_tree_write_classmap:
```

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

(End of definition for \\_\_tag\_tree\_write\_classmap:.)

## 1.8 Namespaces

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

```
__tag/tree/namespaces
```

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

`__tag_tree_write_namespaces:`

```

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

```

(End of definition for `__tag_tree_write_namespaces:.`)

## 1.9 Finishing the structure

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

`__tag_finish_structure:`

```

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

```

```

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

```

(End of definition for \\_\_tag\_finish\_structure:.)

## 1.10 StructParents entry for Page

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

```

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

```

Ulrike Fischer  
Version 1.0b, released 2026-04-24

## Part V

# The **tagpdf-mc-shared** module

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

### 1 Public Commands

---

<code>\tag_mc_begin:n</code>	<code>\tag_mc_begin:n {&lt;key-values&gt;}</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 {&lt;label&gt;}</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 {&lt;name&gt;}</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 {&lt;key-values&gt;}</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 {&lt;true code&gt;} {&lt;false code&gt;}</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 &lt;box&gt;</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 &lt;box&gt; {&lt;stream name&gt;}</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 {&lt;stream name&gt;}</code>
-----------------------------------	---

---

New: 2024-11-18

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

## 2 Public keys

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

---

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

---



---

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

---



---

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

---



---

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

---



---

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

---



---

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

---



---

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

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

### 3 Marked content code – shared

```

1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-mc-code-shared} {2026-04-24} {1.0b}
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 79.)

```

`\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 79.)

**\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 79.)

**\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 79.)

\\_\_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 81.)*

```

220 </shared>

```

Ulrike Fischer  
Version 1.0b, released 2026-04-24

## Part VI

# The tagpdf-mc-generic module

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

### 1 Marked content code – generic mode

```
1 <@@=tag>
2 <*generic>
3 \ProvidesExplPackage {tagpdf-mc-code-generic} {2026-04-24} {1.0b}
4 {part of tagpdf - code related to marking chunks - generic mode}
5 </generic>
6 <*debug>
7 \ProvidesExplPackage {tagpdf-debug-generic} {2026-04-24} {1.0b}
8 {part of tagpdf - debugging code related to marking chunks - generic mode}
9 </debug>
```

#### 1.1 Variables

```
10 <*generic>
```

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

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

`\l__tag_mc_tmpa_tl` temporary variable

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

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

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

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

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



`\tag_mc_new_stream:n`

```

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

```

(End of definition for `\tag_mc_new_stream:n`. This function is documented on page 80.)

`\l__tag_mc_firstmarks_seq`  
`\l__tag_mc_botmarks_seq`

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

```

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

```

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

## 1.2 Functions

`\__tag_mc_begin_marks:nn`  
`\__tag_mc_artifact_begin_marks:n`  
`\__tag_mc_end_marks:`

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

```

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

```

```

55     {
56       b+, %first of begin pair
57       \int_use:N\c@g__tag_MCID_abs_int, %mc-num
58       -1, %structure num
59       #1 %Type
60     }
61   }
62
63   \cs_new_protected:Npn \__tag_mc_end_marks:
64   {
65     \tex_marks:D \g__tag_mc_marks
66     {
67       e-, %first of end pair
68       \int_use:N\c@g__tag_MCID_abs_int, %mc-num
69       \g__tag_struct_stack_current_tl, %structure num
70     }
71     \tex_marks:D \g__tag_mc_marks
72     {
73       e+, %second of end pair
74       \int_use:N\c@g__tag_MCID_abs_int, %mc-num
75       \g__tag_struct_stack_current_tl, %structure num
76     }
77   }

```

(End of definition for \\_\_tag\_mc\_begin\_marks:nn, \\_\_tag\_mc\_artifact\_begin\_marks:n, and \\_\_tag\_mc\_end\_marks:.)

\\_\_tag\_mc\_disable\_marks: This disables the marks. They can't be reenabled, so it should only be used in groups.

```

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

```

(End of definition for \\_\_tag\_mc\_disable\_marks:.)

\\_\_tag\_mc\_get\_marks: This stores the current content of the marks in the sequences. It naturally should only be used in places where it makes sense.

```

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

```

(End of definition for \\_\_tag\_mc\_get\_marks:.)

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

```

93   \cs_new_protected:Npn \__tag_mc_store:nnn #1 #2 #3 %#1 mc-prev, #2 mc-num #3 structure-
      num

```

```

94 {
95   %\prop_show:N \g__tag_struct_cont_mc_prop
96   \prop_get:NnNTF \g__tag_struct_cont_mc_prop {#1} \l__tag_tmpa_tl
97   {
98     \prop_gput:Nne \g__tag_struct_cont_mc_prop {#1}{ \l__tag_tmpa_tl \__tag_struct_mcid_c
99   }
100   {
101     \prop_gput:Nne \g__tag_struct_cont_mc_prop {#1}{ \__tag_struct_mcid_dict:n {#2}}
102   }
103   \prop_gput:Nee \g__tag_mc_parenttree_prop
104   {#2}
105   {#3}
106 }
107 \cs_generate_variant:Nn \__tag_mc_store:nnn {eee}
(End of definition for \__tag_mc_store:nnn.)

```

`\__tag_mc_insert_extra_tmb:n` 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       {
134         %store
135         \__tag_mc_store:eee
136         {

```

```

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 and is currently either `main` for the main galley, `footnote` for footnote note text, or `multicol` for boxes produced for columns in that environment. Other streams may follow over time.

```

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 80.)

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

`\__tag_mc_if_in:TF`

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

`\tag_mc_if_in:TF`

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
79.)

```

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

```

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

```

\__tag_mc_bmc_artifact:n
\__tag_mc_handle_artifact:N
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  
`\tag_mc_end:` in the same group or grouping level, but the code expect that they are issued linearly.  
The tag and the state is passed to the end command through a global var and a global  
boolean.

```

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_tl }
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 79.)

## 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)
actualtext (mc-key)
label (mc-key)
artifact (mc-key)

```

```

385 \keys_define:nn { __tag / mc }
386 {
387   tag .code:n = % the name (H,P,Span) etc
388   {
389     \tl_set:Nc \l__tag_mc_key_tag_tl { #1 }
390     \tl_gset:Nc \g__tag_mc_key_tag_tl { #1 }
391   },
392   raw .code:n =
393   {
394     \tl_put_right:Nc \l__tag_mc_key_properties_tl { #1 }
395   },
396   alt .code:n = % Alt property
397   {
398     \str_set_convert:Noo
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:Nc \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:Nc \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:Noo
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:Nc \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 80.)

Ulrike Fischer  
Version 1.0b, released 2026-04-24

## Part VII

# The tagpdf-mc-luacode module

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

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

### 1 Marked content code – luamode code

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

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

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

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

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

`tag`: the type (a string)

`raw`: more properties (string)

`label`: a string.

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

`kids`: a array of tables

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

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

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

```
1 <@@=tag>
2 <*luamode>
3 \ProvidesExplPackage {tagpdf-mc-code-lua} {2026-04-24} {1.0b}
4   {tagpdf - mc code only for the luamode }
5 </luamode>
6 <*debug>
7 \ProvidesExplPackage {tagpdf-debug-lua} {2026-04-24} {1.0b}
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 80.)
```

`\_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
79.)
```

`\_tag_mc_lua_set_mc_type_attr:n`

`\_tag_mc_lua_set_mc_type_attr:o`

`\_tag_mc_lua_unset_mc_type_attr:`

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

```
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:Nx\l__tag_tmpa_tl{\lua_now:e{ltx.__tag.func.output_num_from ("#1")}} }
87   \lua_now:e
88   {
89     tex.setattribute
90     (
91       "global",
92       luatexbase.attributes.g__tag_mc_type_attr,
93       \l__tag_tmpa_tl
94     )
95   }
96   \lua_now:e
97   {
98     tex.setattribute
99     (
100       "global",
101       luatexbase.attributes.g__tag_mc_cnt_attr,
102       \_tag_get_mc_abs_cnt:
103     )

```

```

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

```

(End of definition for \\_\_tag\_mc\_lua\_set\_mc\_type\_attr:n and \\_\_tag\_mc\_lua\_unset\_mc\_type\_attr:.)

\\_\_tag\_mc\_insert\_mcid\_kids:n These commands will in the finish code replace the dummy for a mc by the real mcid  
 \\_\_tag\_mc\_insert\_mcid\_single\_kids:n kids we need a variant for the case that it is the only kid, to get the array right

```

131 \cs_new:Nn \__tag_mc_insert_mcid_kids:n
132 {
133   \lua_now:e { ltx.__tag.func.mc_insert_kids (#1,0) }
134 }
135
136 \cs_new:Nn \__tag_mc_insert_mcid_single_kids:n
137 {
138   \lua_now:e { ltx.__tag.func.mc_insert_kids (#1,1) }
139 }

```

(End of definition for \\_\_tag\_mc\_insert\_mcid\_kids:n and \\_\_tag\_mc\_insert\_mcid\_single\_kids:n.)

\\_\_tag\_mc\_handle\_stash:n This is the lua variant for the command to put an mcid absolute number in the current  
 \\_\_tag\_mc\_handle\_stash:e structure.

```

140 </luamode>
141 <{*luamode|debug>
142 <luamode>\cs_new_protected:Npn \__tag_mc_handle_stash:n #1 %1 mcidnum
143 <debug>\cs_set_protected:Npn \__tag_mc_handle_stash:n #1 %1 mcidnum
144 {
145   \__tag_check_mc_used:n { #1 }
146   \seq_gput_right:cn % Don't fill a lua table due to the command in the item,
147                     % so use the kernel command
148   { g__tag_struct_kids_\g__tag_struct_stack_current_tl _seq }

```



```

149     {
150         \__tag_mc_insert_mcid_kids:n {#1}%
151     }
152     <debug> \seq_gput_right:cn % Don't fill a lua table due to the command in the item,
153     <debug> % so use the kernel command
154     <debug> { g__tag_struct_debug_kids_\g__tag_struct_stack_current_tl _seq }
155     <debug> {
156     <debug>     MC~#1%
157     <debug> }
158     \lua_now:e
159     {
160         ltx.__tag.func.store_struct_mcab
161         (
162         \g__tag_struct_stack_current_tl,#1
163         )
164     }
165 }
166 </luamode | debug>
167 <*luamode>
168 \cs_generate_variant:Nn \__tag_mc_handle_stash:n { e }
(End of definition for \__tag_mc_handle_stash:n.)

```

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

```

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

```

set the default tag to the structure:

```

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

```

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 79.)

**`\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 79.)

**`\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 79.)

**`\__tag_get_data_mc_tag:`**

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

```

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

```

(End of definition for \\_\_tag\_get\_data\_mc\_tag:.)

## 1.2 Key definitions

```
tag (mc-key)  TODO: check conversion, check if local/global setting is right.
raw (mc-key)  238 \keys_define:nn { __tag / mc }
alt (mc-key)  239 {
lang (mc-key= 240   tag .code:n = %
actualtext (mc-key) 241   {
label (mc-key)  242     \tl_set:Nc \l__tag_mc_key_tag_tl { #1 }
artifact (mc-key) 243     \tl_gset:Nc \g__tag_mc_key_tag_tl { #1 }
                244     \lua_now:e
                245     {
                246       ltx.__tag.func.store_mc_data(\__tag_get_mc_abs_cnt:,"tag","#1")
                247     }
                248   },
                249   raw .code:n =
                250   {
                251     \tl_put_right:Nc \l__tag_mc_key_properties_tl { #1 }
                252     \lua_now:e
                253     {
                254       ltx.__tag.func.store_mc_data(\__tag_get_mc_abs_cnt:,"raw","#1")
                255     }
                256   },
                257   alt .code:n      = % Alt property
                258   {
                259     \tl_if_empty:oF{#1}
                260     {
                261       \str_set_convert:Nc
                262       \l__tag_tmpa_str
                263       { #1 }
                264       { default }
                265       { utf16/hex }
                266       \tl_put_right:Nc \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 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 80.)*

Ulrike Fischer  
Version 1.0b, released 2026-04-24

## Part VIII

# The tagpdf-struct module

## Commands to create the structure

### 1 Public Commands

---

<code>\tag_struct_begin:n</code>	<code>\tag_struct_begin:n {&lt;key-values&gt;}</code>
<code>\tag_struct_end:</code>	<code>\tag_struct_end:</code>
<code>\tag_struct_end:n</code>	<code>\tag_struct_end:n {&lt;tag&gt;}</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 {&lt;label&gt;}</code>
<code>\tag_struct_use_num:n</code>	<code>\tag_struct_use_num:n {&lt;structure number&gt;}</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 {&lt;structure number&gt;}</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 {&lt;object reference&gt;} {&lt;struct parent number&gt;}</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 {&lt;structure number&gt;} {&lt;keyword&gt;} {&lt;value&gt;}</code>
-----------------------------------	--

---

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

---

<code>\tag_struct_gput_ref:nnn</code>	<code>\tag_struct_gput_ref:nnn {&lt;structure number&gt;} {&lt;keyword&gt;} {&lt;value&gt;}</code>
---------------------------------------	--

---

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

## 2 Public keys

### 2.1 Keys for the structure commands

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

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

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

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

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

**title** (*struct key*) This keys allows to set the dictionary entry `/Title` in the structure object. The value **title-o** (*struct key*) is handled as verbatim string and hex encoded. Commands are not expanded. **title-o** will expand the value once.

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

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

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

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

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

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

**AFref** (*struct key*)

**AFinline** (*struct key*)

**AFinline-o** (*struct key*)

**texsource** (*struct key*)

**mathml** (*struct key*)

**AF** = `<object name>`

**AFref** = `<object reference>`

**AF-inline** = `<text content>`

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} {2026-04-24} {1.0b}
4 {part of tagpdf - code related to storing structure}
5 </header>
```

## 3 Variables

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

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

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

`\g__tag_struct_tag_stack_seq` We will perhaps also need the tags. While it is possible to get them from the numbered stack, lets build a tag stack too. We create one which contains also the real tags and one with only the rolemapped tags for faster testing.

```

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

```

`\g__tag_struct_stack_current_tl` The global variable will hold the current structure number. It is already defined in `tagpdf-base`. The local temporary variable will hold the parent when we fetch it from the stack.

```

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

```

In luatex we will store the structure number as attribute.

```

23 \sys_if_engine luatex:TF
24 {
25   \cs_new:Npn \__tag_struct_set_attribute:
26   {
27     \lua_now:e
28     {
29       tex.setattribute
30       (

```

```

31         "global",
32         luatexbase.attributes.g__tag_structnum_attr,
33         \g__tag_struct_stack_current_tl
34     )
35 }
36 }
37 }
38 {
39     \cs_new_eq:N \__tag_struct_set_attribute: \prg_do_nothing:
40 }

```

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

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

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

All properties have at least the keys

### Type StructTreeRoot or StructElem

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

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

```

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

```

```

67                                     % deprecated in PDF 2.0
68      T,                             %title, value in () or <>
69      Lang,                          %language
70      Alt,                           % value in () or <>
71      E,                             % abbreviation
72      ActualText,
73      AF,                            %pdf 2.0, array of dict, associated files
74      NS,                            %pdf 2.0, dict, namespace
75      PhoneticAlphabet,              %pdf 2.0
76      Phoneme                        %pdf 2.0
77  }

```

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

### 3.1 Variables used by the keys

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

```

\g__tag_struct_tag_t1
\g__tag_struct_tag_NS_t1
\l__tag_struct_roletag_t1
\g__tag_struct_roletag_NS_t1
\l__tag_struct_parenttag_t1
\l__tag_struct_parenttag_NS_t1

```

```

78 \tl_new:N \g__tag_struct_tag_t1
79 \tl_new:N \g__tag_struct_tag_NS_t1
80 \tl_new:N \l__tag_struct_roletag_t1
81 \tl_new:N \l__tag_struct_roletag_NS_t1
82 \tl_new:N \l__tag_struct_parenttag_t1
83 \tl_set:Nn \l__tag_struct_parenttag_t1 {STASHED}
84 \tl_new:N \l__tag_struct_parenttag_NS_t1
85 \tl_set:Nn \l__tag_struct_parenttag_NS_t1 {latex}

```

(End of definition for `\g__tag_struct_tag_t1` and others.)

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

```

86 \prop_new_linked:N \g__tag_struct_label_num_prop

```

(End of definition for `\g__tag_struct_label_num_prop`.)

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

```

87 \bool_new:N \l__tag_struct_elem_stash_bool

```

(End of definition for `\l__tag_struct_elem_stash_bool`.)

`\l__tag_struct_addkid_t1` This decides if a structure kid is added at the left or right of the parent. The default is **right**.

```

88 \tl_new:N \l__tag_struct_addkid_t1
89 \tl_set:Nn \l__tag_struct_addkid_t1 {right}

```

(End of definition for `\l__tag_struct_addkid_t1`.)

### 3.2 Variables used by tagging code of basic elements

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

```

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

```

(End of definition for `\g__tag_struct_dest_num_prop`.)

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

```

93 \prop_new_linked:N \g__tag_struct_ref_by_dest_prop
94 \</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:

```

95 <*package| debug>
96 <package>\cs_new_protected:Npn \__tag_struct_prop_gput:nnn #1 #2 #3
97 <debug>\cs_set_protected:Npn \__tag_struct_prop_gput:nnn #1 #2 #3
98 {
99   \__tag_prop_gput:cnn
100   { g__tag_struct_#1_prop }{#2}{#3}
101 <debug>\prop_gput:cnn { g__tag_struct_debug_#1_prop } {#2} {#3}
102 }
103 \cs_generate_variant:Nn \__tag_struct_prop_gput:nnn {onn,nne,nee,nnn}
104 \</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.

```

105 <*package>
106 \tl_gset:Nn \g__tag_struct_stack_current_tl {1}

```

`\__tag_pdf_name_e:n`

```

107 \cs_new:Npn \__tag_pdf_name_e:n #1{\pdf_name_from_unicode_e:n{#1}}
108 \</package>

```

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

`g__tag_struct_1_prop`  
`g__tag_struct_kids_1_seq`

```

109 <*package>
110 \__tag_prop_new:c { g__tag_struct_1_prop }
111 \__tag_seq_new:c { g__tag_struct_kids_1_seq }
112
113 \__tag_struct_prop_gput:nne
114 { 1 }
115 { Type }
116 { \pdf_name_from_unicode_e:n {StructTreeRoot} }
117
118 \__tag_struct_prop_gput:nne

```

```

119 { 1 }
120 { S }
121 { \pdf_name_from_unicode_e:n {StructTreeRoot} }
122
123 \__tag_struct_prop_gput:nne
124 { 1 }
125 { tag }
126 { {StructTreeRoot}{pdf} }
127
128 \__tag_struct_prop_gput:nne
129 { 1 }
130 { rolemap }
131 { {StructTreeRoot}{pdf} }
132
133 \__tag_struct_prop_gput:nne
134 { 1 }
135 { parentrole }
136 { {StructTreeRoot}{pdf} }
137

```

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

```

138 \pdf_version_compare:NnF < {2.0}
139 {
140   \__tag_struct_prop_gput:nne
141   { 1 }
142   { Namespaces }
143   { \pdf_object_ref:n { __tag/tree/namespaces } }
144 }
145 </package>

```

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

```

146 <debug>\prop_new:c { g__tag_struct_debug_1_prop }
147 <debug>\seq_new:c { g__tag_struct_debug_kids_1_seq }
148 <debug>\prop_gset_eq:cc { g__tag_struct_debug_1_prop }{ g__tag_struct_1_prop }
149 <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
150 <*package>
151 \cs_new:Npn \__tag_struct_get_id:n #1 %#1=struct num
152 {
153   (
154     ID.
155     \prg_replicate:nn
156     { \int_abs:n{\g__tag_tree_id_pad_int - \tl_count:e { \int_to_arabic:n { #1 } }} }
157     { 0 }
158     \int_to_arabic:n { #1 }
159   )
160 }

```

(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.

```

161 \pdf_version_compare:NnTF < {2.0}
162 {
163   \cs_new_protected:Npn __tag_struct_set_tag_info:nnn #1 #2 #3
164     {%#1 structure number, #2 tag, #3 NS
165     {
166       __tag_struct_prop_gput:nne
167         { #1 }
168         { S }
169         { \pdf_name_from_unicode_e:n {#2} } %
170       __tag_struct_prop_gput:nnn
171         { #1 }
172         { tag }
173         { {#2} {} }
174     }
175   }
176   {
177     \cs_new_protected:Npn __tag_struct_set_tag_info:nnn #1 #2 #3
178     {
179       __tag_struct_prop_gput:nne
180         { #1 }
181         { S }
182         { \pdf_name_from_unicode_e:n {#2} } %
183       \prop_get:NnNT \g__tag_role_NS_prop {#3} \l__tag_get_tmpc_tl
184       {
185         __tag_struct_prop_gput:nne
186           { #1 }
187           { NS }
188           { \l__tag_get_tmpc_tl } %
189       }
190       __tag_struct_prop_gput:nnn
191         { #1 }
192         { tag }
193         { {#2} {#3} }
194     }
195   }
196   \cs_generate_variant:Nn __tag_struct_set_tag_info:nnn {eoo}
197 }
198 (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.

```

197 \cs_new_protected:Npn __tag_struct_get_role:nnNN #1 #2 #3 #4
198   {%#1 :struct num,
199   %#2 :rolemap or parentrole
200   %#3 :tlvar for tag (rolemapped)
201   %#4 :tlvar for NS (rolemapped, so standard or empty or UNKNOWN)
202   {

```

```

203 \prop_get:cnNTF
204 { g__tag_struct_#1_prop }
205 { #2 }
206 \l__tag_get_tmpc_t1
207 {
208     \tl_set:Ne #3{\exp_last_unbraced:No\use_i:nn { \l__tag_get_tmpc_t1 }}
209     \tl_set:Ne #4{\exp_last_unbraced:No\use_ii:nn { \l__tag_get_tmpc_t1 }}
210 }
211 {
212     \tl_clear:N#3
213     \tl_clear:N#4
214 }
215 }
216 \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!

```

217 \cs_new:Npn \__tag_struct_mcid_dict:n #1 %#1 MCID absnum
218 {
219     <<
220     /Type \c_space_t1 /MCR \c_space_t1
221     /Pg
222     \c_space_t1
223     \pdf_pageobject_ref:n { \property_ref:enn{mcid-#1}{tagabspage}{1} }
224     /MCID \c_space_t1 \property_ref:enn{mcid-#1}{tagmcid}{1}
225     >>
226 }
227 </package>
228 <*package | debug>
229 <package>\cs_new_protected:Npn \__tag_struct_kid_mc_gput_right:nn #1 #2
230 <debug>\cs_set_protected:Npn \__tag_struct_kid_mc_gput_right:nn #1 #2
231 %#1 structure num, #2 MCID absnum%
232 {
233     \__tag_seq_gput_right:ce
234     { g__tag_struct_kids_#1_seq }
235     {
236         \__tag_struct_mcid_dict:n {#2}
237     }
238 <debug> \seq_gput_right:cn
239 <debug> { g__tag_struct_debug_kids_#1_seq }

```



```

240 <debug>      {
241 <debug>      MC~#2
242 <debug>      }
243      \__tag_seq_gput_right:cn
244      { g__tag_struct_kids_#1_seq }
245      {
246      \prop_item:Nn \g__tag_struct_cont_mc_prop {#2}
247      }
248  }
249 <package>\cs_generate_variant:Nn \__tag_struct_kid_mc_gput_right:nn {ne}
(End of definition for \__tag_struct_kid_mc_gput_right:nn.)

```

\\_\_tag\_struct\_kid\_struct\_gput\_right:nn  
\\_\_tag\_struct\_kid\_struct\_gput\_right:ee

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

```

250 <package>\cs_new_protected:Npn\__tag_struct_kid_struct_gput_right:nn #1 #2
251 <debug>\cs_set_protected:Npn\__tag_struct_kid_struct_gput_right:nn #1 #2
252 %%#1 num of parent struct, #2 kid struct
253 {
254      \__tag_seq_gput_right:ce
255      { g__tag_struct_kids_#1_seq }
256      {
257      \pdf_object_ref_indexed:nn { __tag/struct }{ #2 }
258      }
259 <debug>      \seq_gput_right:cn
260 <debug>      { g__tag_struct_debug_kids_#1_seq }
261 <debug>      {
262 <debug>      Struct~#2
263 <debug>      }
264  }
265 <package>\cs_generate_variant:Nn \__tag_struct_kid_struct_gput_right:nn {ee}
(End of definition for \__tag_struct_kid_struct_gput_right:nn.)

```

\\_\_tag\_struct\_kid\_struct\_gput\_left:nn  
\\_\_tag\_struct\_kid\_struct\_gput\_left:ee

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

```

266 <package>\cs_new_protected:Npn\__tag_struct_kid_struct_gput_left:nn #1 #2
267 <debug>\cs_set_protected:Npn\__tag_struct_kid_struct_gput_left:nn #1 #2
268 %%#1 num of parent struct, #2 kid struct
269 {
270      \__tag_seq_gput_left:ce
271      { g__tag_struct_kids_#1_seq }
272      {
273      \pdf_object_ref_indexed:nn { __tag/struct }{ #2 }
274      }
275 <debug>      \seq_gput_left:cn
276 <debug>      { g__tag_struct_debug_kids_#1_seq }
277 <debug>      {
278 <debug>      Struct~#2
279 <debug>      }
280  }
281 <package>\cs_generate_variant:Nn \__tag_struct_kid_struct_gput_left:nn {ee}
(End of definition for \__tag_struct_kid_struct_gput_left:nn.)

```

\\_tag\_struct\_kid\_OBJR\_gput\_right:nnn  
\\_tag\_struct\_kid\_OBJR\_gput\_right:eee

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

```

282 <package>\cs_new_protected:Npn\_tag_struct_kid_OBJR_gput_right:nnn #1 #2 #3
283 <package>
284 <package>
285 <debug>\cs_set_protected:Npn\_tag_struct_kid_OBJR_gput_right:nnn #1 #2 #3
286 %%#1 num of parent struct,#2 obj reference,#3 page object reference
287 {
288   \pdf_object_unnamed_write:nn
289     { dict }
290     {
291       /Type/OBJR/Obj~#2/Pg~#3
292     }
293   \__tag_seq_gput_right:ce
294     { g__tag_struct_kids_#1_seq }
295     {
296       \pdf_object_ref_last:
297     }
298 <debug>   \seq_gput_right:ce
299 <debug>   { g__tag_struct_debug_kids_#1_seq }
300 <debug>   {
301 <debug>     OBJR~reference
302 <debug>   }
303 }
304 </package|debug>
305 <*package>
306 \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.

```

307 \cs_new_protected:Npn\_tag_struct_exchange_kid_command:N #1 %%#1 = seq var
308 {
309   \seq_gpop_left:NN #1 \l__tag_tmpa_tl
310   \tl_replace_once:Nnn \l__tag_tmpa_tl
311     { \__tag_mc_insert_mcid_kids:n }
312     { \__tag_mc_insert_mcid_single_kids:n }
313   \seq_gput_left:No #1 { \l__tag_tmpa_tl }
314 }
315
316 \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.

```

317 \cs_new_protected:Npn\_tag_struct_fill_kid_key:n #1 %%#1 is the struct num
318 {
319   \bool_if:NF \g__tag_mode_lua_bool
320   {

```

```

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

```

check if we get null

```

349 \tl_set:N\l__tag_tmpa_tl
350 { \use:ef\seq_item:cn {g__tag_struct_kids_#1_seq} {1}}}
351 \tl_if_eq:NNF\l__tag_tmpa_tl \c__tag_struct_null_tl
352 {
353   \__tag_struct_prop_gput:nne
354   {#1}
355   {K}
356   {
357     \seq_item:cn
358     {
359       g__tag_struct_kids_#1_seq
360     }
361     {1}
362   }
363 }
364 }
365 {
366   \__tag_struct_prop_gput:nne
367   {#1}
368   {K}
369   {
370     \seq_item:cn
371     {
372       g__tag_struct_kids_#1_seq

```

```

373         }
374         {1}
375     }
376 }
377 } %
378 }
379 { %many kids, use an array
380   \__tag_struct_prop_gput:nne
381   {#1}
382   {K}
383   {
384     [
385       \seq_use:cn
386       {
387         g__tag_struct_kids_#1_seq
388       }
389       {
390         \c_space_tl
391       }
392     ]
393   }
394 }
395 }
396
(End of definition for \__tag_struct_fill_kid_key:n.)

```

## 4.5 Output of the object

\\_\_tag\_struct\_get\_dict\_content:nnN

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.

```

397 \cs_new_protected:Npn \__tag_struct_get_dict_content:nN #1 #2 {%#1: structure num
398   {
399     \tl_clear:N #2
400     \prop_map_inline:cn { g__tag_struct_#1_prop }
401     {

```

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.

```

402     \cs_if_exist_use:cTF {\__tag_struct_format_##1:nnN}
403     {
404       {##1}{##2}#2
405     }
406     {
407       \tl_put_right:Ne #2 { \c_space_tl/##1-##2 }
408     }
409   }
410 }
(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

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.

```

411 \cs_new:Nn\__tag_struct_format_rolemap:nnN{}
412 \cs_new:Nn\__tag_struct_format_parentrole:nnN{}
413 \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.

```

414 \cs_new_protected:Nn\__tag_struct_format_parentnum:nnN
415 {
416   \tl_put_right:Ne #3 { ~/P~\pdf_object_ref_indexed:nn { __tag/struct } { #2 } }
417 }

```

(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.

```

418 \cs_new_protected:Nn\__tag_struct_format_Ref:nnN
419 {
420   \tl_put_right:Nn #3 { ~/#1~[ ] %]
421   \clist_map_inline:nn{ #2 }
422   {
423     ##1 #3
424   }
425   \tl_put_right:Nn #3
426   { %[
427     \c_space_tl]
428   }
429 }

```

(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.

```

430 \cs_new_protected:Npn \__tag_struct_write_obj:n #1 % #1 is the struct num
431 {
432   \prop_if_exist:cTF { g__tag_struct_#1_prop }
433   {

```

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.

```

434     \prop_get:cnNF { g__tag_struct_#1_prop } {parentnum}\l__tag_tmpb_tl
435     {
436       %       \prop_gput:cne { g__tag_struct_#1_prop } {P}
437       %       {\pdf_object_ref_indexed:nn { __tag/struct }{1}}
438       \prop_gput:cne { g__tag_struct_#1_prop } {parentnum}{2}
439       \prop_gput:cne { g__tag_struct_#1_prop } {S}{/Artifact}
440       \seq_if_empty:cF {g__tag_struct_kids_#1_seq}
441       {
442         \msg_warning:nnee
443         {tag}
444         {struct-orphan}
445         { #1 }
446         {\seq_count:c{g__tag_struct_kids_#1_seq}}
447       }
448     }
449     \__tag_struct_fill_kid_key:n { #1 }
450     \__tag_struct_get_dict_content:nN { #1 } \l__tag_tmpa_tl
451     \pdf_object_write_indexed:nnne
452     { __tag/struct }{ #1 }

```

```

453         {dict}
454         {
455             \l__tag_tmpa_tl\c_space_tl
456             /ID~\__tag_struct_get_id:n{#1}
457         }
458
459     }
460     {
461         \msg_error:nnn { tag } { struct-no-objnum } { #1}
462     }
463 }

```

(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:

```

```

464 \cs_new_protected:Npn \__tag_struct_insert_annot:nn #1 #2
465   %#1 object reference to the annotation/xform
466   %#2 structparent number
467   {
468     \bool_if:NT \g__tag_active_struct_bool
469     {
470       %get the number of the parent structure:
471       \seq_get:NNF
472       \g__tag_struct_stack_seq
473       \l__tag_struct_stack_parent_tmpa_tl
474       {
475         \msg_error:nn { tag } { struct-faulty-nesting }
476       }
477       %put the obj number of the annot in the kid entry, this also creates
478       %the OBJR object
479       \__tag_property_record:nn {@tag@objr@page@#2 }{ tagabspage }
480       \__tag_struct_kid_OBJR_gput_right:eee
481       {

```

```

482         \l__tag_struct_stack_parent_tmpa_tl
483     }
484     {
485         #1 %
486     }
487     {
488         \pdf_pageobject_ref:n
489         { \property_ref:nnn {@tag@objr@page@#2 }{ tagabspage }{1} }
490     }
491     % add the parent obj number to the parent tree:
492     % the command always expands its arguments!
493     \__tag_parenttree_add_objr:nn
494     {
495         #2
496     }
497     {
498         \pdf_object_ref_indexed:nn
499         { __tag/struct }{ \l__tag_struct_stack_parent_tmpa_tl }
500     }
501     % increase the int:
502     \int_gincr:N \c@g__tag_parenttree_obj_int
503 }
504 }

```

(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.

```

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

```

```

527         { __tag/struct }{ #1 }
528     }
529 }
(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`.

```

530 \cs_new:Npn \__tag_get_data_struct_tag:
531 {
532     \exp_args:Ne
533     \tl_tail:n
534     {
535         \prop_item:cn {g__tag_struct_\g__tag_struct_stack_current_tl _prop}{S}
536     }
537 }
(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`.

```

538 \cs_new:Npn \__tag_get_data_struct_id:
539 {
540     \__tag_struct_get_id:n {\g__tag_struct_stack_current_tl}
541 }
542 \</package>
(End of definition for \__tag_get_data_struct_id:.)

```

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

```

543 \< *base>
544 \cs_new:Npn \__tag_get_data_struct_num:
545 {
546     \g__tag_struct_stack_current_tl
547 }
548 \</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.

```

549 \< *base>
550 \cs_new:Npn \__tag_get_data_struct_counter:
551 {
552     \int_use:N \c@g__tag_struct_abs_int
553 }
554 \</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

```

555 (*package)
556 \cs_new_protected:Npn \_tag_struct_check_parent_child_aux:nnnnN #1#2#3#4#5
557 {
558 % #1 structure number of parent
559 % #2 key to use to retrieve role of parent (either rolemap or parentrole field)
560 % #3 structure number of parent
561 % #4 key to use to retrieve role of child (either rolemap or parentrole field)
562 % #5 t1 for return value

get parent rolemap

563 \_tag_struct_get_role:nnNN
564 {#1}
565 {#2}
566 \l__tag_get_parent_tmpa_tl
567 \l__tag_get_parent_tmpb_tl

get child rolemap

568 \_tag_struct_get_role:nnNN
569 {#3}
570 {#4}
571 \l__tag_get_child_tmpa_tl
572 \l__tag_get_child_tmpb_tl

check

573 \_tag_role_check_parent_child:ooooN
574 { \l__tag_get_parent_tmpa_tl } % rolemapped from above
575 { \l__tag_get_parent_tmpb_tl } % rolemapped from above
576 { \l__tag_get_child_tmpa_tl } %
577 { \l__tag_get_child_tmpb_tl } %
578 #5
579 }

(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.

```

580 \cs_new_protected:Npn \_tag_struct_check_parent_child:nn #1 #2
581 % #1 structure number of parent
582 % #2 structure number of child. %
583 % This assumes that the fields rolemap/parentrole has already been filled.
584 {

```

This records if logging is on

```

585 \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
586 {
587 \prop_get:cnN{g__tag_struct_#1_prop}{tag}\l__tag_get_parent_tmpa_tl
588 \prop_get:cnN{g__tag_struct_#2_prop}{tag}\l__tag_get_parent_tmpb_tl
589 \msg_note:nnee
590 { tag }
591 { role-parent-child-check }
592 {
593 \quark_if_no_value:NTF \l__tag_get_parent_tmpa_tl
594 {??}
595 {

```

```

596         \exp_last_unbraced:No\use_ii:nn
597         { \l__tag_get_parent_tmpa_tl }
598         :
599         \exp_last_unbraced:No\use_i:nn
600         { \l__tag_get_parent_tmpa_tl }
601     }
602 }
603 {
604     \quark_if_no_value:NTF \l__tag_get_parent_tmpb_tl
605     {??}
606     {
607         \exp_last_unbraced:No\use_ii:nn
608         { \l__tag_get_parent_tmpb_tl }
609         :
610         \exp_last_unbraced:No\use_i:nn
611         { \l__tag_get_parent_tmpb_tl }
612     }
613 }
614 }
615 \__tag_struct_check_parent_child_aux:nnnnN
616 {#1}
617 {rolemap}
618 {#2}
619 {rolemap}
620 \l__tag_parent_child_check_tl

```

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

```

621 \int_compare:nNtT {\l__tag_parent_child_check_tl} = { \c__tag_role_rule_checkparent_tl }
622 {
623     \__tag_struct_check_parent_child_aux:nnnnN
624     {#1}
625     {parentrole}
626     {#2}
627     {rolemap}
628     \l__tag_parent_child_check_tl
629 }
630 \__tag_check_struct_forbidden_parent_child:onn
631 {\l__tag_parent_child_check_tl}
632 {#1}
633 {#2}
634 }
635 \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.

```

636 \cs_new_protected:Npn \__tag_struct_use_check_parent_child:nn #1 #2
637 % #1 structure number of parent
638 % #2 structure number of child. %
639 {
640   \__tag_struct_get_role:enNN
641   {#2}
642   {rolemap}
643   \l__tag_get_child_tmpa_tl
644   \l__tag_get_child_tmpb_tl
645   \str_case:onTF { \l__tag_get_child_tmpa_tl }
646   {
647     {Part} {}
648     {Div} {}
649     {NonStruct} {}
650   }
651   { %child=Part etc
652     \__tag_struct_get_role:enNN
653     {#2}
654     {parentrole}
655     \l__tag_get_child_tmpa_tl
656     \l__tag_get_child_tmpb_tl
657     \str_if_eq:noTF
658     {STASHED}{\l__tag_get_child_tmpa_tl}
659     {
660       % warn about unknown relationship
661     }
662     {
663       % test if
664       \__tag_struct_get_role:enNN
665       {#1}
666       {parentrole}
667       \l__tag_get_parent_tmpa_tl
668       \l__tag_get_parent_tmpb_tl
669       \tl_if_eq:NNTF\l__tag_get_parent_tmpa_tl \l__tag_get_child_tmpa_tl
670       {
671         \__tag_struct_check_parent_child:nn {#1}{#2}
672       }
673       {
674         %warn that parent-tag was misused.
675       }
676     }
677   }
678   {
679     %child not Part etc, normal parent child test.
680     \__tag_struct_check_parent_child:nn {#1}{#2}
681   }
682 }
683 \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.

```

684 \socket_new:nn { tag/struct/tag }{1}
685 \socket_new_plug:nnn { tag/struct/tag }{ latex-tags }
686 {
687   \prop_get:NeNTF \g__tag_role_tags_NS_prop {#1} \l__tag_tmp_unused_tl
688   {
689     \seq_set_split:Nne \l__tag_tmpa_seq { / }
690     {#1/\l__tag_tmp_unused_tl}
691   }
692   {
693     \seq_set_split:Nne \l__tag_tmpa_seq { / }
694     {#1/}
695   }
696   \tl_gset:Ne \g__tag_struct_tag_tl { \seq_item:Nn\l__tag_tmpa_seq {1} }
697   \tl_gset:Ne \g__tag_struct_tag_NS_tl{ \seq_item:Nn\l__tag_tmpa_seq {2} }
698   \__tag_check_structure_tag:N \g__tag_struct_tag_tl
699 }
700
701 \socket_new_plug:nnn { tag/struct/tag }{ pdf-tags }
702 {
703   \prop_get:NeNTF \g__tag_role_tags_NS_prop {#1} \l__tag_tmp_unused_tl
704   {
705     \seq_set_split:Nne \l__tag_tmpa_seq { / }
706     {#1/\l__tag_tmp_unused_tl}
707   }
708   {
709     \seq_set_split:Nne \l__tag_tmpa_seq { / }
710     {#1/}
711   }
712   \tl_gset:Ne \g__tag_struct_tag_tl { \seq_item:Nn\l__tag_tmpa_seq {1} }
713   \tl_gset:Ne \g__tag_struct_tag_NS_tl{ \seq_item:Nn\l__tag_tmpa_seq {2} }
714   \__tag_role_get:ooNN
715   { \g__tag_struct_tag_tl }
716   { \g__tag_struct_tag_NS_tl}
717   \l__tag_tmpa_tl
718   \l__tag_tmpb_tl
719   \tl_gset:Ne \g__tag_struct_tag_tl {\l__tag_tmpa_tl}
720   \tl_gset:Ne \g__tag_struct_tag_NS_tl{\l__tag_tmpb_tl}
721   \__tag_check_structure_tag:N \g__tag_struct_tag_tl
722 }
723 \socket_assign_plug:nn { tag/struct/tag } {latex-tags}

```

```

label (struct key)
stash (struct key) 724 \keys_define:nn { __tag / struct }
parent (struct key) 725 {
firstkid (struct key) 726   label .code:n      =
tag (struct key) 727   {
title (struct key) 728     \prop_gput:Nee\g__tag_struct_label_num_prop
title-o (struct key)
alt (struct key)
actualtext (struct key)
lang (struct key)
ref (struct key)
E (struct key)
phoneme (struct key)

```

```

729         {#1}{\int_use:N \c@g__tag_struct_abs_int}
730     \__tag_property_record:eo
731     {tagpdfstruct-#1}
732     { \c__tag_property_struct_clist }
733 },
734 stash .bool_set:N      = \l__tag_struct_elem_stash_bool,
735 parent .code:n         =
736 {
737     \bool_lazy_and:nnTF
738     {
739         \prop_if_exist_p:c { g__tag_struct_\int_eval:n {#1}_prop }
740     }
741     {
742         \int_compare_p:nNn {#1}<{\c@g__tag_struct_abs_int}
743     }
744     { \tl_set:Ne \l__tag_struct_stack_parent_tmpa_tl { \int_eval:n {#1} } }
745     {
746         \msg_warning:nnee { tag } { struct-unknown }
747         { \int_eval:n {#1} }
748         { parent-key-ignored }
749     }
750 },
751 parent .default:n      = {-1},
752 parent-tag .code:n     =
753 {
754     \prop_get:NeNTF \g__tag_role_tags_NS_prop {#1} \l__tag_tmp_unused_tl
755     {
756         \seq_set_split:Nne \l__tag_tmpa_seq { / }
757         {#1/\l__tag_tmp_unused_tl}
758     }
759     {
760         \seq_set_split:Nne \l__tag_tmpa_seq { / }
761         {#1/}
762     }
763     \tl_set:Ne \l__tag_struct_parenttag_tl { \seq_item:Nn \l__tag_tmpa_seq {1} }
764     \tl_set:Ne \l__tag_struct_parenttag_NS_tl { \seq_item:Nn \l__tag_tmpa_seq {2} }
765     \__tag_role_get:ooNN
766     { \l__tag_struct_parenttag_tl }
767     { \l__tag_struct_parenttag_NS_tl }
768     \l__tag_tmpa_tl
769     \l__tag_tmpb_tl
770     \tl_set:No \l__tag_struct_parenttag_tl { \l__tag_tmpa_tl }
771     \tl_set:No \l__tag_struct_parenttag_NS_tl { \l__tag_tmpb_tl }
772     \__tag_check_structure_tag:N \l__tag_struct_parenttag_tl
773 },
774 firstkid .code:n = { \tl_set:Nn \l__tag_struct_addkid_tl {left} },
775 tag .code:n      = % S property
776 {
777     \socket_use:nn { tag/struct/tag }{#1}
778 },
779 title .code:n    = % T property
780 {
781     \str_set_convert:Nnnn
782     \l__tag_tmpa_str

```

```

783         { #1 }
784         { default }
785         { utf16/hex }
786     \_tag_struct_prop_gput:nne
787     { \int_use:N \c@g__tag_struct_abs_int }
788     { T }
789     { <\l__tag_tmpa_str> }
790 },
791 title-o .code:n          = % T property
792 {
793     \str_set_convert:Nonn
794     \l__tag_tmpa_str
795     { #1 }
796     { default }
797     { utf16/hex }
798     \_tag_struct_prop_gput:nne
799     { \int_use:N \c@g__tag_struct_abs_int }
800     { T }
801     { <\l__tag_tmpa_str> }
802 },
803 alt .code:n             = % Alt property
804 {
805     \tl_if_empty:oF{#1}
806     {
807         \str_set_convert:Noon
808         \l__tag_tmpa_str
809         { #1 }
810         { default }
811         { utf16/hex }
812         \_tag_struct_prop_gput:nne
813         { \int_use:N \c@g__tag_struct_abs_int }
814         { Alt }
815         { <\l__tag_tmpa_str> }
816     }
817 },
818 alttext .meta:n = {alt=#1},
819 actualtext .code:n = % ActualText property
820 {
821     \tl_if_empty:oF{#1}
822     {
823         \str_set_convert:Noon
824         \l__tag_tmpa_str
825         { #1 }
826         { default }
827         { utf16/hex }
828         \_tag_struct_prop_gput:nne
829         { \int_use:N \c@g__tag_struct_abs_int }
830         { ActualText }
831         { <\l__tag_tmpa_str>}
832     }
833 },
834 phoneme .code:n = % Phoneme property
835 {
836     \tl_if_empty:oF{#1}

```

```

837         {
838             \str_set_convert:Noon
839             \l__tag_tmpa_str
840             { #1 }
841             { default }
842             { utf16/hex }
843             \__tag_struct_prop_gput:nne
844             { \int_use:N \c@g__tag_struct_abs_int }
845             { Phoneme }
846             { <\l__tag_tmpa_str>}
847         }
848     },
849     lang .code:n          = % Lang property
850     {
851         \__tag_struct_prop_gput:nne
852         { \int_use:N \c@g__tag_struct_abs_int }
853         { Lang }
854         { (#1) }
855     },
856 }

```

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

```

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

```

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

```

857 \cs_new_protected:Npn \__tag_struct_Ref_obj:nN #1 #2 %#1 a object reference
858 {
859     \tl_put_right:Nc#2
860     {
861         \c_space_tl#1
862     }
863 }
864
865 \cs_new_protected:Npn \__tag_struct_Ref_label:nN #1 #2 %#1 a label
866 {
867     \prop_get:NnNTF \g__tag_struct_label_num_prop {#1} \l__tag_tmpb_tl
868     {
869         \tl_put_right:Nc#2
870         {
871             \c_space_tl\tag_struct_object_ref:e{ \l__tag_tmpb_tl }
872         }
873     }
874     {
875         \msg_warning:nnn {tag}{struct-Ref-unknown}{Label~'#1'}
876     }
877 }
878 \cs_new_protected:Npn \__tag_struct_Ref_dest:nN #1 #2 %#1 a dest name
879 {
880     \prop_get:NnNTF \g__tag_struct_dest_num_prop {#1} \l__tag_tmpb_tl
881     {

```

```

882     \tl_put_right:Ne#2
883     {
884         \c_space_tl\tag_struct_object_ref:e{ \l__tag_tmpb_tl }
885     }
886 }
887 {
888     \msg_warning:nnn {tag}{struct-Ref-unknown}{Destination~'#1'}
889 }
890 }
891 \cs_new_protected:Npn \__tag_struct_Ref_dest_parent:nN #1 #2 %#1 a dest name
892 {
893     \prop_get:NnNTF \g__tag_struct_dest_num_prop {#1} \l__tag_tmpb_tl
894     { %do not overwrite \l__tag_tmpa_tl!
895         \prop_get:cnN
896         { g__tag_struct_\l__tag_tmpb_tl _prop }{parentnum}\l__tag_tmpb_tl
897         \tl_put_right:Ne#2
898         {
899             \c_space_tl\tag_struct_object_ref:e{ \l__tag_tmpb_tl }
900         }
901     }
902     {
903         \msg_warning:nnn {tag}{struct-Ref-unknown}{Destination~'#1'}
904     }
905 }
906 \cs_new_protected:Npn \__tag_struct_Ref_num:nN #1 #2 %#1 a structure number
907 {
908     \tl_put_right:Ne#2
909     {
910         \c_space_tl\tag_struct_object_ref:e{ #1 }
911     }
912 }
913
(End of definition for \__tag_struct_Ref_obj:nN and others.)

```

ref (struct key)

E (struct key)

```

914 \keys_define:nn { __tag / struct }
915 {
916     ref .code:n          = % ref property
917     {
918         \clist_map_inline:on {#1}
919         {
920             \tag_struct_gput:nne
921             {\int_use:N \c@g__tag_struct_abs_int}{ref_label}{ ##1 }
922         }
923     },
924     E .code:n           = % E property
925     {
926         \str_set_convert:Nnon
927         \l__tag_tmpa_str
928         { #1 }
929         { default }
930         { utf16/hex }
931         \__tag_struct_prop_gput:nne
932         { \int_use:N \c@g__tag_struct_abs_int }

```



```

933         { E }
934         { <\l__tag_tmpa_str> }
935     },
936 }

```

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

```

937 \int_new:N\g__tag_struct_AFobj_int
938 \cs_generate_variant:Nn \pdf_file_embed_stream:nnN {neN}
939 \cs_new_protected:Npn \__tag_struct_add_inline_AF:nn #1 #2
940 % #1 content, #2 extension
941 {
942     \tl_if_empty:nF{#1}
943     {
944         \group_begin:
945         \int_gincr:N \g__tag_struct_AFobj_int
946         \pdf_file_embed_stream:neN
947         {#1}
948         {tag-AFfile\int_use:N\g__tag_struct_AFobj_int.#2}
949         \l__tag_tmpa_tl
950         \__tag_struct_add_AF:ee
951         { \int_use:N \c@g__tag_struct_abs_int }
952         { \l__tag_tmpa_tl }
953         \__tag_struct_prop_gput:nne
954         { \int_use:N \c@g__tag_struct_abs_int }
955         { AF }
956         {
957             [
958                 \tl_use:c
959                 { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
960             ]
961         }
962         \group_end:
963     }
964 }
965
966 \cs_generate_variant:Nn \__tag_struct_add_inline_AF:nn {on}
967 \cs_new_protected:Npn \__tag_struct_add_AF:nn #1 #2
968 % #1 struct num #2 object reference
969 {
970     \tl_if_exist:cTF
971     {
972         g__tag_struct_#1_AF_tl
973     }
974     {
975         \tl_gput_right:ce
976         { g__tag_struct_#1_AF_tl }
977         { \c_space_tl #2 }

```

```

978     }
979     {
980         \tl_new:c
981         { g__tag_struct_#1_AF_tl }
982         \tl_gset:ce
983         { g__tag_struct_#1_AF_tl }
984         { #2 }
985     }
986 }
987 \cs_generate_variant:Nn \__tag_struct_add_AF:nn {en,ee}
988 \keys_define:nn { __tag / struct }
989 {
990     AF .code:n          = % AF property
991     {
992         \pdf_object_if_exist:eTF {#1}
993         {
994             \__tag_struct_add_AF:ee
995             { \int_use:N \c@g__tag_struct_abs_int }{\pdf_object_ref:e {#1}}
996             \__tag_struct_prop_gput:nne
997             { \int_use:N \c@g__tag_struct_abs_int }
998             { AF }
999             {
1000                 [
1001                     \tl_use:c
1002                     { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
1003                 ]
1004             }
1005         }
1006         {
1007             % message?
1008         }
1009     },
1010     AFref .code:n        = % AF property
1011     {
1012         \tl_if_empty:eF {#1}
1013         {
1014             \__tag_struct_add_AF:ee { \int_use:N \c@g__tag_struct_abs_int }{#1}
1015             \__tag_struct_prop_gput:nne
1016             { \int_use:N \c@g__tag_struct_abs_int }
1017             { AF }
1018             {
1019                 [
1020                     \tl_use:c
1021                     { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
1022                 ]
1023             }
1024         }
1025     },
1026     ,AFinline .code:n =
1027     {
1028         \__tag_struct_add_inline_AF:nn {#1}{txt}
1029     }
1030     ,AFinline-o .code:n =
1031     {

```

```

1032     \__tag_struct_add_inline_AF:on {#1}{txt}
1033   }
1034   ,texsource .code:n =
1035   {
1036     \group_begin:
1037     \pdfdict_put:nnn { l_pdffile/Filespec } {Desc}{(TeX~source)}
1038     \pdfdict_put:nnn { l_pdffile/Filespec }{AFRelationship} { /Source }
1039     \__tag_struct_add_inline_AF:on {#1}{tex}
1040     \group_end:
1041   }
1042   ,mathml .code:n =
1043   {
1044     \group_begin:
1045     \pdfdict_put:nnn { l_pdffile/Filespec } {Desc}{(mathml~representation)}
1046     \pdfdict_put:nnn { l_pdffile/Filespec }{AFRelationship} { /Supplement }
1047     \pdfdict_put:nne { l_pdffile }{Subtype}
1048     { \pdf_name_from_unicode_e:n{application/mathml+xml} }
1049     \__tag_struct_add_inline_AF:on {#1}{xml}
1050     \group_end:
1051   }
1052 }

```

**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!

```

1053 \keys_define:nn { __tag / setup }
1054 {
1055   root-AF .code:n =
1056   {
1057     \pdf_object_if_exist:nTF {#1}
1058     {
1059       \__tag_struct_add_AF:ee { 1 }{\pdf_object_ref:n {#1}}
1060       \__tag_struct_prop_gput:nne
1061       { 1 }
1062       { AF }
1063       {
1064         [
1065           \tl_use:c
1066           { g__tag_struct_1_AF_tl }
1067         ]
1068       }
1069     }
1070     {
1071     }
1072   },
1073 }
1074 }

```

**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

```

1075 \keys_define:nn { __tag / setup }
1076 {
1077   root-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 \keys_set:nn
1087 {__tag / setup}
1088 {root-AF={__tag_latex_css_\int_use:N\g__tag_unique_cnt_int}}
1089 \group_end:
1090 }
1091 }

```

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

```

1092 \keys_define:nn { __tag / setup }
1093 {
1094   catalog-supplemental-file .code:n =
1095   {
1096     \group_begin:
1097     \pdfdict_put:nnn {l_pdffile/Filespec} {AFRelationship}{/Supplement}
1098     \int_gincr:N \g__tag_unique_cnt_int
1099     \pdffile_embed_file:eee
1100     {#1}
1101     {#1}
1102     {__tag_latex_css_\int_use:N\g__tag_unique_cnt_int}
1103     \pdfmanagement_add:nne
1104     {Catalog}
1105     {AF}
1106     {\pdf_object_ref:e{__tag_latex_css_\int_use:N\g__tag_unique_cnt_int }}
1107     \group_end:
1108   }
1109 }

```

## 6 User commands

We allow to set a language by default

`\l__tag_struct_lang_tl`

```

1110 \tl_new:N \l__tag_struct_lang_tl
1111 \</package>
(End of definition for \g__tag_struct_AFobj_int and \l__tag_struct_lang_tl.)

```

`\tag_struct_begin:n`  
`\tag_struct_end:`

```

1112 <base>\cs_new_protected:Npn \tag_struct_begin:n #1 {\int_gincr:N \c@g__tag_struct_abs_int}
1113 <base>\cs_new_protected:Npn \tag_struct_end:{ }
1114 <base>\cs_new_protected:Npn \tag_struct_end:n{ }
1115 <*package| debug>
1116 <package>\cs_set_protected:Npn \tag_struct_begin:n #1 %#1 key-val
1117 <debug>\cs_set_protected:Npn \tag_struct_begin:n #1 %#1 key-val
1118 {
1119 <package>\__tag_check_if_active_struct:T
1120 <debug>\__tag_check_if_active_struct:TF

```

```

1121     {
1122         \group_begin:
1123         \int_gincr:N \c@g__tag_struct_abs_int
1124         \__tag_prop_new:c { g__tag_struct_\int_eval:n { \c@g__tag_struct_abs_int }_prop }
1125     <debug>         \prop_new:c { g__tag_struct_debug_\int_eval:n { \c@g__tag_struct_abs_int }_prop }
1126         \__tag_seq_new:c { g__tag_struct_kids_\int_eval:n { \c@g__tag_struct_abs_int }_seq }
1127     <debug>         \seq_new:c { g__tag_struct_debug_kids_\int_eval:n { \c@g__tag_struct_abs_int }_seq }
1128         \pdf_object_new_indexed:nn { __tag/struct }
1129         { \c@g__tag_struct_abs_int }
1130         \__tag_struct_prop_gput:nnn
1131         { \int_use:N \c@g__tag_struct_abs_int }
1132         { Type }
1133         { /StructElem }
1134         \tl_if_empty:NF \l__tag_struct_lang_tl
1135         {
1136             \__tag_struct_prop_gput:nne
1137             { \int_use:N \c@g__tag_struct_abs_int }
1138             { Lang }
1139             { (\l__tag_struct_lang_tl) }
1140         }
1141         \__tag_struct_prop_gput:nnn
1142         { \int_use:N \c@g__tag_struct_abs_int }
1143         { Type }
1144         { /StructElem }
1145
1146         \tl_set:Nn \l__tag_struct_stack_parent_tmpa_tl {-1}
1147         \keys_set:nn { __tag / struct } { #1 }
1148
1149         \__tag_struct_set_tag_info:eoo
1150         { \int_use:N \c@g__tag_struct_abs_int }
1151         { \g__tag_struct_tag_tl }
1152         { \g__tag_struct_tag_NS_tl }
1153         \__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.

```

1153         \int_compare:nNnT { \l__tag_struct_stack_parent_tmpa_tl } = { -1 }
1154         {
1155             \seq_get:NNF
1156             \g__tag_struct_stack_seq
1157             \l__tag_struct_stack_parent_tmpa_tl
1158             {
1159                 \msg_error:nn { tag } { struct-faulty-nesting }
1160             }
1161         }
1162         \seq_gpush:NV \g__tag_struct_stack_seq \c@g__tag_struct_abs_int
1163         \__tag_role_get:ooNN
1164         { \g__tag_struct_tag_tl }
1165         { \g__tag_struct_tag_NS_tl }
1166         \l__tag_struct_roletag_tl
1167         \l__tag_struct_roletag_NS_tl

```

We push the role tag on the stack:

```

1168         \seq_gpush:Ne \g__tag_struct_tag_stack_seq
1169         {{\g__tag_struct_tag_tl}{\l__tag_struct_roletag_tl}}

```

```

1170     \seq_gpush:Ne \g__tag_struct_roletag_stack_seq
1171     {\l__tag_struct_roletag_t1}
1172     \tl_gset:NV \g__tag_struct_stack_current_t1 \c@g__tag_struct_abs_int
1173     \__tag_struct_set_attribute:
1174     %\seq_show:N \g__tag_struct_stack_seq

```

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

```

1175     \__tag_struct_prop_gput:nne
1176     { \int_use:N \c@g__tag_struct_abs_int }
1177     { rolemap }
1178     {
1179       {\l__tag_struct_roletag_t1}{\l__tag_struct_roletag_NS_t1}
1180     }

```

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.

```

1181     \str_case:onTF { \l__tag_struct_roletag_t1 }
1182     {
1183       {Part} {}
1184       {Div} {}
1185       {NonStruct} {}
1186     }
1187     {
1188       \bool_if:NTF \l__tag_struct_elem_stash_bool
1189       {
1190         \__tag_struct_prop_gput:nne
1191         { \int_use:N \c@g__tag_struct_abs_int }
1192         { parentrole }
1193         {
1194           {\l__tag_struct_parenttag_t1}{\l__tag_struct_parenttag_NS_t1}
1195         }
1196       }
1197       {
1198         \prop_get:cnNT
1199         { g__tag_struct_ \l__tag_struct_stack_parent_tmpa_t1 _prop }
1200         { parentrole }
1201         \l__tag_get_tmpc_t1
1202         {
1203           \__tag_struct_prop_gput:nno
1204           { \int_use:N \c@g__tag_struct_abs_int }
1205           { parentrole }
1206           {
1207             \l__tag_get_tmpc_t1
1208           }
1209         }
1210       }
1211     }
1212     {
1213       \__tag_struct_prop_gput:nne
1214       { \int_use:N \c@g__tag_struct_abs_int }

```

```

1215         { parentrole }
1216     {
1217         {\l__tag_struct_roletag_tl}{\l__tag_struct_roletag_NS_tl}
1218     }
1219 }
1220 \bool_if:NF
1221     \l__tag_struct_elem_stash_bool
1222 {

```

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.

```

1223     \socket_use:nn{tag/check/parent-child}
1224     {
1225         \__tag_struct_check_parent_child:oo
1226         { \l__tag_struct_stack_parent_tmpa_tl }
1227         { \int_use:N \c@g__tag_struct_abs_int }
1228     }

```

Set the Parent structure number.

```

1229     \__tag_struct_prop_gput:nne
1230     { \int_use:N \c@g__tag_struct_abs_int }
1231     { parentnum }
1232     {
1233         \l__tag_struct_stack_parent_tmpa_tl
1234     }
1235 %record this structure as kid:
1236 %\tl_show:N \g__tag_struct_stack_current_tl
1237 %\tl_show:N \l__tag_struct_stack_parent_tmpa_tl
1238 \use:c { __tag_struct_kid_struct_gput_ \l__tag_struct_addkid_tl :ee }
1239     { \l__tag_struct_stack_parent_tmpa_tl }
1240     { \g__tag_struct_stack_current_tl }
1241 %\prop_show:c { g__tag_struct_\g__tag_struct_stack_current_tl _prop }
1242 %\seq_show:c {g__tag_struct_kids_\l__tag_struct_stack_parent_tmpa_tl _seq}
1243 }

```

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.

```

1244 <debug>         \prop_gset_eq:cc
1245 <debug>         { g__tag_struct_debug_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
1246 <debug>         { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
1247 <debug>         \prop_gput:cne
1248 <debug>         { g__tag_struct_debug_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
1249 <debug>         { parentnum }
1250 <debug>         {
1251 <debug>             \bool_if:NTF \l__tag_struct_elem_stash_bool
1252 <debug>             {no~parent:~stashed}
1253 <debug>             {
1254 <debug>                 \l__tag_struct_stack_parent_tmpa_tl\c_space_tl =~
1255 <debug>                 \prop_item:cn{ g__tag_struct_\l__tag_struct_stack_parent_tmpa_tl _p
1256 <debug>             }
1257 <debug>         }
1258 <debug>         \prop_gput:cne
1259 <debug>         { g__tag_struct_debug_\int_eval:n {\c@g__tag_struct_abs_int}_prop }

```

```

1260 <debug>                { NS }
1261 <debug>                { \g__tag_struct_tag_NS_t1 }
1262                %\prop_show:c { g__tag_struct \g__tag_struct_stack_current_t1 _prop }
1263                %\seq_show:c { g__tag_struct_kids \l__tag_struct_stack_parent_tmpa_t1 _seq }
1264 <debug> \__tag_debug_struct_begin_insert:n { #1 }
1265                \group_end:
1266        }
1267 <debug>{ \__tag_debug_struct_begin_ignore:n { #1 } }
1268        }
1269 <package>\cs_set_protected:Nn \tag_struct_end:
1270 <debug>\cs_set_protected:Nn \tag_struct_end:
1271        { %take the current structure num from the stack:
1272          %the objects are written later, lua mode hasn't all needed info yet
1273          %\seq_show:N \g__tag_struct_stack_seq
1274 <package>\__tag_check_if_active_struct:T
1275 <debug>\__tag_check_if_active_struct:TF
1276        {
1277          \seq_gpop:NN \g__tag_struct_roletag_stack_seq \l__tag_tmpa_t1
1278          \seq_gpop:NN \g__tag_struct_tag_stack_seq \l__tag_tmpa_t1
1279          \seq_gpop:NNTF \g__tag_struct_stack_seq \l__tag_tmpa_t1
1280          {
1281            \__tag_check_info_closing_struct:o { \g__tag_struct_stack_current_t1 }
1282          }
1283          { \__tag_check_no_open_struct: }
1284          % get the previous one, shouldn't be empty as the root should be there
1285          \seq_get:NNTF \g__tag_struct_stack_seq \l__tag_tmpa_t1
1286          {
1287            \tl_gset:No \g__tag_struct_stack_current_t1 { \l__tag_tmpa_t1 }
1288            \__tag_struct_set_attribute:
1289          }
1290          {
1291            \__tag_check_no_open_struct:
1292          }
1293          \seq_get:NNT \g__tag_struct_tag_stack_seq \l__tag_tmpa_t1
1294          {
1295            \tl_gset:Ne \g__tag_struct_tag_t1
1296            { \exp_last_unbraced:No\use_i:nn { \l__tag_tmpa_t1 } }
1297            \prop_get:NoNT\g__tag_role_tags_NS_prop { \g__tag_struct_tag_t1 } \l__tag_tmpa_t1
1298            {
1299              \tl_gset:Ne \g__tag_struct_tag_NS_t1 { \l__tag_tmpa_t1 }
1300            }
1301          }
1302 <debug>\__tag_debug_struct_end_insert:
1303        }
1304 <debug>{ \__tag_debug_struct_end_ignore: }
1305        }
1306
1307 \cs_set_protected:Npn \tag_struct_end:n #1
1308 {
1309 <debug> \__tag_check_if_active_struct:T{ \__tag_debug_struct_end_check:n{#1} }
1310 \tag_struct_end:
1311 }
1312 </package | debug>

```

(End of definition for \tag\_struct\_begin:n and \tag\_struct\_end:. These functions are documented



on page 110.)

`\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.

```

1313 <base>\cs_new_protected:Npn \tag_struct_use:n #1 {}
1314 <*package| debug>
1315 \cs_set_protected:Npn \tag_struct_use:n #1 %#1 is the label
1316 {
1317   \__tag_check_if_active_struct:T
1318   {
1319     \prop_if_exist:cTF
1320     { g__tag_struct_\property_ref:enn{tagpdfstruct-#1}{tagstruct}{unknown}_prop } %
1321     {
1322       \__tag_check_struct_used:n {#1}
1323       \tl_set:Nx \l__tag_get_child_tmpa_tl
1324       { \property_ref:enn{tagpdfstruct-#1}{tagstruct}{1} }

```

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

```

1325       \__tag_struct_kid_struct_gput_right:ee
1326       { \g__tag_struct_stack_current_tl }
1327       { \l__tag_get_child_tmpa_tl }

```

add the current structure to the labeled one as parents

```

1328       \__tag_prop_gput:cne
1329       { g__tag_struct_\l__tag_get_child_tmpa_tl _prop }
1330       { parentnum }
1331       {
1332         \g__tag_struct_stack_current_tl
1333       }

```

debug code

```

1334 <debug>         \prop_gput:cne
1335 <debug>         { g__tag_struct_debug_\l__tag_get_child_tmpa_tl _prop }
1336 <debug>         { parentnum }
1337 <debug>         {
1338 <debug>           \g__tag_struct_stack_current_tl\c_space_tl=~
1339 <debug>           \g__tag_struct_tag_tl
1340 <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.

```

1341       \socket_use:nn{tag/check/parent-child}
1342       {
1343         \__tag_struct_use_check_parent_child:oo
1344         { \g__tag_struct_stack_current_tl }
1345         { \l__tag_get_child_tmpa_tl }
1346       }
1347     }
1348   {
1349     \msg_warning:nnn{ tag }{struct-label-unknown}{#1}
1350   }
1351 }

```

```

1352 }
1353 </package | debug>
(End of definition for \tag_struct_use:n. This function is documented on page 110.)

```

**\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.

```

1354 <base>\cs_new_protected:Npn \tag_struct_use_num:n #1 {}
1355 <*package | debug>
1356 \cs_set_protected:Npn \tag_struct_use_num:n #1 %#1 is structure number
1357 {
1358   \__tag_check_if_active_struct:T
1359   {
1360     \prop_if_exist:cTF
1361     { g__tag_struct_#1_prop } %
1362     {
1363       \prop_get:cnNT
1364       {g__tag_struct_#1_prop}
1365       {parentnum}
1366       \l__tag_tmpa_tl
1367       {
1368         \msg_warning:nnn { tag } {struct-used-twice} {#1}
1369       }

```

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

```

1370   \__tag_struct_kid_struct_gput_right:ee
1371   { \g__tag_struct_stack_current_tl }
1372   { #1 }

```

add the current structure to #1 as parent

```

1373   \__tag_struct_prop_gput:nne
1374   { #1 }
1375   { parentnum }
1376   {
1377     \g__tag_struct_stack_current_tl
1378   }
1379 <debug> \prop_gput:cne
1380 <debug> { g__tag_struct_debug_#1_prop }
1381 <debug> { parentnum }
1382 <debug> {
1383 <debug> \g__tag_struct_stack_current_tl\c_space_tl=~
1384 <debug> \g__tag_struct_tag_tl
1385 <debug> }

```

check if the tag is allowed as child.

```

1386   \socket_use:nn{tag/check/parent-child}
1387   {
1388     \__tag_struct_use_check_parent_child:oo
1389     {\g__tag_struct_stack_current_tl}
1390     {#1}
1391   }
1392 }
1393 {
1394   \msg_warning:nnn{ tag }{struct-label-unknown}{#1}
1395 }

```

```

1396     }
1397   }
1398   \cs_generate_variant:Nn\tag_struct_use_num:n {o}
1399   \</package>| debug>

```

(End of definition for \tag\_struct\_use\_num:n. This function is documented on page 110.)

**\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.

```

1400 \<package>
1401 \cs_new:Npn \tag_struct_object_ref:n #1
1402 {
1403   \pdf_object_ref_indexed:nn {__tag/struct}{ #1 }
1404 }
1405 \cs_generate_variant:Nn \tag_struct_object_ref:n {e}
1406 \</package>

```

(End of definition for \tag\_struct\_object\_ref:n. This function is documented on page 110.)

**\tag\_struct\_gput:nnn** This is a command that allows to update the data of a structure. This often can't be done simply by replacing the value, as we have to preserve and extend existing content. We use therefore dedicated functions adjusted to the key in question. The first argument is the number of the structure, the second a keyword referring to a function, the third the value. Currently the existing keywords are 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\_dest\_parent** expects a destination name set with \MakeLinkTarget too. It then will refer to the parent of structure in which this \MakeLinkTarget was used—this is useful if the target command is, e.g., in a list and one wants to reference the surrounding list item structure. The keyword **ref\_num** expects a structure number. At last there is the keyword **attribute** which allows to add or extend the /A key of the structure. The value is the content of one attribute dictionary, so for example /O /Layout /BBox [10 10 50 50]. The content is stored in an object and the object reference is then added to the /A.

```

1407 \<base>\cs_new_protected:Npn \tag_struct_gput:nnn #1 #2 #3{
1408 \<package>
1409 \cs_set_protected:Npn \tag_struct_gput:nnn #1 #2 #3
1410 {
1411   \cs_if_exist_use:cF {__tag_struct_gput_data_#2:nn}
1412   { %warning??
1413     \use_none:nn
1414   }
1415   {#1}{#3}
1416 }
1417 \cs_generate_variant:Nn \tag_struct_gput:nnn {ene,nne}
1418 \</package>

```

(End of definition for \tag\_struct\_gput:nnn. This function is documented on page 111.)

**\\_\_tag\_struct\_gput\_data\_ref\_aux:nnn**

```

1419 \<package>
1420 \cs_new_protected:Npn \__tag_struct_gput_data_ref_aux:nnn #1 #2 #3

```

```

1421 % #1 receiving struct num, #2 key word #3 value
1422 {
1423   \prop_get:cnNTF
1424   { g__tag_struct_#1_prop }
1425   {Ref}
1426   \l__tag_get_tmpc_tl
1427   {
1428     \tl_put_right:No \l__tag_get_tmpc_tl
1429     {\cs:w __tag_struct_Ref_#2:nN \cs_end: {#3},}
1430   }
1431   {
1432     \tl_set:No \l__tag_get_tmpc_tl
1433     {\cs:w __tag_struct_Ref_#2:nN \cs_end: {#3},}
1434   }
1435   \__tag_struct_prop_gput:nno
1436   { #1 }
1437   { Ref }
1438   { \l__tag_get_tmpc_tl }
1439 }
1440 \cs_new_protected:Npn \__tag_struct_gput_data_ref:nn #1 #2
1441 {
1442   \__tag_struct_gput_data_ref_aux:nnn {#1}{obj}{#2}
1443 }
1444 \cs_new_protected:Npn \__tag_struct_gput_data_ref_label:nn #1 #2
1445 {
1446   \__tag_struct_gput_data_ref_aux:nnn {#1}{label}{#2}
1447 }
1448 \cs_new_protected:Npn \__tag_struct_gput_data_ref_dest:nn #1 #2
1449 {
1450   \__tag_struct_gput_data_ref_aux:nnn {#1}{dest}{#2}
1451 }
1452 \cs_new_protected:Npn \__tag_struct_gput_data_ref_dest_parent:nn #1 #2
1453 {
1454   \__tag_struct_gput_data_ref_aux:nnn {#1}{dest_parent}{#2}
1455 }
1456 \cs_new_protected:Npn \__tag_struct_gput_data_ref_num:nn #1 #2
1457 {
1458   \__tag_struct_gput_data_ref_aux:nnn {#1}{num}{#2}
1459 }
1460
1461 \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

1462 \cs_new_protected:Npn \__tag_struct_gput_data_attribute:nn #1 #2
1463 {
1464   \pdf_object_unnamed_write:nn {dict} {#2}
1465   \prop_get:cnNTF { g__tag_struct_#1_prop }{A} \l__tag_tmpa_tl
1466   {
1467     \tl_remove_once:Nn\l__tag_tmpa_tl{[]}
1468     \tl_remove_once:Nn\l__tag_tmpa_tl{[]}]
1469     \__tag_struct_gput:cne { g__tag_struct_#1_prop }
1470     { A }
1471     {

```

```

1472         [ \l__tag_tmpa_tl \c_space_tl \pdf_object_ref_last: ]
1473     }
1474 }
1475 {
1476     \__tag_prop_gput:cne { g__tag_struct_#1_prop }
1477     { A }
1478     { \pdf_object_ref_last: }
1479 }
1480 }

```

(End of definition for \\_\_tag\_struct\_gput\_data\_attribute:nn.)

`\tag_struct_insert_annot:nn`  
`\tag_struct_insert_annot:ee`  
`\tag_struct_insert_annot:ee`  
`\tag_struct_parent_int:`

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:`. 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.

```

1481 \cs_new_protected:Npn \tag_struct_insert_annot:nn #1 #2 %#1 should be an object reference
1482                                     %#2 struct parent num
1483 {
1484     \__tag_check_if_active_struct:T
1485     {
1486         \__tag_struct_insert_annot:nn {#1}{#2}
1487     }
1488 }
1489
1490 \cs_generate_variant:Nn \tag_struct_insert_annot:nn {xx,ee}
1491 \cs_new:Npn \tag_struct_parent_int: {\int_use:c { c@g__tag_parenttree_obj_int }}
1492
1493 \end{package}
1494

```

(End of definition for `\tag_struct_insert_annot:nn` and `\tag_struct_parent_int:`. These functions are documented on page 110.)

## 7 Attributes and attribute classes

```

1495 \<header>
1496 \ProvidesExplPackage {tagpdf-attr-code} {2026-04-24} {1.0b}
1497 {part of tagpdf - code related to attributes and attribute classes}
1498 \end{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`

```

1499 \<package>
1500 \prop_new:N \g__tag_attr_entries_prop
1501 \prop_new_linked:N \g__tag_attr_class_used_prop
1502 \tl_new:N \l__tag_attr_value_tl
1503 \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.

```
1504 \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
  role/new-attribute (setup-key)
newattribute (deprecated)
```

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

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},
}

1505 \cs_new_protected:Npn \__tag_attr_new_entry:nn #1 #2 {%#1:name, #2: content
1506 {
1507   \prop_gput:Nen \g__tag_attr_entries_prop
1508   {\pdf_name_from_unicode_e:n{#1}}{#2}
1509 }
1510
1511 \cs_generate_variant:Nn \__tag_attr_new_entry:nn {ee}
1512 \keys_define:nn { __tag / setup }
1513 {
1514   role/new-attribute .code:n =
1515   {
1516     \__tag_attr_new_entry:nn #1
1517   }
1518
  deprecated name
1518   ,newattribute .code:n =
1519   {
1520     \__tag_attr_new_entry:nn #1
1521   },
1522 }
```

(End of definition for \\_\_tag\_attr\_new\_entry:nn, role/new-attribute (setup-key), and newattribute (deprecated). These functions are documented on page 113.)

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

```
1523 \keys_define:nn { __tag / struct }
1524 {
1525   attribute-class .code:n =
1526   {
1527     \clist_set:Ne \l__tag_tmpa_clist { #1 }
1528     \seq_set_from_clist:NN \l__tag_tmpb_seq \l__tag_tmpa_clist
```

we convert the names into pdf names with slash

```

1529 \seq_set_map_e:NNn \l__tag_tmpa_seq \l__tag_tmpb_seq
1530 {
1531   \pdf_name_from_unicode_e:n {##1}
1532 }
1533 \seq_map_inline:Nn \l__tag_tmpa_seq
1534 {
1535   \prop_get:NnNF \g__tag_attr_entries_prop {##1}\l__tag_tmpa_tl
1536   {
1537     \msg_error:nnn { tag } { attr-unknown } { ##1 }
1538   }
1539   \prop_gput:Nnn\g__tag_attr_class_used_prop { ##1} {}
1540 }
1541 \tl_set:Ne \l__tag_tmpa_tl
1542 {
1543   \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}
1544   \seq_use:Nn \l__tag_tmpa_seq { \c_space_tl }
1545   \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}
1546 }
1547 \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 0 }
1548 {
1549   __tag_struct_prop_gput:nne
1550   { \int_use:N \c@g__tag_struct_abs_int }
1551   { C }
1552   { \l__tag_tmpa_tl }
1553   %\prop_show:c { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
1554 }
1555 }
1556 }

```

attribute (struct key)

```

1557 \keys_define:nn { __tag / struct }
1558 {
1559   attribute .code:n = % A property (attribute, value currently a dictionary)
1560   {
1561     \clist_set:Ne \l__tag_tmpa_clist { #1 }
1562     \clist_if_empty:NF \l__tag_tmpa_clist
1563     {
1564       \seq_set_from_clist:NN \l__tag_tmpb_seq \l__tag_tmpa_clist

```

we convert the names into pdf names with slash

```

1565 \seq_set_map_e:NNn \l__tag_tmpa_seq \l__tag_tmpb_seq
1566 {
1567   \pdf_name_from_unicode_e:n {##1}
1568 }
1569 \tl_set:Ne \l__tag_attr_value_tl
1570 {
1571   \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[%]
1572 }
1573 \seq_map_inline:Nn \l__tag_tmpa_seq
1574 {
1575   \prop_get:NnNF \g__tag_attr_entries_prop {##1}\l__tag_tmp_unused_tl
1576   {
1577     \msg_error:nnn { tag } { attr-unknown } { ##1 }

```

```

1578     }
1579     \prop_get:NnNF \g__tag_attr_objref_prop {##1}\l__tag_tmpa_tl
1580     {%\prop_show:N \g__tag_attr_entries_prop
1581       \pdf_object_unnamed_write:ne
1582       { dict }
1583       {
1584         \prop_item:Nn\g__tag_attr_entries_prop {##1}
1585       }
1586       \prop_gput:Nne \g__tag_attr_objref_prop {##1} {\pdf_object_ref_last:}
1587     }
1588     \tl_put_right:Ne \l__tag_attr_value_tl
1589     {
1590       \c_space_tl
1591       \prop_item:Nn \g__tag_attr_objref_prop {##1}
1592     }
1593     % \tl_show:N \l__tag_attr_value_tl
1594     }
1595     \tl_put_right:Ne \l__tag_attr_value_tl
1596     { %[
1597       \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{]}%
1598     }
1599     % \tl_show:N \l__tag_attr_value_tl
1600     \__tag_struct_prop_gput:nne
1601     { \int_use:N \c@g__tag_struct_abs_int }
1602     { A }
1603     { \l__tag_attr_value_tl }
1604   }
1605 },
1606 }
1607 </package>

```

Ulrike Fischer  
Version 1.0b, released 2026-04-24



## Part IX

# The tagpdf driver for luatex

```
1 <@@=tag>
2 <*luatex>
3 \ProvidesExplFile {tagpdf-luatex.def} {2026-04-24} {1.0b}
4   {tagpdf~driver~for~luatex}
```

## 1 Loading the lua

The space code requires that the fall back font has been loaded and initialized, so we force that first. But perhaps this could be done in the kernel.

```
5 {
6   \fontencoding{TU}\fontfamily{lmr}\fontseries{m}\fontshape{n}\fontsize{10pt}{10pt}\selectfont
7 }
8 \lua_now:e { tagpdf=require('tagpdf.lua') }
```

The following defines wrappers around prop and seq commands to store the data also in lua tables. I probably want also lua tables I put them in the ltx.@@.tables namespaces. The tables will be named like the variables but without backslash. To access such a table with a dynamical name create a string and then use ltx.@@.tables[string]. Old code, I'm not quite sure if this was a good idea. Now I have mix of table in ltx.@@.tables and ltx.@@.mc/struct. And a lot is probably not needed. TODO: this should be cleaned up, but at least roles are currently using the table!

```

    \__tag_prop_new:N
    \__tag_seq_new:N
    \__tag_prop_gput:Nnn
    \__tag_seq_gput_right:Nn
    \__tag_seq_gput_left:Nn
    \__tag_seq_item:cn
    \__tag_prop_item:cn
    \__tag_seq_show:N
    \__tag_prop_show:N
9 \cs_set_protected:Npn \__tag_prop_new:N #1
10 {
11   \prop_new:N #1
12   \lua_now:e { ltx.__tag.tables['\cs_to_str:N#1'] = {} }
13 }
14
15 \cs_set_protected:Npn \__tag_prop_new_linked:N #1
16 {
17   \prop_new_linked:N #1
18   \lua_now:e { ltx.__tag.tables['\cs_to_str:N#1'] = {} }
19 }
20
21
22 \cs_set_protected:Npn \__tag_seq_new:N #1
23 {
24   \seq_new:N #1
25   \lua_now:e { ltx.__tag.tables['\cs_to_str:N#1'] = {} }
26 }
27
28
29 \cs_set_protected:Npn \__tag_prop_gput:Nnn #1 #2 #3
30 {
31   \prop_gput:Nnn #1 { #2 } { #3 }
32   \lua_now:e { ltx.__tag.tables['\cs_to_str:N#1'] ["#2"] = "\lua_escape:n{#3}" }
33 }
```

```

34
35 \cs_set_protected:Npn \__tag_seq_gput_right:Nn #1 #2
36 {
37   \seq_gput_right:Nn #1 { #2 }
38   \lua_now:e { table.insert(ltx.__tag.tables['\cs_to_str:N#1'], "#2") }
39 }

```

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

```

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

```

(End of definition for \\_\_tag\_prop\_new:N and others.)

```

71 \</luatex>

```

The module declaration

```

72 <*lua>
73 -- tagpdf.lua
74 -- Ulrike Fischer
75
76 local ProvidesLuaModule = {
77   name      = "tagpdf",
78   version   = "1.0b",      --TAGVERSION
79   date      = "2026-04-24", --TAGDATE
80   description = "tagpdf lua code",
81   license   = "The LATEX Project Public License 1.3c"

```

```

82 }
83
84 if luatexbase and luatexbase.provides_module then
85   luatexbase.provides_module (ProvidesLuaModule)
86 end
87
88 --[[
89 The code has quite probably a number of problems
90 - more variables should be local instead of global
91 - the naming is not always consistent due to the development of the code
92 - the traversing of the shipout box must be tested with more complicated setups
93 - it should probably handle more node types
94 -
95 --]]
96

```

Some comments about the lua structure.

```

97 --[[
98 the main table is named ltx.__tag. It contains the functions and also the data
99 collected during the compilation.
100
101 ltx.__tag.mc      will contain mc connected data.
102 ltx.__tag.role    will contain data related to parent-child relations.
103 ltx.__tag.struct  will contain structure related data.
104 ltx.__tag.page    will contain page data
105 ltx.__tag.tables  contains also data from mc and struct (from older code). This needs cleaning
106                 There are certainly dublettes, but I don't dare yet ...
107 ltx.__tag.func    will contain (public) functions.
108 ltx.__tag.trace   will contain tracing/logging functions.
109 local functions starts with __
110 functions meant for users will be in ltx.tag
111
112 functions
113 ltx.__tag.func.get_num_from (tag):   takes a tag (string) and returns the id number
114 ltx.__tag.func.output_num_from (tag): takes a tag (string) and prints (to tex) the id number
115 ltx.__tag.func.get_tag_from (num):   takes a num and returns the tag
116 ltx.__tag.func.output_tag_from (num): takes a num and prints (to tex) the tag
117 ltx.__tag.func.store_mc_data (num,key,data): stores key=data in ltx.__tag.mc[num]
118 ltx.__tag.func.store_mc_label (label,num): stores label=num in ltx.__tag.mc.labels
119 ltx.__tag.func.store_mc_kid (mcnum,kid,page): stores the mc-kids of mcnum on page page
120 ltx.__tag.func.store_mc_in_page(mcnum,mcpagecnt,page): stores in the page table the number of
121 ltx.__tag.func.store_struct_mcabs (structnum,mcnum): stores relations structnum<->mcnum (abs
122 ltx.__tag.func.mc_insert_kids (mcnum): inserts the /K entries for mcnum by wandering through
123 ltx.__tag.func.mark_page_elements(box,mcpagecnt,mccntprev,mcopen,name,mctypeprev) : the main
124 ltx.__tag.func.mark_shipout (): a wrapper around the core function which inserts the last EM
125 ltx.__tag.func.fill_parent_tree_line (page): outputs the entries of the parenttree for this
126 ltx.__tag.func.output_parenttree(): outputs the content of the parenttree
127 ltx.__tag.func.pdf_object_ref(name,index): outputs the object reference for the object name
128 ltx.__tag.func.markspaceon(), ltx.__tag.func.markspaceoff(): (de)activates the marking of po
129 ltx.__tag.func.linkbin() returns the number of OBJR stored in the linkbin structure
130 ltx.__tag.trace.show_mc_data (num,loglevel): shows ltx.__tag.mc[num] is the current log leve
131 ltx.__tag.trace.show_all_mc_data (max,loglevel): shows a maximum about mc's if the current l
132 ltx.__tag.trace.show_seq: shows a sequence (array)
133 ltx.__tag.trace.show_struct_data (num): shows data of structure num
134 ltx.__tag.trace.show_prop: shows a prop

```

```

135 ltx.__tag.trace.log
136 ltx.__tag.trace.showspace : boolean
137
138 ltx.tag.get_structnum: number, shows the current structure number
139 ltx.tag.get_structnum_next: number, shows the next structure number
140 --]]
141

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

## 2 User commands to access data

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

\

```

205 local __tag_get_struct_num =
206 function()
207   local a = token.get_macro("g__tag_struct_stack_current_tl")
208   return a
209 end
210
211 local __tag_get_struct_counter =
212 function()

```

```

213   local a = tex.getcount("c@g__tag_struct_abs_int")
214   return a
215 end
216
217 local __tag_get_struct_num_next =
218   function()
219     local a = tex.getcount("c@g__tag_struct_abs_int") + 1
220     return a
221   end
222
223 ltx.tag.get_struct_num = __tag_get_struct_num
224 ltx.tag.get_struct_counter = __tag_get_struct_counter
225 ltx.tag.get_struct_num_next = __tag_get_struct_num_next
(End of definition for \. This function is documented on page ??.)

```

### 3 Logging functions

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

```

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

```

`ltx.__tag.trace.show_seq` This shows the content of a seq as stored in the tables table. It is used by the `\@@_seq_show:N` function. It is not used in user commands, only for debugging, and so requires log level >0.

```

234 function ltx.__tag.trace.show_seq (seq)
235   if (type(seq) == "table") then
236     for i,v in ipairs(seq) do
237       __tag_log ("[" .. i .. "] => " .. tostring(v),1)
238     end
239   else
240     __tag_log ("sequence " .. tostring(seq) .. " not found",1)
241   end
242 end
(End of definition for ltx.__tag.trace.show_seq.)

```

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

```

243 local __tag_pairs_prop =
244   function (prop)
245     local a = {}
246     for n in pairs(prop) do tableinsert(a, n) end
247     table.sort(a)
248     local i = 0                -- iterator variable

```

```

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

```

(End of definition for \_\_tag\_pairs\_prop and ltx.\_\_tag.trace.show\_prop.)

ltx.\_\_tag.trace.show\_mc\_data This shows some data for a mc given by num. If something is shown depends on the log level. The function is used by the following function and then in \ShowTagging

```

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

```

(End of definition for ltx.\_\_tag.trace.show\_mc\_data.)

ltx.\_\_tag.trace.show\_all\_mc\_data This shows data for the mc's between min and max (numbers). It is used by the \ShowTagging function.

```

283 function ltx.__tag.trace.show_all_mc_data (min,max,loglevel)
284 for i = min, max do
285     ltx.__tag.trace.show_mc_data (i,loglevel)
286 end
287 texio.write_nl("")
288 end

```

(End of definition for ltx.\_\_tag.trace.show\_all\_mc\_data.)

ltx.\_\_tag.trace.show\_struct\_data This function shows some struct data. Unused but kept for debugging.

```

289 function ltx.__tag.trace.show_struct_data (num)
290 if ltx.__tag and ltx.__tag.struct and ltx.__tag.struct[num] then

```

```

291   for k,v in ipairs(ltx.__tag.struct[num]) do
292     __tag_log  ("struct "..num..": "..tostring(k).."=>"..tostring(v),1)
293   end
294   else
295     __tag_log  ("struct "..num.." not found ",1)
296   end
297 end
(End of definition for ltx.__tag.trace.show_struct_data.)

```

## 4 Helper functions

### 4.1 Retrieve data functions

`__tag_get_mc_cnt_type_tag` This takes a node as argument and returns the mc-cnt, the mc-type and and the tag (calculated from the mc-cnt.

```

298 local __tag_get_mc_cnt_type_tag = function (n)
299   local mcnt      = nodegetattribute(n,mccntattributeid) or -1
300   local mctype    = nodegetattribute(n,mctypeattributeid) or -1
301   local tag       = ltx.__tag.func.get_tag_from(mctype)
302   return mcnt,mctype,tag
303 end
(End of definition for __tag_get_mc_cnt_type_tag.)

```

`__tag_get_mathsubtype` This function allows to detect if we are at the begin or the end of math. It takes as argument a mathnode.

```

304 local function __tag_get_mathsubtype (mathnode)
305   if mathnode.subtype == 0 then
306     subtype = "beginmath"
307   else
308     subtype = "endmath"
309   end
310   return subtype
311 end
(End of definition for __tag_get_mathsubtype.)

```

`ltx.__tag.tables.role_tag_attribute` The first is a table with key a tag and value a number (the attribute) The second is an array with the attribute value as key.

```

312 ltx.__tag.tables.role_tag_attribute = {}
313 ltx.__tag.tables.role_attribute_tag = {}
(End of definition for ltx.__tag.tables.role_tag_attribute.)

```

`ltx.__tag.func.alloctag`

```

314 local __tag_alloctag =
315   function (tag)
316     if not ltx.__tag.tables.role_tag_attribute[tag] then
317       table.insert(ltx.__tag.tables.role_attribute_tag,tag)
318       ltx.__tag.tables.role_tag_attribute[tag]=#ltx.__tag.tables.role_attribute_tag
319       __tag_log  ("Add "..tag.." "..ltx.__tag.tables.role_tag_attribute[tag],3)
320     end
321   end
322 ltx.__tag.func.alloctag = __tag_alloctag
(End of definition for ltx.__tag.func.alloctag.)

```



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

```

323 local __tag_get_num_from =
324 function (tag)
325   if ltx.__tag.tables.role_tag_attribute[tag] then
326     a= ltx.__tag.tables.role_tag_attribute[tag]
327   else
328     a= -1
329   end
330   return a
331 end

```

```

332
333 ltx.__tag.func.get_num_from = __tag_get_num_from
334

```

```

335 function ltx.__tag.func.output_num_from (tag)
336   local num = __tag_get_num_from (tag)
337   tex.sprint(catlatex,num)
338   if num == -1 then
339     __tag_log ("Unknown tag "..tag.." used")
340   end
341 end

```

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

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

```

342 local __tag_get_tag_from =
343 function (num)
344   if ltx.__tag.tables.role_attribute_tag[num] then
345     a = ltx.__tag.tables.role_attribute_tag[num]
346   else
347     a= "UNKNOWN"
348   end
349   return a
350 end

```

```

351
352 ltx.__tag.func.get_tag_from = __tag_get_tag_from
353

```

```

354 function ltx.__tag.func.output_tag_from (num)
355   tex.sprint(catlatex,__tag_get_tag_from (num))
356 end

```

(End of definition for `__tag_get_tag_from`, `ltx.__tag.func.get_tag_from`, and `ltx.__tag.func.output_tag_from`.)

`ltx.__tag.func.store_mc_data` This function stores for `key=data` for mc-chunk `num`. It is used in the `tagpdf-mc` code, to store for example the tag string, and the raw options.

```

357 function ltx.__tag.func.store_mc_data (num,key,data)
358   ltx.__tag.mc[num] = ltx.__tag.mc[num] or { }
359   ltx.__tag.mc[num][key] = data
360   __tag_log ("INFO TEX-STORE-MC-DATA: "..num.." => "..tostring(key).. " => "..tostring(data),3)
361 end

```

(End of definition for ltx.\_\_tag.func.store\_mc\_data.)

ltx.\_\_tag.func.store\_mc\_label This function stores the label=num relationship in the labels subtable. TODO: this is probably unused and can go.

```

362 function ltx.__tag.func.store_mc_label (label,num)
363   ltx.__tag.mc["labels"] = ltx.__tag.mc["labels"] or { }
364   ltx.__tag.mc.labels[label] = num
365 end

```

(End of definition for ltx.\_\_tag.func.store\_mc\_label.)

ltx.\_\_tag.func.store\_mc\_kid This function is used in the traversing code. It stores a sub-chunk of a mc mcnum into the kids table.

```

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

```

(End of definition for ltx.\_\_tag.func.store\_mc\_kid.)

ltx.\_\_tag.func.mc\_num\_of\_kids This function returns the number of kids a mc mcnum has. We need to account for the case that a mc can have no kids.

```

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

```

(End of definition for ltx.\_\_tag.func.mc\_num\_of\_kids.)

## 4.2 Functions to insert the pdf literals

\_\_tag\_backend\_create\_emc\_node This insert the emc node. We support also dvips and dvipdfmx backend  
\_\_tag\_insert\_emc\_node

```

380 local __tag_backend_create_emc_node
381 if tex.outputmode == 0 then
382   if token.get_macro("c_sys_backend_str") == "dvipdfmx" then
383     function __tag_backend_create_emc_node ()
384       local emcnode = nodenew("whatsit","special")
385       emcnode.data = "pdf:code EMC"
386       return emcnode
387     end
388   else -- assume a dvips variant
389     function __tag_backend_create_emc_node ()
390       local emcnode = nodenew("whatsit","special")
391       emcnode.data = "ps:SDict begin mark /EMC pdfmark end"
392       return emcnode
393     end
394   end
395 else -- pdf mode
396   function __tag_backend_create_emc_node ()

```

```

397     local emcnode = nodenew("whatsit","pdf_literal")
398     emcnode.data = "EMC"
399     emcnode.mode=1
400     return emcnode
401 end
402 end
403
404 local function __tag_insert_emc_node (head,current)
405     local emcnode= __tag_backend_create_emc_node()
406     head = node.insert_before(head,current,emcnode)
407     return head
408 end
(End of definition for __tag_backend_create_emc_node and __tag_insert_emc_node.)

```

\_\_tag\_backend\_create\_bmc\_node  
\_\_tag\_insert\_bmc\_node

This inserts a simple bmc node

```

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

```

\_\_tag\_backend\_create\_bdc\_node  
\_\_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.

```

438 local __tag_backend_create_bdc_node
439
440 if tex.outputmode == 0 then
441     if token.get_macro("c_sys_backend_str") == "dvipdfmx" then
442         function __tag_backend_create_bdc_node (tag,dict)

```

```

443     local bdcnode = nodenew("whatsit","special")
444     bdcnode.data = "pdf:code /"..tag.."<<"..dict..">> BDC"
445     return bdcnode
446 end
447 else -- assume a dvips variant
448     function __tag_backend_create_bdc_node (tag,dict)
449         local bdcnode = nodenew("whatsit","special")
450         bdcnode.data = "ps:SDict begin mark/"..tag.."<<"..dict..">> /BDC pdfmark end"
451         return bdcnode
452     end
453 end
454 else -- pdf mode
455     function __tag_backend_create_bdc_node (tag,dict)
456         local bdcnode = nodenew("whatsit","pdf_literal")
457         bdcnode.data = "/"..tag.."<<"..dict..">> BDC"
458         bdcnode.mode=1
459         return bdcnode
460     end
461 end
462
463 local function __tag_insert_bdc_node (head,current,tag,dict)
464     bdcnode= __tag_backend_create_bdc_node (tag,dict)
465     head = node.insert_before(head,current,bdcnode)
466     return head
467 end

```

(End of definition for \_\_tag\_backend\_create\_bdc\_node and \_\_tag\_insert\_bdc\_node.)

`__tag_pdf_object_ref` This allows to reference a pdf object reserved with the l3pdf command by name. The return value is `n 0 R`, if the object doesn't exist, `n` is 0.

```

468 local function __tag_pdf_object_ref (name,index)
469     local object
470     if ltx.pdf.object_id then
471         object = ltx.pdf.object_id (name,index) ..' 0 R'
472     else
473         local tokenname = 'c__pdf_object_'..name..'/'..index..'_int'
474         object = token.create(tokenname).mode ..' 0 R'
475     end
476     return object
477 end
478 ltx.__tag.func.pdf_object_ref = __tag_pdf_object_ref

```

(End of definition for \_\_tag\_pdf\_object\_ref.)

## 5 Function for the real space chars

`__tag_show_spacemark` A debugging function, it is used to inserts red color markers in the places where space chars can go, it can have side effects so not always reliable, but ok.

```

479 local function __tag_show_spacemark (head,current,color,height)
480     local markcolor = color or "1 0 0"
481     local markheight = height or 10
482     local pdfstring
483     if tex.outputmode == 0 then
484         -- ignore dvi mode for now

```

```

485 else
486   pdfstring = node.new("whatsit", "pdf_literal")
487   pdfstring.data =
488     string.format("q ..markcolor.." RG "..markcolor.." rg 0.4 w 0 %g m 0 %g l S Q",-
3,markheight)
489   head = node.insert_after(head,current,pdfstring)
490   return head
491 end
492 end
(End of definition for __tag_show_spacemark.)

```

```

__tag_fakespace This is used to define a lua version of \pdf_fakespace
ltx.__tag.func.fakespace
493 local function __tag_fakespace()
494   tex.setattribute(iwspaceattributeid,1)
495   tex.setattribute(iwfontattributeid,font.current())
496 end
497 ltx.__tag.func.fakespace = __tag_fakespace
(End of definition for __tag_fakespace and ltx.__tag.func.fakespace.)

```

`__tag_mark_spaces` a function to mark up places where real space chars should be inserted. It only sets attributes, these are then be used in a later traversing which inserts the actual spaces. When space handling is activated this function is inserted in some callbacks.

```

498 --[[ a function to mark up places where real space chars should be inserted
499       it only sets an attribute.
500 --]]
501
502 local function __tag_mark_spaces (head)
503   local inside_math = false
504   for n in nodetraverse(head) do
505     local id = n.id
506     if id == GLYPH then
507       local glyph = n
508       default_currfontid = glyph.font
509       if glyph.next and (glyph.next.id == GLUE)
510         and not inside_math and (glyph.next.width > 0)
511       then
512         nodesetattribute(glyph.next,iwspaceattributeid,1)
513         nodesetattribute(glyph.next,iwfontattributeid,glyph.font)
514         -- for debugging
515         if ltx.__tag.trace.showspace then
516           __tag_show_spacemark (head,glyph)
517         end
518       elseif glyph.next and (glyph.next.id==KERN) and not inside_math then
519         local kern = glyph.next
520         if kern.next and (kern.next.id== GLUE) and (kern.next.width > 0)
521         -- the attribute is also set on the kern in case the kern+glue is
522         -- discarded at a line break tagging issue #1102
523         -- TODO iterate back through all discardable nodes.
524       then
525         nodesetattribute(kern,iwspaceattributeid,1)
526         nodesetattribute(kern,iwfontattributeid,glyph.font)
527         nodesetattribute(kern.next,iwspaceattributeid,1)
528         nodesetattribute(kern.next,iwfontattributeid,glyph.font)

```

```

529     end
530   end
531   -- look also back
532   if glyph.prev and (glyph.prev.id == GLUE)
533     and not inside_math
534     and (glyph.prev.width >0)
535     and not nodehasattribute(glyph.prev,iwspaceattributeid)
536   then
537     nodesetattribute(glyph.prev,iwspaceattributeid,1)
538     nodesetattribute(glyph.prev,iwfontattributeid,glyph.font)
539   -- for debugging
540   if ltx.__tag.trace.showspace then
541     __tag_show_spacemark (head,glyph)
542   end
543   end
544   elseif id == PENALTY then
545     local glyph = n
546     -- __tag_log ("PENALTY " .. n.subtype .. "VALUE" .. n.penalty,3)
547     if glyph.next and (glyph.next.id == GLUE)
548       and not inside_math and (glyph.next.width >0) and n.subtype==0
549     then
550       nodesetattribute(glyph.next,iwspaceattributeid,1)
551       -- changed 2024-01-18, issue #72
552       nodesetattribute(glyph.next,iwfontattributeid,default_currfontid)
553     -- for debugging
554     if ltx.__tag.trace.showspace then
555       __tag_show_spacemark (head,glyph)
556     end
557     end
558   elseif id == MATH then
559     inside_math = (n.subtype == 0)
560   end
561 end
562 return head
563 end
(End of definition for __tag_mark_spaces.)

```

```

__tag_activate_mark_space 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
564 local function __tag_activate_mark_space ()
565   if not luatexbase.in_callback ("pre_linebreak_filter","markspaces") then
566     luatexbase.add_to_callback("pre_linebreak_filter",__tag_mark_spaces,"markspaces")
567     luatexbase.add_to_callback("hpack_filter",__tag_mark_spaces,"markspaces")
568   end
569 end
570
571 ltx.__tag.func.markspaceon=__tag_activate_mark_space
572
573 local function __tag_deactivate_mark_space ()
574   if luatexbase.in_callback ("pre_linebreak_filter","markspaces") then
575     luatexbase.remove_from_callback("pre_linebreak_filter","markspaces")
576     luatexbase.remove_from_callback("hpack_filter","markspaces")
577   end
578 end

```

```

579
580 ltx.__tag.func.markspaceoff=__tag_deactivate_mark_space
(End of definition for __tag_activate_mark_space, ltx.__tag.func.markspaceon, and ltx.__tag.func.markspaceoff.)

```

We need two local variable to setup a default space char.

```

581 local default_space_char = nodenew(GLYPH)
582 local default_fontid      = fontid("TU/lmr/m/n/10")
583 local default_currfontid = fontid("TU/lmr/m/n/10")
584 default_space_char.char   = 32
585 default_space_char.font   = default_fontid

```

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

```

586 local function __tag_font_has_space (fontid)
587   t= fonts.hashses.identifiers[fontid]
588   if luaotfload.aux.slot_of_name(fontid,"space")
589     or t and t.characters and t.characters[32] and t.characters[32]["unicode"]==32
590   then
591     return true
592   else
593     return false
594   end
595 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.

```

596 local function __tag_space_chars_shipout (box)
597   local head = box.head
598   if head then
599     for n in node.traverse(head) do
600       local spaceattr = -1
601       if not nodehasattribute(n,iwspaceOffattributeid) then
602         spaceattr = nodegetattribute(n,iwspaceattributeid) or -1
603       end
604       if n.id == HLIST then -- enter the hlist
605         __tag_space_chars_shipout (n)
606       elseif n.id == VLIST then -- enter the vlist
607         __tag_space_chars_shipout (n)
608       elseif n.id == GLUE then
609         if ltx.__tag.trace.showspace and spaceattr==1 then
610           __tag_show_spacemark (head,n,"0 1 0")
611         end
612         if spaceattr==1 then
613           local space
614           local space_char = node.copy(default_space_char)
615           local curfont     = nodegetattribute(n,iwfontattributeid)
616           __tag_log ("INFO SPACE-FUNCTION-FONT: ".. tostring(curfont),3)
617           if curfont and
618             -- luaotfload.aux.slot_of_name(curfont,"space")
619             __tag_font_has_space (curfont)
620           then
621             space_char.font=curfont
622           end
623           head, space = node.insert_before(head, n, space_char) --

```

```

624         n.width      = n.width - space.width
625         space.attr    = n.attr
626     end
627 end
628 end
629 box.head = head
630 end
631 end
632
633 function ltx.__tag.func.space_chars_shipout (box)
634     __tag_space_chars_shipout (box)
635 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.

```

636 function ltx.__tag.func.mc_insert_kids (mcnum,single)
637     if ltx.__tag.mc[mcnum] then
638         __tag_log("INFO TEX-MC-INSERT-KID-TEST: " .. mcnum,4)
639         if ltx.__tag.mc[mcnum]["kids"] then
640             if #ltx.__tag.mc[mcnum]["kids"] > 1 and single==1 then
641                 tex.sprint(catlatex,"")
642             end
643             for i,kidstable in ipairs( ltx.__tag.mc[mcnum]["kids"] ) do
644                 local kidnum = kidstable["kid"]
645                 local kidpage = kidstable["page"]
646                 local kidpageobjnum = pdfpageref(kidpage)
647                 __tag_log("INFO TEX-MC-INSERT-KID: " .. mcnum ..
648                     " insert KID " .. i ..
649                     " with num " .. kidnum ..
650                     " on page " .. kidpage.."/"..kidpageobjnum,3)
651                 tex.sprint(catlatex,"<</Type /MCR /Pg " .. kidpageobjnum .. " 0 R /MCID " .. kidnum .. ">> ")
652             end
653             if #ltx.__tag.mc[mcnum]["kids"] > 1 and single==1 then
654                 tex.sprint(catlatex,"")
655             end
656         else
657             -- this is typically not a problem, e.g. empty hbox in footer/header can
658             -- trigger this warning.
659             __tag_log("WARN TEX-MC-INSERT-NO-KIDS: " .. mcnum .. " has no kids",2)
660             if single==1 then
661                 tex.sprint(catlatex,"null")
662             end
663         end
664     else
665         __tag_log("WARN TEX-MC-INSERT-MISSING: " .. mcnum .. " doesn't exist",0)
666     end
667 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.

```

668 function ltx.__tag.func.store_struct_mcabs (structnum,mcnum)
669   ltx.__tag.struct[structnum]=ltx.__tag.struct[structnum] or { }
670   ltx.__tag.struct[structnum]["mc"]=ltx.__tag.struct[structnum]["mc"] or { }
671   -- a structure can contain more than on mc chunk, the content should be ordered
672   tableinsert(ltx.__tag.struct[structnum]["mc"],mcnum)
673   __tag_log("INFO TEX-MC-INTO-STRUCT: "...
674             mcnum.." inserted in struct "..structnum,3)
675   -- but every mc can only be in one structure
676   ltx.__tag.mc[mcnum]= ltx.__tag.mc[mcnum] or { }
677   ltx.__tag.mc[mcnum]["parent"] = structnum
678 end
679

```

(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.

```

680 -- pay attention: lua counts arrays from 1, tex pages from one
681 -- mcid and arrays in pdf count from 0.
682 function ltx.__tag.func.store_mc_in_page (mcnum,mcpagecnt,page)
683   ltx.__tag.page[page] = ltx.__tag.page[page] or {}
684   ltx.__tag.page[page][mcpagecnt] = mcnum
685   __tag_log("INFO TAG-MC-INTO-PAGE: page " .. page ..
686             ": inserting MCID " .. mcpagecnt .. " => " .. mcnum,3)
687 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)

```

688 local function __tag_update_mc_attributes (head,mcnum,type)
689   for n in node.traverse(head) do
690     node.set_attribute(n,mccntattributeid,mcnum)
691     node.set_attribute(n,mctypeattributeid,type)
692     if n.id == HLIST or n.id == VLIST then
693       __tag_update_mc_attributes (n.list,mcnum,type)
694     end
695   end
696   return head
697 end
698 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.

```

699 --[[
700   Now follows the core function
701   It wades through the shipout box and checks the attributes
702   ARGUMENTS
703   box: is a box,
704   mcpagecnt: num, the current page cnt of mc (should start at -1 in shipout box), needed for

```

```

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

```

```

759     __tag_log ("INFO TAG-KERN-SUBTYPE: "..
760         node.type(node.getid(n)).." "..n.subtype,4)
761 else
762     -- math is currently only logged.
763     -- we could mark the whole as math
764     -- for inner processing the mlist_to_hlist callback is probably needed.
765     if n.id == MATH then
766         __tag_log("INFO TAG-MATH-SUBTYPE: "..
767             node.type(node.getid(n)).." "..__tag_get_mathsubtype(n),4)
768     end
769     -- endmath
770     __tag_log("INFO TAG-MC-COMPARE: current "..
771         mccnt.." prev "..mccntprev,4)
772     if mccnt~=mccntprev then -- a new mc chunk
773         __tag_log ("INFO TAG-NEW-MC-NODE: "..
774             node.type(node.getid(n))..
775             " MC"..tostring(mccnt)..
776             " <=> PREVIOUS "..tostring(mccntprev),4)
777     if mcopen~=0 then -- there is a chunk open, close it (hope there is only one ...
778         box.list=__tag_insert_emc_node (box.list,n)
779         mcopen = mcopen - 1
780         __tag_log ("INFO TAG-INSERT-EMC: " ..
781             mcpagencnt .. " MCOPEN = " .. mcopen,3)
782     if mcopen ~=0 then
783         __tag_log ("WARN TAG-OPEN-MC: " .. mcopen,1)
784     end
785 end
786 if ltx.__tag.mc[mccnt] then
787     if ltx.__tag.mc[mccnt]["artifact"] then
788         __tag_log("INFO TAG-INSERT-ARTIFACT: "..
789             tostring(ltx.__tag.mc[mccnt]["artifact"]),3)
790     if ltx.__tag.mc[mccnt]["artifact"] == "" then
791         box.list = __tag_insert_bmc_node (box.list,n,"Artifact")
792     else
793         box.list = __tag_insert_bdc_node (box.list,n,"Artifact", "/Type /"..ltx.__tag.mc[mccnt]
794     end
795 else
796     __tag_log("INFO TAG-INSERT-TAG: "..
797         tostring(tag),3)
798     mcpagencnt = mcpagencnt +1
799     __tag_log ("INFO TAG-INSERT-BDC: "..mcpagencnt,3)
800     local dict= "/MCID "..mcpagencnt
801     if ltx.__tag.mc[mccnt]["raw"] then
802         __tag_log("INFO TAG-USE-RAW: "..
803             tostring(ltx.__tag.mc[mccnt]["raw"]),3)
804         dict= dict .. " " .. ltx.__tag.mc[mccnt]["raw"]
805     end
806     if ltx.__tag.mc[mccnt]["alt"] then
807         __tag_log("INFO TAG-USE-ALT: "..
808             tostring(ltx.__tag.mc[mccnt]["alt"]),3)
809         dict= dict .. " " .. ltx.__tag.mc[mccnt]["alt"]
810     end
811     if ltx.__tag.mc[mccnt]["lang"] then
812         __tag_log("INFO TAG-USE-LANG: "..

```

```

813         tostring(ltx.__tag.mc[mccnt]["lang"]),3)
814         dict= dict .. " " .. ltx.__tag.mc[mccnt]["lang"]
815     end
816     if ltx.__tag.mc[mccnt]["actualtext"] then
817         __tag_log("INFO TAG-USE-ACTUALTEXT: "..
818             tostring(ltx.__tag.mc[mccnt]["actualtext"]),3)
819         dict= dict .. " " .. ltx.__tag.mc[mccnt]["actualtext"]
820     end
821     box.list = __tag_insert_bdc_node (box.list,n,tag, dict)
822     ltx.__tag.func.store_mc_kid (mccnt,mcpagecnt,abspage)
823     ltx.__tag.func.store_mc_in_page(mccnt,mcpagecnt,abspage)
824     ltx.__tag.trace.show_mc_data (mccnt,3)
825 end
826 mcopen = mcopen + 1
827 else
828     if tagunmarkedbool.mode == truebool.mode then
829         __tag_log("INFO TAG-NOT-TAGGED: this has not been tagged, using artifact",2)
830         box.list = __tag_insert_bmc_node (box.list,n,"Artifact")
831         mcopen = mcopen + 1
832     else
833         __tag_log("WARN TAG-NOT-TAGGED: this has not been tagged",1)
834     end
835 end
836 mccntprev = mccnt
837 end
838 end -- end if
839 end -- end for
840 if head then
841     mccnthead, mctypehead,taghead = __tag_get_mc_cnt_type_tag (head)
842     __tag_log ("INFO TAG-ENDHEAD: " ..
843         node.type(node.getid(head))..
844         " MC"..tostring(mccnthead)..
845         " => TAG "..tostring(mctypehead)..
846         " => "..tostring(taghead),4)
847 else
848     __tag_log ("INFO TAG-ENDHEAD: ".. tostring(head),4)
849 end
850 __tag_log ("INFO TAG-QUITTING-BOX "..
851     tostring(name)..
852     " TYPE ".. node.type(node.getid(box)),4)
853 return mcopen,mcpagecnt,mccntprev,mctypeprev
854 end
855
(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.

```

856 function ltx.__tag.func.mark_shipout (box)
857     mcopen = ltx.__tag.func.mark_page_elements (box,-1,-100,0,"Shipout",-1)
858     if mcopen~=0 then -- there is a chunk open, close it (hope there is only one ...
859         local emcnode = __tag_backend_create_emc_node ()
860         local list = box.list
861         if list then

```

```

862     list = node.insert_after (list,node.tail(list),emcnode)
863     mcopen = mcopen - 1
864     __tag_log ("INFO SHIPOUT-INSERT-LAST-EMC: MCOPEN " .. mcopen,3)
865 else
866     __tag_log ("WARN SHIPOUT-UPS: this shouldn't happen",0)
867 end
868 if mcopen ~=0 then
869     __tag_log ("WARN SHIPOUT-MC-OPEN: " .. mcopen,1)
870 end
871 end
872 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.

```

873 function ltx.__tag.func.fill_parent_tree_line (page)
874     -- we need to get page-> i=kid -> mcnum -> structnum
875     -- pay attention: the kid numbers and the page number in the parent tree start with 0!
876     local numsentry = ""
877     local pdfpage = page-1
878     if ltx.__tag.page[page] and ltx.__tag.page[page][0] then
879         mcchunks=#ltx.__tag.page[page]
880         __tag_log("INFO PARENTTREE-NUM: page " ..
881             page.." has "..mcchunks.." +1 Elements ",4)
882         for i=0,mcchunks do
883             -- what does this log??
884             __tag_log("INFO PARENTTREE-CHUNKS: " ..
885                 ltx.__tag.page[page][i],4)
886         end
887         if mcchunks == 0 then
888             -- only one chunk so no need for an array
889             local mcnum = ltx.__tag.page[page][0]
890             local structnum = ltx.__tag.mc[mcnum]["parent"]
891             local propname = "g__tag_struct"..structnum.."__prop"
892             --local objref = ltx.__tag.tables[propname]["objref"] or "XXXX"
893             local objref = __tag_pdf_object_ref('__tag/struct',structnum)
894             __tag_log("INFO PARENTTREE-STRUCT-OBJREF: =====>" ..
895                 tostring(objref),5)
896             numsentry = pdfpage .. " [".. objref .. "]"
897             __tag_log("INFO PARENTTREE-NUMENTRY: page " ..
898                 page.. " num entry = " .. numsentry,3)
899         else
900             numsentry = pdfpage .. " ["
901             for i=0,mcchunks do
902                 local mcnum = ltx.__tag.page[page][i]
903                 local structnum = ltx.__tag.mc[mcnum]["parent"] or 0
904                 local propname = "g__tag_struct"..structnum.."__prop"
905                 --local objref = ltx.__tag.tables[propname]["objref"] or "XXXX"
906                 local objref = __tag_pdf_object_ref('__tag/struct',structnum)
907                 numsentry = numsentry .. " " .. objref
908             end
909         end
910     end

```

```

909     numsentry = numsentry .. "]" "
910     __tag_log("INFO PARENTTREE-NUMENTRY: page " ..
911     page.. " num entry = ".. numsentry,3)
912     end
913 else
914     __tag_log ("INFO PARENTTREE-NO-DATA: page "..page,3)
915     numsentry = pdfpage.." ["
916     end
917     return numsentry
918 end
919
920 function ltx.__tag.func.output_parenttree (abspage)
921 for i=1,abspage do
922     line = ltx.__tag.func.fill_parent_tree_line (i) .. "^^J"
923     tex.sprint(catlatex,line)
924 end
925 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.

```

926 do
927     local properties = node.get_properties_table()
928     local is_soft_hyphen_prop = 'tagpdf.rewrite-softhyphen.is_soft_hyphen'
929     local hyphen_char = 0x2D
930     local soft_hyphen_char = 0xAD

```

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

```

931     local softhyphen_fonts = setmetatable({}, {__index = function(t, fid)
932         local fdir = identifiers[fid]
933         local format = fdir and fdir.format
934         local result = (format == 'opentype' or format == 'truetype')
935         local characters = fdir and fdir.characters
936         result = result and (characters and characters[soft_hyphen_char]) ~= nil
937         t[fid] = result
938         return result
939     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.

```

940     local function process_softhyphen_pre(head, _context, _dir)
941         if softhyphenbool.mode ~= truebool.mode then return true end
942         for disc, sub in node.traverse_id(DISC, head) do
943             if sub == explicit_disc or sub == regular_disc then
944                 for n, _ch, _f in node.traverse_char(disc.pre) do
945                     local props = properties[n]
946                     if not props then
947                         props = {}
948                         properties[n] = props
949                     end
950                     props[is_soft_hyphen_prop] = true
951                 end
952             end
953         end

```

```

954     return true
955 end
956

```

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

```

957 local function process_softhyphen_post(head, _context, _dir)
958   if softhyphenbool.mode ~= truebool.mode then return true end
959   for disc, sub in node.traverse_id(DISC, head) do
960     for n, ch, fid in node.traverse_glyph(disc.pre) do
961       local props = properties[n]
962       if softhyphen_fonts[fid] and ch == hyphen_char and props and props[is_soft_hyphen_prop]
963         if nodegetattribute(n,softhyphenattribute) then
964           n.char = soft_hyphen_char
965           props.glyph_info = nil
966         else
967           nodesetattribute(n,mctypeattributeid,-2147483647)
968           nodesetattribute(n,mccntattributeid,-2147483647)
969         end
970       end
971     end
972   end
973   return true
974 end
975
976 luatexbase.add_to_callback('pre_shaping_filter', process_softhyphen_pre, 'tagpdf.rewrite-
softhyphen')
977 luatexbase.add_to_callback('post_shaping_filter', process_softhyphen_post, 'tagpdf.rewrite-
softhyphen')
978 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

```

979 local function role_get_parent_child_rule (parent,child)
980   local state=
981   ltx.__tag.role.matrix[ltx.__tag.role.index[parent]]
982   and ltx.__tag.role.matrix[ltx.__tag.role.index[parent]][ltx.__tag.role.index[child]] or 0
983   return state
984 end
985 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.

```

986 function check_update_stashed (struct,loglevel,loop)

```

```

987 loop = (loop or 0) + 1
988 if loop > 10 then
989   __tag_log ('Warning: Too deeply nested stashed structures',0)
990   return
991 end
992 __tag_log ('updating parentrole for stashed structure '..struct,loglevel)
993 local parent = ltx.__tag.tables['g__tag_struct_ '..struct..'__prop']['parentnum']
994 if parent then
995   local ptag =
996     string.match(ltx.__tag.tables['g__tag_struct_ '..parent..'__prop']['parentrole'], "{(.-)}{(.-)}{(.-)}")
997   if ptag == 'STASHED' then
998     -- look at the parent and update it first
999     check_update_stashed (parent,loglevel,loop)
1000   end
1001   -- now copy the parent role from the parent
1002   ltx.__tag.tables['g__tag_struct_ '..struct..'__prop']['parentrole']
1003   =
1004   ltx.__tag.tables['g__tag_struct_ '..parent..'__prop']['parentrole']
1005   __tag_log
1006   ('new parentrole: ' .. ltx.__tag.tables['g__tag_struct_ '..struct..'__prop']['parentrole'],0)
1007 else
1008   __tag_log ('Warning: structure '..struct..' has no parent.',0)
1009 end
1010 end
1011
1012 function check_parent_child_rules (loglevel)
1013   texio.write_nl('\n')
1014   __tag_log ('checking parent-child rules ...' ,0)
1015   for i=2,ltx.tag.get_struct_counter() do
1016     local t,tNS=
1017       string.match(ltx.__tag.tables['g__tag_struct_ '..i..'__prop']['tag'], "{(.-)}{(.-)}{(.-)}")
1018     local r,rNS=
1019       string.match(ltx.__tag.tables['g__tag_struct_ '..i..'__prop']['rolemap'], "{(.-)}{(.-)}{(.-)}")
1020     local p,pNS=
1021       string.match(ltx.__tag.tables['g__tag_struct_ '..i..'__prop']['parentrole'], "{(.-)}{(.-)}{(.-)}")
1022     local parent=ltx.__tag.tables['g__tag_struct_ '..i..'__prop']['parentnum']
1023     if parent then
1024       __tag_log (i..'': '.. t..'': '..tNS,loglevel)
1025       __tag_log (i..'': '.. r..'': '..rNS,loglevel)
1026       __tag_log (i..'': '.. p..'': '..pNS,loglevel)
1027       __tag_log ('parent of ' ..i..'': '.. parent,loglevel )
1028       if p == 'STASHED' then
1029         check_update_stashed (i,loglevel,0)
1030         p,pNS=
1031         string.match(ltx.__tag.tables['g__tag_struct_ '..i..'__prop']['parentrole'], "{(.-)}{(.-)}{(.-)}")
1032       end
1033       local pt,ptNS=
1034       string.match(ltx.__tag.tables['g__tag_struct_ '..parent..'__prop']['tag'], "{(.-)}{(.-)}{(.-)}")

```



```

1035     local pr,prNS=
1036     string.match(ltx.__tag.tables['g__tag_struct'..'parent..'__prop']['rolemap'], "{(.-
    )}{(.-)}")
1037     local pp,ppNS=
1038     string.match(ltx.__tag.tables['g__tag_struct'..'parent..'__prop']['parentrole'], "{(.-
    )}{(.-)}")
1039     if pp == 'STASHED' then
1040         check_update_stashed (parent,loglevel,0)
1041         pp,ppNS=
1042         string.match(ltx.__tag.tables['g__tag_struct'..'parent..'__prop']['parentrole'], "{(.-
    )}{(.-)}")
1043     end
1044     __tag_log (parent..'': '.. pt..'': '..ptNS,loglevel)
1045     __tag_log (parent..'': '.. pr..'': '..prNS,loglevel)
1046     __tag_log (parent..'': '.. pp..'': '..ppNS,loglevel)
1047     -- now check the rule.
1048     -- at first rolemap of child against rolemap of parent.
1049     local state=ltx.__tag.func.role_get_parent_child_rule (pr,r)
1050     __tag_log ('rule of '..pr..' -> '..r..' is '..state,loglevel)
1051     -- if the state is 7 we check against parentrole of the parent
1052     if state == 7 then
1053         state=ltx.__tag.func.role_get_parent_child_rule (pp,r)
1054         __tag_log ('Parent-Child relation '..pp..' -> '..r..' is '..state,loglevel)
1055     end
1056     if state == 0 then
1057         __tag_log
1058         ('Warning: Parent-Child relation '
1059         '..ptNS..'': '..pt..' -> '..tNS..'': '..t..' is unknown',0)
1060         __tag_log
1061         ('Structure ' ..parent..' -> '..i,0)
1062     end
1063     if state == -1 then
1064         __tag_log
1065         ('Warning: Parent-Child relation '
1066         '..ptNS..'': '..pt..' -> '..tNS..'': '..t..' is not allowed',0)
1067         __tag_log
1068         ('Structure ' ..parent..' -> '..i,0)
1069     end
1070     -- check also for MC
1071     state =ltx.__tag.func.role_get_parent_child_rule ( r ,'MC')
1072     local curtag=r
1073     if state == 7 then
1074         state =ltx.__tag.func.role_get_parent_child_rule ( p ,'MC')
1075         local curtag=p
1076     end
1077     if state == -1 then
1078         if ltx.__tag.struct[i] and NEXT(ltx.__tag.struct[i]) then
1079             __tag_log
1080             ('Warning: Real content (MC) is not allowed in ' ..curtag,0)
1081         end
1082     end
1083     __tag_log('=====',loglevel)
1084 end
1085 end -- end for

```

```

1086 end
1087
1088 ltx.__tag.func.check_parent_child_rules=check_parent_child_rules
1089
(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.

```

1090 local linkbincnt = 0
1091
1092 function ltx.__tag.func.linkbin()
1093   return linkbincnt
1094 end
1095
1096 if luatexbase.callbacktypes['linksplit'] then
1097   luatexbase.add_to_callback('linksplit', function(start_link, position)
1098     if start_link == nil then return end
1099     local structnum =
1100       node.get_attribute(start_link,luatexbase.attributes.g__tag_structnum_attr)
1101     if structnum and structnum > -1 then
1102       local s = ltx.__tag.tables['g__tag_struct_'.structnum..'_prop']['rolemap']
1103       if s and (string.find(s,'Link') or string.find(s,'Reference')) then
1104         local struct_insert_annot_shipout = token.create'__tag_struct_insert_annot_shipout:'
1105         local parentnum = tex.count['c@g__tag_parenttree_obj_int']
1106         start_link.link_attr =
1107           start_link.link_attr ..
1108           ' /LTEX_position /' .. position ..
1109           '/StructParent ' .. parentnum
1110         tex.sprint(catlatex,struct_insert_annot_shipout,'{'..
1111           structnum..''}{'..
1112           start_link.objnum..' 0 R'}{'..
1113           parentnum ..'}')
1114         -- the counter must be set explicitly as struct_insert_annot_shipout doesn't do it!
1115         tex.setcount('global','c@g__tag_parenttree_obj_int',parentnum +1)
1116         __tag_log(position .. " link part has object id " .. start_link.objnum .. " and str
1117       else
1118         local linkbin = token.get_macro("c__tag_struct_link_bin_tl")
1119         if linkbin then
1120           local struct_insert_annot_shipout = token.create'__tag_struct_insert_annot_shipout:'
1121           local parentnum = tex.count['c@g__tag_parenttree_obj_int']
1122           start_link.link_attr = string.gsub (start_link.link_attr, "/Contents%s+<%w+>", "")
1123           start_link.link_attr =
1124             start_link.link_attr ..
1125             ' /LTEX_position /' .. position ..
1126             '/StructParent ' .. parentnum .. '/Contents (artifact)'
1127           tex.sprint(catlatex,struct_insert_annot_shipout,'{'..
1128             linkbin..''}{'..
1129             start_link.objnum..' 0 R'}{'..
1130             parentnum ..'}')
1131           tex.setcount('global','c@g__tag_parenttree_obj_int',parentnum +1)

```

```

1132         linkbincnt=linkbincnt+1
1133         if linkbincnt == 1 then
1134             __tag_log('Info: some Link annotation(s) are not in Link or Reference structure el
1135             __tag_log('        moved into a container at the end of the structure tree\n',0)
1136         end
1137     else
1138         __tag_log('Warning: Link not in Link or Reference structure element',0)
1139         __tag_log('OBJR not created',0)
1140         __tag_log('',0)
1141     end
1142 end
1143 end
1144 end, 'tagpdf')
1145 end
1146 </lua>

```

Ulrike Fischer  
Version 1.0b, released 2026-04-24

## Part X

# The tagpdf-roles module

## Tags, roles and namespace code

---

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

---

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

The key-value list knows the following keys:

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

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

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

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

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

---

```
\tag_check_child:nnTF \tag_check_child:nnTF {\tag} {\namespace} {\true code} {\false code}
```

---

This checks if the tag `\tag` from the name space `\namespace` can be used at the current position. In tagpdf-base it is always true.

```
1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-roles-code} {2026-04-24} {1.0b}
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.

`\prop_new_linked:N \g__tag_role_tags_NS_prop`  
(End of definition for `\g__tag_role_tags_NS_prop`.)

<code>\g__tag_role_tags_class_prop</code>	<p>With pdf 2.0 we store the class in the NS dependent props. With pdf &lt;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.</p> <pre> <sup>8</sup> \prop_new:N \g__tag_role_tags_class_prop (End of definition for \g__tag_role_tags_class_prop.) </pre>
<code>\g__tag_role_NS_prop</code>	<p>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.</p> <p><b>mathml</b> <a href="http://www.w3.org/1998/Math/MathML">http://www.w3.org/1998/Math/MathML</a></p> <p><b>pdf2</b> <a href="http://iso.org/pdf2/ssn">http://iso.org/pdf2/ssn</a></p> <p><b>pdf</b> <a href="http://iso.org/pdf/ssn">http://iso.org/pdf/ssn</a> (default)</p> <p><b>user</b> <code>\c__tag_role_userNS_id_str</code> (random id, for user tags)</p> <p><b>latex</b> <a href="https://www.latex-project.org/ns/dft">https://www.latex-project.org/ns/dft</a></p> <p><b>latex-book</b> <a href="https://www.latex-project.org/ns/book">https://www.latex-project.org/ns/book</a></p> <p>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.</p> <pre> <sup>9</sup> \prop_new:N \g__tag_role_NS_prop (End of definition for \g__tag_role_NS_prop.) </pre>
<code>\g__tag_role_index_prop</code>	<p>This prop contains the standard tags (pdf in pdf&lt;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.</p> <pre> <sup>10</sup> \prop_new:N \g__tag_role_index_prop (End of definition for \g__tag_role_index_prop.) </pre>
<code>\l__tag_role_debug_prop</code>	<p>This variable is used to pass more infos to debug messages.</p> <pre> <sup>11</sup> \prop_new:N \l__tag_role_debug_prop (End of definition for \l__tag_role_debug_prop.) </pre> <p>We need also a bunch of temporary variables.</p> <pre> <sup>12</sup> \tl_new:N \l__tag_role_tag_tmpa_tl <sup>13</sup> \tl_new:N \l__tag_role_tag_namespace_tmpa_tl <sup>14</sup> \tl_new:N \l__tag_role_tag_namespace_tmpb_tl <sup>15</sup> \tl_new:N \l__tag_role_role_tmpa_tl <sup>16</sup> \tl_new:N \l__tag_role_role_namespace_tmpa_tl <sup>17</sup> \seq_new:N \l__tag_role_tmpa_seq (End of definition for \l__tag_role_tag_tmpa_tl and others.) </pre>

## 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:nNt {\l__tag_loglevel_int} > { 0 }
133         {
134             \msg_info:nnn { tag }{new-tag}{#1}
135         }
136     }

```

now the addition

```

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

```

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

```

142     \tl_if_empty:nF { #2 }
143     {
144         \prop_get:NnNTF \g__tag_role_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 {\l__tag_loglevel_int} > { 0 }
 171 {
 172 \msg_info:nnn { tag }{new-tag}{#1}
 173 }
 174 \prop_if_exist:cTF
 175 { g__tag_role_NS_#4_class_prop }
 176 {
 177 \prop_get:cnN { g__tag_role_NS_#4_class_prop } {#3}\l__tag_tmpa_tl
 178 \quark_if_no_value:NT \l__tag_tmpa_tl
 179 {
 180 \tl_set:Nn\l__tag_tmpa_tl{--UNKNOWN--}
 181 }
 182 }
 183 { \tl_set:Nn\l__tag_tmpa_tl{--UNKNOWN--} }
 184 \_tag_role_alloctag:nno {#1}{#2}{ \l__tag_tmpa_tl }`

Do not remap standard tags. TODO add warning?

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

\\_\_tag\_role\_get:nnNN

```

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

```

## 1.4 Helper command to read the data from files

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

`__tag_role_read_namespace_line:nw`

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

```

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

```

```

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

```

(End of definition for \\_\_tag\_role\_read\_namespace\_line:nw.)

\\_\_tag\_role\_read\_namespace:nn This command reads a namespace file in the format tagpdf-ns-XX.def

```

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

```

(End of definition for \\_\_tag\_role\_read\_namespace:nn.)

\\_\_tag\_role\_read\_namespace:n This command reads the default namespace file.

```

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

```

(End of definition for \\_\_tag\_role\_read\_namespace:n.)

## 1.5 Reading the default data

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

```

341 \__tag_role_read_namespace:n {pdf}

```

```

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

```

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

```

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

```

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

```

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

```

## 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 emptyness?
800         \__tag_role_add_tag:oo
801         { \l__tag_role_tag_tmpa_tl }
802         { \l__tag_role_role_tmpa_tl }
803     }
804     {
805         \__tag_role_add_tag:oooo
806         { \l__tag_role_tag_tmpa_tl }
807         { \l__tag_role_tag_namespace_tmpa_tl }
808         { \l__tag_role_role_tmpa_tl }
809         { \l__tag_role_role_namespace_tmpa_tl }
810     }
811 }
812 ,role/map-tags .choice:
813 ,role/map-tags/false .code:n = { \socket_assign_plug:nn { tag/struct/tag } {latex-
tags} }
814 ,role/map-tags/pdf .code:n = { \socket_assign_plug:nn { tag/struct/tag } {pdf-
tags} }

815 ,role/user-NS .code:n =
816 {
817     \pdf_version_compare:NnF < {2.0}
818     {
819         \pdf_string_from_unicode:nnN{utf8/string}{https://www.latex-project.org/ns/local/#1}
820         \tl_if_empty:NF \l__tag_tmpa_str
821         {
822             \pdfdict_gput:nne
823             {g__tag_role/Namespace_user_dict}
824             {NS}
825             {\l__tag_tmpa_str}
826         }
827     }
828 }

```

deprecated names

```

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

```

831 }  
832 </package>  
*(End of definition for tag (rolemap-key) and others. These functions are documented on page 180.)*

Ulrike Fischer  
Version 1.0b, released 2026-04-24

## Part XI

# The tagpdf-space module

## Code related to real space chars

---

activate/space (setup-key)  
interwordspace (deprecated)

---

This key allows to activate/deactivate the real space chars if the engine supports it. The allowed values are `true`, `on`, `false`, `off`. The old name of the key `interwordspace` is still supported but deprecated.

---

show-spaces (deprecated)

---

This key is deprecated. Use `debug/show=spaces` instead. This key works only with `luatex` and shows with small red bars where spaces have been inserted. This is only for debugging and is not completely reliable (and change affect other literals and tagging), so it should be used with care.

```
1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-space-code} {2026-04-24} {1.0b}
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=spaces0ff},
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/spac
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/spaces0ff .code:n =
69     { \lua_now:e{!tx.__tag.trace.showspace=nil} },
70   }
71 }

```

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

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

```

72 \sys_if_engine_luatex:T

```

```

73 {
74   \cs_new_protected:Nn \__tag_fakespace:
75   {
76     \group_begin:
77     \lua_now:e{\ltx.__tag.func.fakespace()}
78     \skip_horizontal:n{\c_zero_skip}
79     \group_end:
80   }
81 }

```

We need also a command to interrupt the insertion of real space chars in places where we want to insert manually special spaces. In pdftex this can be done with `\pdfinterwordspaceoff` and `\pdfinterwordspaceon`. These commands insert what-sits and this mean they act globally. In luatex a attribute is used to this effect, for consistency this is also set globally.

The off command sets the attributes in luatex.

```

\tag_spacechar_on: 82 \cs_new_protected:Npn \tag_spacechar_off: {}
\tag_spacechar_off: 83 \cs_new_protected:Npn \tag_spacechar_on: {}
84
85 \sys_if_engine_luatex:T
86 {
87   \cs_set_protected:Npn \tag_spacechar_off:
88   {
89     \lua_now:e
90     {
91       tex.setattribute
92       (
93         "global",
94         luatexbase.attributes.g__tag_interwordspaceOff_attr,
95         1
96       )
97     }
98   }
99   \cs_set_protected:Npn \tag_spacechar_on:
100   {
101     \lua_now:e
102     {
103       tex.setattribute
104       (
105         "global",
106         luatexbase.attributes.g__tag_interwordspaceOff_attr,
107         -2147483647
108       )
109     }
110   }
111 }
112 \sys_if_engine_pdftex:T
113 {
114   \sys_if_output_pdf:T
115   {
116     \cs_set_protected:Npn \tag_spacechar_off:
117     {
118       \pdfinterwordspaceoff
119     }

```

```

120     \cs_set_protected:Npn \tag_spacechar_on:
121     {
122         \pdfinterwordspaceon
123     }
124 }
125
126 \end{package}

```

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

# Index

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

Symbols	
<code>\\</code> .....	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, 565, 628, 636
<code>\_</code> .....	410, 421
A	
<code>activate\_ (setup-key)</code> .....	42, 274
<code>activate-all (deprecated) (key)</code> .....	1
<code>activate-mc (deprecated) (key)</code> .....	1
<code>activate-struct (deprecated) (key)</code> ...	1
<code>activate-tree (deprecated) (key)</code> .....	1
<code>activate/all (key)</code> .....	1, 242
<code>activate/collect-link-artifacts (key)</code> .....	1, 340
<code>activate/mc (key)</code> .....	1, 242
<code>activate/socket\_ (setup-key)</code> .....	274
<code>activate/softhyphen (key)</code> .....	1, 276
<code>activate/space\_ (setup-key)</code> .....	203
<code>activate/spaces (key)</code> .....	1
<code>activate/spaces\_ (setup-key)</code> .....	6
<code>activate/struct (key)</code> .....	1, 242
<code>activate/struct-dest (key)</code> .....	1, 242
<code>activate/tagunmarked (key)</code> .....	1, 273
<code>activate/tree (key)</code> .....	1, 242
<code>actualtext (key)</code> .....	1, 724
<code>actualtext\_ (mc-key)</code> .....	80, 238, 384
<code>add-new-tag\_ (deprecated)</code> .....	752
<code>add-new-tag\_ (setup-key)</code> .....	180
<code>\AddToHook</code> ..	13, 16, 31, 34, 44, 57, 58, 228, 238, 240, 292, 425, 427, 428, 430
<code>\AddToHookNext</code> .....	623
<code>AF (key)</code> .....	1, 937
<code>AFinline (key)</code> .....	1, 937
<code>AFinline-o (key)</code> .....	1, 937
<code>AFref (key)</code> .....	1, 937
<code>alt (key)</code> .....	1, 724
<code>alt\_ (mc-key)</code> .....	80, 238, 384
<code>artifact\_ (mc-key)</code> .....	80, 238, 384
artifact-bool internal commands:	
<code>__artifact-bool</code> .....	182
artifact-type internal commands:	
<code>__artifact-type</code> .....	182
<code>\AssignTaggingSocketPlug</code> .....	477, 484, 586, 587, 644, 653
<code>\AtBeginDocument</code> .....	570
<code>attr-unknown</code> .....	23, 84
<code>attribute (key)</code> .....	1, 1557
<code>attribute-class (key)</code> .....	1, 1523
B	
benchmark commands:	
<code>\benchmark_tic:</code> .....	645, 647
<code>\benchmark_toc:</code> .....	648
bool commands:	
<code>\bool_gset_eq:NN</code> .....	494, 509, 521, 539, 594, 608
<code>\bool_gset_false:N</code> .....	62, 221, 277, 278, 372, 495, 522, 595
<code>\bool_gset_true:N</code> .....	36, 57, 88, 96, 175, 281, 296, 311, 313, 326, 865
<code>\bool_if:NTF</code> ....	9, 13, 18, 31, 40, 40, 64, 68, 69, 74, 80, 85, 100, 124, 135, 196, 203, 208, 228, 248, 264, 265, 303, 319, 323, 339, 351, 360, 369, 389, 407, 418, 468, 489, 504, 516, 534, 589, 603, 1188, 1220, 1251
<code>\bool_if:nTF</code> .....	515
<code>\bool_lazy_all:nTF</code> .....	258
<code>\bool_lazy_and:nnTF</code> .....	294, 300, 309, 310, 355, 373, 548, 619, 656, 697, 737, 868
<code>\bool_new:N</code> .....	7, 16, 20, 21, 35, 73, 83, 84, 85, 86, 87, 87, 89, 91, 93, 94, 95, 255, 312, 313, 485, 864
<code>\bool_set_false:N</code> .....	17, 165, 166, 167, 176, 189, 190, 191, 222, 344, 379, 459, 488, 515, 588
<code>\bool_set_true:N</code>	16, 90, 92, 175, 176, 177, 200, 201, 202, 256, 346, 378, 458
box commands:	
<code>\box_dp:N</code> .....	180, 184
<code>\box_ht:N</code> .....	170
<code>\box_new:N</code> .....	78, 79
<code>\box_set_dp:Nn</code> .....	178, 180
<code>\box_set_eq:NN</code> .....	193
<code>\box_set_ht:Nn</code> .....	177, 179
<code>\box_use_drop:N</code> .....	182, 186
<code>\boxmaxdepth</code> .....	93, 181
C	
c@g internal commands:	
<code>\c@g__tag_MCID_abs_int</code> .....	11, 15, 28, 37, 50, 57, 60, 68, 74, 134, 138, 178, 242, 245, 288, 295, 346

\c@g__tag_parenttree_obj_int	155, 502	78, 79, 80, 81, 82, 83, 84, 85, 86, 93,
\c@g__tag_struct_abs_int	6, 20, 40,	95, 96, 107, 108, 109, 117, 120, 122,
	58, 91, 114, 115, 118, 125, 128, 149,	126, 137, 142, 147, 153, 163, 163,
	166, 226, 248, 373, 552, 729, 742,	167, 169, 171, 172, 177, 189, 197,
	787, 799, 813, 829, 844, 852, 921,	211, 212, 213, 214, 215, 216, 221,
	932, 951, 954, 959, 995, 997, 1002,	229, 233, 246, 250, 255, 259, 260,
	1014, 1016, 1021, 1112, 1123, 1124,	263, 266, 266, 277, 281, 282, 285,
	1125, 1126, 1127, 1129, 1131, 1137,	286, 292, 295, 307, 317, 318, 318,
	1142, 1149, 1152, 1162, 1172, 1176,	322, 328, 329, 333, 335, 337, 337,
	1191, 1204, 1214, 1227, 1230, 1245,	341, 344, 345, 348, 349, 358, 364,
	1246, 1248, 1259, 1550, 1553, 1601	377, 393, 396, 397, 404, 404, 411,
catalog-supplemental-file (key) ...	1092	415, 417, 423, 430, 437, 442, 461,
cctab commands:		464, 466, 467, 468, 469, 486, 500,
\c_document_cctab	49, 54, 75, 144, 510	503, 505, 513, 529, 535, 541, 542,
\chapter .....	191, 357, 375	549, 556, 556, 576, 580, 584, 584,
check commands:		591, 592, 599, 600, 606, 636, 640,
check_parent_child_rules .....	986	641, 642, 643, 857, 865, 878, 891,
check_update_stashed .....	986	906, 939, 967, 1112, 1113, 1114,
clist commands:		1313, 1354, 1407, 1420, 1440, 1444,
\clist_const:Nn .....	80, 81	1448, 1452, 1456, 1462, 1481, 1505
\clist_if_empty:NTF .....	1562	\cs_set:Nn .....
\clist_map_inline:nn ...	107, 421, 918	554, 555, 616, 617
\clist_new:N .....	76	\cs_set:Npn .....
\clist_set:Nn .....	1527, 1561	47, 52, 78, 98
color commands:		\cs_set_eq:NN .....
\color_select:n .....	410, 421	14, 20,
cs commands:		66, 80, 81, 82, 140, 141, 142, 143,
\cs:w .....	1429, 1433	144, 145, 146, 147, 148, 149, 182,
\cs_end: .....	1429, 1433	183, 194, 208, 217, 218, 225, 230,
\cs_generate_variant:Nn .....	44,	235, 236, 237, 238, 366, 367, 368,
	79, 99, 100, 101, 102, 103, 103,	369, 547, 548, 549, 550, 556, 557,
	104, 105, 106, 107, 107, 107, 108,	561, 562, 563, 564, 618, 619, 647, 648
	116, 117, 118, 126, 134, 151, 152,	\cs_set_protected:Nn .....
	153, 153, 154, 155, 156, 157, 163,	.. 169, 216, 265, 359, 365, 1269, 1270
	164, 168, 181, 196, 216, 226, 249,	\cs_set_protected:Npn .....
	253, 265, 265, 276, 281, 306, 316,	9,
	343, 589, 635, 683, 717, 938, 966,	15, 16, 22, 29, 35, 38, 40, 48, 49, 52,
	987, 1398, 1405, 1417, 1461, 1490, 1511	58, 62, 63, 65, 67, 71, 72, 83, 83, 87,
\cs_gset_eq:NN .....	440	97, 99, 102, 116, 120, 143, 161, 170,
\cs_if_exist:NTF .....	249, 625, 645	184, 195, 220, 228, 230, 240, 251,
\cs_if_exist_p:N ..	357, 360, 375, 378	267, 285, 299, 305, 347, 351, 355,
\cs_if_exist_use:NTF .....	402, 1411	359, 1116, 1117, 1307, 1315, 1356, 1409
\cs_if_free:NTF .....	48	\cs_to_str:N .....
\cs_new:Nn .....		.. 12, 18, 25, 32, 38, 43, 61, 62, 68, 69
	83, 109, 131, 136, 293, 411, 412, 413	\cs_undefine:N .....
\cs_new:Npn .....	9, 15, 25, 27,	57
	107, 119, 151, 156, 217, 229, 235,	
	237, 371, 530, 538, 544, 550, 1401, 1491	
\cs_new_eq:NN .....	39	
\cs_new_protected:Nn .....		
	74, 127, 167, 296, 414, 418	
\cs_new_protected:Npn .....		
	13, 17, 20, 22, 23, 30,	
	31, 36, 42, 43, 45, 60, 61, 63, 65, 67,	

  

D	
debug/log (key) .....	1, 260
debug/show (key) .....	259
debug/structures_ (show-key) ...	43, 243
debug/uncompress (key) .....	260
\DebugSocketsOn .....	45
\DeclareOption .....	37, 38
dim commands:	
\c_max_dim .....	169, 194
\c_zero_dim .....	177, 178, 179
\directlua .....	231
\documentclass .....	16



\DocumentMetadata ..... 15

## E

E (key) ..... 1, 724, 914

\endinput ..... 22

\ERRORusetaggingsocket ..... 95, 110

exclude-header-footer<sub>□</sub>(deprecated) 542

exp commands:

\exp\_args:Ne ..... 122, 532

\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, 208, 209, 238,

239, 596, 599, 607, 610, 614, 618, 1296

\exp\_not:n ..... 187, 206

## F

file commands:

\file\_if\_exist:nTF ..... 322

\file\_input:n ..... 356

firstkid (key) ..... 1, 724

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, 944, 1036, 1044, 1079, 1096, 1122

\group\_end: ..... 74, 79, 213,

350, 962, 1040, 1050, 1089, 1107, 1265

## H

\halign ..... 45

hbox commands:

\hbox\_set:Nn ..... 171, 172

hook commands:

\hook\_gput\_code:nnn . 7, 11, 33, 57,

66, 80, 156, 239, 277, 278, 353, 371,

387, 391, 668, 675, 682, 689, 696,

703, 709, 716, 722, 729, 737, 750,

761, 774, 785, 798, 809, 822, 832, 845

\hook\_new:n ..... 348

\hook\_use:n ..... 353

## I

\IfPDFManagementActiveF ..... 6

\ignorespaces ..... 42

int commands:

\int\_abs:n ..... 156

\int\_case:nnTF ..... 88, 103, 331

\int\_compare:nNnTF ..... 22,

58, 70, 98, 116, 124, 125, 132, 137,

142, 157, 170, 173, 173, 231, 279,

337, 367, 386, 413, 416, 436, 444,

444, 445, 450, 450, 512, 528, 537,

544, 551, 558, 562, 578, 578, 585,

586, 593, 601, 608, 621, 734, 748, 1153

\int\_compare:nTF ..... 180,

484, 1543, 1545, 1547, 1571, 1597

\int\_compare\_p:nNn ..... 742

\int\_decr:N ..... 172, 197

\int\_eval:n ..... 118, 138, 166,

197, 396, 629, 637, 739, 744, 747,

959, 1002, 1021, 1124, 1125, 1126,

1127, 1245, 1246, 1248, 1259, 1553

\int\_gincr:N ... 178, 242, 288, 295,

331, 335, 339, 343, 349, 353, 357,

361, 502, 945, 1081, 1098, 1112, 1123

\int\_gset:Nn ..... 7, 82, 158

\int\_if\_zero:nTF ..... 172,

173, 197, 198, 625, 633

\int\_incr:N ..... 93, 164, 188, 442

\int\_new:N ..... 6, 77, 78,

82, 97, 155, 160, 315, 316, 317, 318, 937

\int\_rand:n .. 61, 62, 64, 66, 68, 70, 71

\int\_set:Nn .... 261, 264, 267, 268, 269

\int\_step\_inline:nn ..... 473, 479

\int\_step\_inline:nnn ..... 25, 91, 248

\int\_step\_inline:nnnn ..... 149,

174, 177, 200, 469, 475

\int\_to\_arabic:n ..... 156, 158

\int\_to\_Hex:n 61, 62, 64, 66, 68, 70, 71

\int\_use:N .. 11, 15, 20, 28, 37, 40,

49, 50, 54, 57, 58, 60, 68, 74, 75, 89,

104, 125, 132, 134, 144, 163, 180,

187, 206, 226, 234, 241, 245, 277,

279, 346, 373, 410, 421, 441, 442,

450, 451, 510, 517, 552, 729, 787,

799, 813, 829, 844, 852, 921, 932,

948, 951, 954, 995, 997, 1014, 1016,

1085, 1088, 1102, 1106, 1131, 1137,

1142, 1149, 1152, 1176, 1191, 1204,

1214, 1227, 1230, 1491, 1550, 1601

\int\_zero:N ..... 90, 105, 430

intarray commands:

\intarray\_gset:Nnn ..... 397,

406, 421, 425, 447

\intarray\_item:Nn ..... 449, 452, 554

`\intarray_new:Nn` ..... 390, 439  
`interwordspace_`(deprecated) .... 203, 6  
ior commands:  
`\ior_close:N` ..... 330, 467  
`\ior_map_inline:Nn` ..... 326, 438  
`\ior_open:Nn` ..... 324, 433, 436  
`\g_tmpa_ior` .....  
..... 324, 326, 330, 433, 436, 438, 467  
iow commands:  
`\iow_newline:` ..... 205, 303  
`\iow_term:n` 198, 200, 203, 209, 213,  
265, 355, 359, 363, 367, 371, 375, 379

## K

kernel internal commands:  
`\__kernel_pdffdict_name:n` ..... 45  
`\g__kernel_pdfmanagement_end_-`  
`run_code_tl` ..... 866  
keys commands:  
`\keys_define:nn` .....  
..... 8, 32, 34, 40, 52, 120, 132,  
182, 192, 195, 235, 238, 243, 244,  
280, 375, 384, 385, 391, 397, 462,  
542, 611, 724, 752, 760, 877, 914,  
988, 1053, 1075, 1092, 1512, 1523, 1557  
`\keys_set:nn` ..... 10,  
18, 18, 19, 117, 187, 190, 285, 318,  
321, 338, 342, 428, 872, 1086, 1147  
`\keys_set_known:nnnN` ..... 765

## L

`label (key)` ..... 1, 724  
`\label` ..... 12  
`label_`(mc-key) ..... 80, 238, 384  
`lang (key)` ..... 1, 724  
`lang_`(mc-key=) ..... 238  
`\llap` ..... 410  
`log (deprecated) (key)` ..... 260  
ltx. internal commands:  
`ltx.__tag.func.alloctag` ..... 314  
`ltx.__tag.func.check_parent_-`  
`child_rules` ..... 986  
`ltx.__tag.func.fakespace` ..... 493  
`ltx.__tag.func.fill_parent_tree_-`  
`line` ..... 873  
`ltx.__tag.func.get_num_from` .... 323  
`ltx.__tag.func.get_tag_from` .... 342  
`ltx.__tag.func.mark_page_-`  
`elements` ..... 699  
`ltx.__tag.func.mark_shipout` .... 856  
`ltx.__tag.func.markspaceoff` .... 564  
`ltx.__tag.func.markspaceon` .... 564  
`ltx.__tag.func.mc_insert_kids` .. 636  
`ltx.__tag.func.mc_num_of_kids` .. 372

`ltx.__tag.func.output_num_from` . 323  
`ltx.__tag.func.output_parenttree` 873  
`ltx.__tag.func.output_tag_from` . 342  
`ltx.__tag.func.role_get_parent_-`  
`child_rule` ..... 979  
`ltx.__tag.func.space_chars_-`  
`shipout` ..... 596  
`ltx.__tag.func.store_mc_data` ... 357  
`ltx.__tag.func.store_mc_in_page` 680  
`ltx.__tag.func.store_mc_kid` .... 366  
`ltx.__tag.func.store_mc_label` .. 362  
`ltx.__tag.func.store_struct_-`  
`mcabs` ..... 668  
`ltx.__tag.func.update_mc_-`  
`attributes` ..... 688  
`ltx.__tag.tables.role_tag_-`  
`attribute` ..... 312  
`ltx.__tag.trace.log` ..... 226  
`ltx.__tag.trace.show_all_mc_data` 283  
`ltx.__tag.trace.show_mc_data` ... 268  
`ltx.__tag.trace.show_prop` ..... 243  
`ltx.__tag.trace.show_seq` ..... 234  
`ltx.__tag.trace.show_struct_data` 289  
lua commands:  
`\lua_escape:n` ..... 32  
`\lua_now:n` ..... 8, 12, 15, 18,  
25, 26, 27, 32, 35, 38, 42, 43, 49, 50,  
54, 58, 59, 61, 62, 63, 67, 68, 69, 69,  
73, 77, 86, 87, 87, 89, 96, 101, 109,  
111, 120, 126, 133, 138, 141, 150,  
158, 162, 181, 189, 230, 237, 244,  
252, 268, 282, 284, 299, 303, 314,  
317, 327, 329, 399, 408, 497, 510, 661

## M

`\MakeLinkTarget` ..... 147  
`mathml (key)` ..... 1, 937  
`\maxdimen` ..... 192  
`mc-current` ..... 22, 16  
`mc-current_`(show-key) ..... 43, 132  
`mc-data_`(show-key) ..... 43, 120  
`mc-label-unknown` ..... 22, 9  
`mc-marks_`(show-key) ..... 43, 192  
`mc-nested` ..... 22, 6  
`mc-not-open` ..... 22, 13  
`mc-popped` ..... 22, 14  
`mc-pushed` ..... 22, 14  
`mc-tag-missing` ..... 22, 8  
`mc-used-twice` ..... 22, 12  
`\MessageBreak` ..... 10, 14, 15  
mode commands:  
`\mode_leave_vertical:` ..... 638  
msg commands:  
`\msg_error:nn` .... 325, 346, 475, 1159



\pdf\_version\_compare:NnTF .....  
         20, 81, 138, 154, 161, 227,  
         257, 324, 351, 431, 471, 590, 797, 817  
 \pdf\_version\_gset:n ..... 243  
 pdfannot commands:  
   \pdfannot\_dict\_put:nnn .....  
         100, 743, 767, 791, 815, 838  
   \pdfannot\_link\_ref\_last: .....  
         757, 781, 805, 829, 852  
 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 ..... 1037,  
         1038, 1045, 1046, 1047, 1080, 1097  
   \pdfdict\_use:n ..... 283, 332, 339  
 \pdffakespace ..... 43, 303  
 pdfffile commands:  
   \pdfffile\_embed\_file:nnn .....  
         108, 1082, 1099  
   \pdfffile\_embed\_stream:nnN . 938, 946  
   \pdfffile\_embed\_stream:nnn ..... 101  
 \pdfglyptounicode ..... 30  
 \pdfinterwordspaceoff ..... 205, 118  
 \pdfinterwordspaceon ..... 205, 34, 122  
 pdfmanagement commands:  
   \pdfmanagement\_add:nnn .....  
         52, 70, 71, 344, 346, 348, 393, 1103  
   \pdfmanagement\_remove:nn ..... 350  
 phoneme (key) ..... 724  
 prg commands:  
   \prg\_do\_nothing: 39, 82, 91, 106, 366,  
         367, 368, 369, 440, 561, 562, 563, 564  
   \prg\_generate\_conditional\_-  
     variant:Nnn ..... 98  
   \prg\_new\_conditional:Nnn ... 68, 226  
   \prg\_new\_conditional:Npnn .....  
         251, 275, 290, 298, 308, 507, 513, 524  
   \prg\_new\_eq\_conditional:NNn . 82, 233  
   \prg\_new\_protected\_conditional:Npnn  
         ..... 719  
   \prg\_replicate:nn ..... 155  
   \prg\_return\_false: .....  
         78, 230, 252, 270, 281,  
         284, 294, 305, 315, 510, 522, 528, 749  
   \prg\_return\_true: 79, 229, 267, 280,  
         293, 302, 312, 511, 521, 527, 719, 750  
   \prg\_set\_conditional:Npnn ..... 256  
   \prg\_set\_protected\_conditional:Npnn  
         ..... 721  
 process commands:  
   process\_softhyphen\_pre\_uuuuprocess\_-  
     softhyphen\_post ..... 926  
 \ProcessOptions ..... 39  
 prop commands:  
   \prop\_clear:N ..... 176  
   \prop\_count:N ..... 203  
   \prop\_gclear:N ..... 883  
   \prop\_get:NnN .. 127, 144, 145, 177,  
         198, 214, 268, 299, 456, 468, 470,  
         483, 484, 485, 546, 547, 587, 588, 895  
   \prop\_get:NnNTF .. 44, 96, 130, 137,  
         144, 158, 183, 183, 203, 205, 236,  
         295, 320, 330, 350, 365, 384, 395,  
         419, 431, 434, 514, 564, 599, 604,  
         609, 614, 650, 677, 687, 703, 754,  
         780, 785, 867, 880, 893, 1198, 1297,  
         1363, 1423, 1465, 1535, 1575, 1579  
   \prop\_gput:Nnn .....  
         24, 26, 27, 28, 31, 56, 88, 90,  
         91, 97, 98, 99, 100, 101, 101, 102,  
         103, 110, 112, 113, 119, 121, 122,  
         143, 145, 269, 272, 286, 291, 383,  
         436, 438, 439, 448, 469, 475, 481,  
         488, 490, 491, 728, 884, 886, 1247,  
         1258, 1334, 1379, 1507, 1539, 1586  
   \prop\_gremove:Nn ..... 137, 149, 887  
   \prop\_gset\_eq:NN ..... 148, 1244  
   \prop\_gset\_from\_keyval:Nn ..... 857  
   \prop\_if\_exist:NnTF ..... 174,  
         209, 234, 320, 432, 648, 675, 1319, 1360  
   \prop\_if\_exist\_p:N ..... 739  
   \prop\_item:Nn 41, 99, 102, 104, 109,  
         115, 147, 246, 535, 1255, 1584, 1591  
   \prop\_map\_function:NN ..... 254  
   \prop\_map\_inline:Nn .... 267, 272,  
         293, 326, 363, 381, 400, 486, 495, 870  
   \prop\_map\_tokens:Nn ..... 344  
   \prop\_new:N .... 8, 9, 10, 11, 11, 25,  
         33, 73, 140, 146, 856, 1125, 1500, 1503  
   \prop\_new\_linked:N .....  
         7, 17, 86, 91, 93, 141, 1501  
   \prop\_put:Nnn ..... 103, 188  
   \prop\_show:N .....  
         67, 95, 149, 1241, 1262, 1553, 1580  
 property commands:  
   \property\_new:nnnn .....  
         123, 126, 130, 133, 137  
   \property\_record:nn ..... 59, 112  
   \property\_ref:nn ..... 111, 117  
   \property\_ref:nnn .....  
         42, 116, 121, 181, 190,  
         223, 224, 351, 486, 489, 497, 1320, 1324  
 \providecommand ..... 310  
 \ProvidesExplFile ..... 3  
 \ProvidesExplPackage ..... 3, 3,  
         3, 3, 3, 3, 3, 3, 3, 7, 7, 20, 31, 1496

## Q

\quad ..... 222, 223  
 quark commands:  
   \q\_no\_value 606, 616, 662, 667, 689, 694  
   \quark\_if\_no\_value:NTF .....  
     132, 178, 199, 215, 269, 300, 593, 604  
   \quark\_if\_no\_value\_p:N .....  
     549, 550, 620, 621, 698, 699  
   \q\_stop ..... 259, 292, 328

## R

raw<sub>␣</sub>(mc-key) ..... 80, 238, 384  
 ref (key) ..... 1, 724, 914  
 \RemoveFromHook ..... 35, 341  
 \renewcommand ..... 458, 459  
 \RenewDocumentCommand ..... 8  
 \RequirePackage ... 40, 362, 365, 371, 374  
 \rlap ..... 421  
 role<sub>␣</sub>(rolemap-key) ..... 180, 752  
 role commands:  
   role\_get\_parent\_child\_rule .... 979  
 role-MC-child-forbidden ..... 104  
 role-missing ..... 23, 86  
 role-namespace<sub>␣</sub>(rolemap-key) . 180, 752  
 role-parent-child-check ..... 90  
 role-parent-child-forbidden ..... 111  
 role-parent-child-result ..... 23, 92  
 role-parent-child-unresolved ..... 164  
 role-remapping ..... 23, 213  
 role-struct-parent-child-forbidden . 94  
 role-tag ..... 23, 215  
 role-unknown ..... 23, 86  
 role-unknown-NS ..... 23, 86  
 role-unknown-tag ..... 23, 86  
 role/new-attribute<sub>␣</sub>(setup-key) 113, 1505  
 role/new-tag<sub>␣</sub>(setup-key) ..... 752  
 root-AF (key) ..... 1, 1053  
 root-supplemental-file (key) .... 1075

## S

\selectfont ..... 6  
 seq commands:  
   \seq\_clear:N ..... 321, 474  
   \seq\_const\_from\_clist:Nn .... 41, 54  
   \seq\_count:N ..... 22, 25, 58,  
     333, 446, 1543, 1545, 1547, 1571, 1597  
   \seq\_get:NN ..... 723, 736  
   \seq\_get:NTF 471, 610, 1155, 1285, 1293  
   \seq\_gpop:NN ..... 1277, 1278  
   \seq\_gpop:NTF ..... 106, 1279  
   \seq\_gpop\_left:NN ..... 309  
   \seq\_gpush:Nn .....  
     13, 16, 17, 89, 96, 1162, 1168, 1170  
   \seq\_gput\_left:Nn .. 42, 145, 275, 313

\seq\_gput\_right:Nn .....  
   37, 144, 146, 152, 238, 259, 298, 494  
 \seq\_gset\_eq:NN ..... 159, 221, 328  
 \seq\_if\_empty:NTF ..... 200, 440  
 \seq\_if\_in:NnTF ..... 292  
 \seq\_item:Nn .....  
   59, 116, 118, 125, 129, 136, 140,  
   146, 350, 357, 370, 517, 519, 526,  
   696, 697, 712, 713, 763, 764, 775, 776  
 \seq\_log:N 175, 199, 238, 420, 581, 596  
 \seq\_map\_function:NN ..... 263  
 \seq\_map\_indexed\_inline:Nn 446, 459  
 \seq\_map\_inline:Nn 289, 322, 1533, 1573  
 \seq\_new:N ..... 12,  
   14, 14, 15, 15, 16, 17, 18, 19, 21,  
   22, 24, 74, 75, 142, 147, 1127, 1504  
 \seq\_pop\_left:NN ..... 455, 457, 458  
 \seq\_put\_right:Nn ..... 323  
 \seq\_remove\_all:Nn ..... 326  
 \seq\_set\_eq:NN ..... 207, 208  
 \seq\_set\_from\_clist:NN ... 1528, 1564  
 \seq\_set\_from\_clist:Nn .....  
   87, 90, 196, 216, 443, 454  
 \seq\_set\_map\_e:NNn ..... 1529, 1565  
 \seq\_set\_split:Nnn ..... 51,  
   105, 689, 693, 705, 709, 756, 760, 774  
 \seq\_show:N .....  
   60, 148, 205, 206, 239, 324,  
   325, 327, 504, 1174, 1242, 1263, 1273  
 \seq\_use:Nn .....  
   50, 110, 111, 205, 222, 223, 385, 1544

### Setup keys:

activate-all (deprecated) ..... 1  
 activate-mc (deprecated) ..... 1  
 activate-struct (deprecated) .... 1  
 activate-tree (deprecated) ..... 1  
 activate/all ..... 1, 242  
 activate/collect-link-artifacts  
   ..... 1, 340  
 activate/mc ..... 1, 242  
 activate/softhyphen ..... 1, 276  
 activate/spaces ..... 1  
 activate/struct ..... 1, 242  
 activate/struct-dest ..... 1, 242  
 activate/tagunmarked ..... 1, 273  
 activate/tree ..... 1, 242  
 catalog-supplemental-file .... 1092  
 debug/log ..... 1, 260  
 debug/show ..... 259  
 debug/uncompress ..... 260  
 log (deprecated) ..... 260  
 no-struct-dest (deprecated) ..... 1  
 page/tabsorder ..... 1, 342  
 root-AF ..... 1, 1053

root-supplemental-file	<a href="#">1075</a>	AFinline-o	<a href="#">1, 937</a>
tabsorder (deprecated)	<a href="#">1, 342</a>	AFref	<a href="#">1, 937</a>
tagunmarked (deprecated)	<a href="#">1, 273</a>	alt	<a href="#">1, 724</a>
uncompress (deprecated)	<a href="#">260</a>	attribute	<a href="#">1, 1557</a>
shipout commands:		attribute-class	<a href="#">1, 1523</a>
\g_shipout_readonly_int		E	<a href="#">1, 724, 914</a>
	<a href="#">127, 132, 241, 396, 517</a>	firstkid	<a href="#">1, 724</a>
show-kids	<a href="#">23, 64</a>	label	<a href="#">1, 724</a>
show-spaces <sub>␣</sub> (deprecated)	<a href="#">203, 6</a>	lang	<a href="#">1, 724</a>
show-struct	<a href="#">23, 64</a>	mathml	<a href="#">1, 937</a>
\ShowTagging	<a href="#">20, 43, 114</a>	parent	<a href="#">1, 724</a>
skip commands:		phoneme	<a href="#">724</a>
\skip_horizontal:n	<a href="#">78</a>	ref	<a href="#">1, 724, 914</a>
\c_zero_skip	<a href="#">78</a>	stash	<a href="#">1, 724</a>
socket commands:		tag	<a href="#">1, 724</a>
\socket_assign_plug:nn	<a href="#">200, 204, 205, 209, 210, 723, 813, 814</a>	texsource	<a href="#">1, 937</a>
\socket_if_exist:nTF	<a href="#">631</a>	title	<a href="#">1, 724</a>
\socket_new:nn	<a href="#">183, 184, 684</a>	title-o	<a href="#">1, 724</a>
\socket_new_plug:nnn	<a href="#">185, 685, 701</a>	\SuspendTagging	<a href="#">45</a>
\socket_use:n	<a href="#">28, 65</a>	sys commands:	
\socket_use:nn	<a href="#">70, 205, 341, 777, 1223, 1341, 1386</a>	\c_sys_backend_str	<a href="#">46</a>
\socket_use:nnn	<a href="#">75</a>	\c_sys_engine_str	<a href="#">12, 14</a>
\socket_use:nw	<a href="#">86</a>	\sys_if_engine luatex:TF	<a href="#">23, 36, 50, 72, 83, 85, 105, 187, 223, 282, 297, 312, 327, 345, 354, 391, 493, 501</a>
\socket_use_expandable:n	<a href="#">81</a>	\sys_if_engine luatex:p:	<a href="#">657</a>
\socket_use_expandable:nw	<a href="#">101</a>	\sys_if_engine pdftex:TF	<a href="#">26, 112</a>
stash (key)	<a href="#">1, 724</a>	\sys_if_output_pdf:TF	<a href="#">11, 28, 114</a>
stash <sub>␣</sub> (mc-key)	<a href="#">81, 182</a>	sys-no-interwordspace	<a href="#">23, 228</a>
str commands:			
\str_case:nnTF	<a href="#">46, 645, 1181</a>		
\str_const:Nn	<a href="#">59</a>		
\str_if_eq:nnTF	<a href="#">117, 127, 526, 612, 657</a>		
\str_if_eq_p:nn	<a href="#">310, 517, 519</a>		
\str_new:N	<a href="#">72</a>		
\str_set_convert:Nnnn	<a href="#">106, 261, 296, 398, 415, 781, 793, 807, 823, 838, 926</a>		
\str_use:N	<a href="#">272, 309</a>		
\c_tilde_str	<a href="#">57, 59</a>		
\string	<a href="#">15, 16</a>		
struct-faulty-nesting	<a href="#">23, 32</a>		
struct-label-unknown	<a href="#">23, 38</a>		
struct-missing-tag	<a href="#">23, 35</a>		
struct-no-objnum	<a href="#">23, 24</a>		
struct-orphan	<a href="#">23, 25</a>		
struct-Ref-unknown	<a href="#">42</a>		
struct-show-closing	<a href="#">23, 40</a>		
struct-stack <sub>␣</sub> (show-key)	<a href="#">43, 235</a>		
struct-unknown	<a href="#">23, 22</a>		
struct-used-twice	<a href="#">23, 36</a>		
Structure keys:			
actualtext	<a href="#">1, 724</a>		
AF	<a href="#">1, 937</a>		
AFinline	<a href="#">1, 937</a>		



```

\tag_mc_artifact_group_end: . . . . . 79, 60, 61, 71
\tag_mc_begin:n . . . . . 11,
79, 25, 66, 114, 169, 169, 295, 295,
299, 305, 409, 420, 497, 525, 575, 641
\tag_mc_begin_pop:n . . . . . 79,
76, 80, 81, 102, 506, 536, 605, 651
\tag_mc_end: . . . . .
79, 31, 75, 93, 216, 216, 295, 296,
359, 365, 411, 422, 503, 532, 580, 649
\tag_mc_end_push: . . . . .
79, 65, 80, 80, 83, 491, 518, 591, 639
\tag_mc_if_in: . . . . . 82, 233
\tag_mc_if_in:TF . . . . . 79, 42, 68, 226
\tag_mc_if_in_p: . . . . . 79, 68, 226
\tag_mc_new_stream:n 80, 17, 17, 67, 67
\tag_mc_reset_box:N 79, 79, 79, 228, 228
\tag_mc_use:n . . . . . 79, 36, 36, 36, 38
\tag_resume:n . . . . .
7, 73, 159, 195, 208, 218, 502, 531
\tag_socket_use:n . . . . .
45, 46, 61, 62, 427, 428
\tag_socket_use:nn . . . 45, 46, 61, 67
\tag_socket_use:nnn . . . 45, 46, 61, 72
\tag_socket_use_expandable:n . . .
45, 46, 61, 78
\tag_spacechar_off: . . . 82, 82, 87, 116
\tag_spacechar_on: . . . 82, 83, 99, 120
\tag_start: . . . . . 7, 159, 170, 183, 212
\tag_start:n . . . . . 7, 159, 208, 216, 218
\tag_stop: . . . 7, 54, 159, 161, 182, 211
\tag_stop:n . . . . . 7, 159, 194, 215, 217
\tag_struct_begin:n . 110, 48, 524,
574, 597, 640, 1112, 1112, 1116, 1117
\tag_struct_end: . . . . .
110, 26, 53, 533, 581,
602, 650, 1112, 1113, 1269, 1270, 1310
\tag_struct_end:n . . . 110, 1114, 1307
\tag_struct_gput:nnn . . . . .
111, 920, 1407, 1407, 1409, 1417
\tag_struct_gput_ref:nnn . . . . . 111
\tag_struct_insert_annot:nn . . .
110, 149, 756,
780, 804, 828, 851, 1481, 1481, 1490
\tag_struct_object_ref:n . . . 110,
871, 884, 899, 910, 1400, 1401, 1405
\tag_struct_parent_int: . . . . .
110, 149, 746, 757, 770, 781,
794, 805, 818, 829, 841, 852, 1481, 1491
\tag_struct_use:n . . . . .
110, 111, 58, 1313, 1313, 1315
\tag_struct_use_num:n . . . . .
110, 234, 1354, 1354, 1356, 1398
\tag_suspend:n . . . . .
7, 68, 159, 184, 194, 217, 498, 526
\tag_tool:n . . . . . 13, 13, 14, 16, 20
tag internal commands:
\tag_activate_linkbin: . . 221, 240
\tag_activate_mark_space . . . . . 564
\tag_active_mc_bool . . . . .
40, 83, 245, 252, 261, 300
\tag_active_mc_bool . . . . .
89, 166, 176, 190, 201, 264, 300
\tag_active_socket_bool . . . . .
64, 69, 74,
80, 85, 89, 100, 167, 177, 191, 202, 282
\tag_active_space_bool . . . . .
13, 57, 62, 83
\tag_active_struct_bool . . . . .
83, 247, 254, 260, 296, 310, 468
\tag_active_struct_bool . . . . .
89, 165, 175, 189, 200, 263, 310
\tag_active_struct_dest_bool . . . . .
83, 251, 258, 295
\tag_active_tree_bool . . . . .
9, 68, 83, 246, 253, 262, 351, 389
\tag_add_missing_mcs:Nn . . . . .
94, 167, 167, 219
\tag_add_missing_mcs_to_-
stream:Nn . 65, 65, 66, 189, 189, 225
\tag_attr_class_used_prop . . . . .
291, 293, 1499, 1539
\tag_attr_class_used_seq 289, 1504
\tag_attr_entries_prop . . . . .
295, 1499, 1507, 1535, 1575, 1580, 1584
\tag_attr_new_entry:nn . . . . .
512, 1505, 1505, 1511, 1516, 1520
\tag_attr_objref_prop . . . . .
1499, 1579, 1586, 1591
\tag_attr_value_tl . . . . . 1499,
1569, 1588, 1593, 1595, 1599, 1603
\tag_backend_create_bdc_node . . 438
\tag_backend_create_bmc_node . . 409
\tag_backend_create_emc_node . . 380
\tag_check_add_tag_role:nn . . . . .
129, 358, 358
\tag_check_add_tag_role:nnn . . . . .
169, 377
\tag_check_benchmark_tic: . 356,
360, 364, 368, 372, 376, 380, 641, 647
\tag_check_benchmark_toc: . 358,
362, 366, 370, 374, 378, 382, 642, 648
\tag_check_forbidden_parent_-
child:nnnn . . . . . 120, 120, 134, 171
\tag_check_if_active_mc: . . . . . 298

```

\__tag_check_if_active_mc:TF ...	\__tag_check_timeout_v:n .....
..... 85, 104,	..... 110, 111, 114,
171, 191, 218, <u>297</u> , 301, 307, 361, 367	149, 157, 164, 202, 211, <u>230</u> , 230, 265
\__tag_check_if_active_struct: . 308	\__tag_check_unresolved_parent_-
\__tag_check_if_active_struct:TF	child:nnnn ..... 169, 169
..... 40, <u>297</u> , 1119, 1120,	\g__tag_css_bool .. 864, 865, 868, 879
1274, 1275, 1309, 1317, 1358, 1484	\g__tag_css_prop .....
\__tag_check_if_mc_in_galley: .. 507	.... 856, 857, 870, 883, 884, 886, 887
\__tag_check_if_mc_in_galley:TF	\__tag_debug_mc_begin_ignore:n .
..... 198, 219	..... 354, 542
\__tag_check_if_mc_tmb_missing: 513	\__tag_debug_mc_begin_insert:n .
\__tag_check_if_mc_tmb_missing:TF	..... 309, 535
..... 112, 207, 224, <u>513</u>	\__tag_debug_mc_end_ignore: 379, 556
\__tag_check_if_mc_tmb_missing_-	\__tag_debug_mc_end_insert: 369, 549
p: ..... <u>513</u>	\__tag_debug_struct_begin_-
\__tag_check_if_mc_tme_missing: 524	ignore:n ..... 584, 1267
\__tag_check_if_mc_tme_missing:TF	\__tag_debug_struct_begin_-
..... 155, 211, 228, <u>524</u>	insert:n ..... 576, 1264
\__tag_check_if_mc_tme_missing_-	\__tag_debug_struct_end_check:n
p: ..... <u>524</u>	..... 606, 1309
\__tag_check_info_closing_-	\__tag_debug_struct_end_ignore:
struct:n ..... 335, 335, 343, 1281	..... 599, 1304
\__tag_check_init_mc_used: ....	\__tag_debug_struct_end_insert:
..... 437, 437, 440, 446	..... 591, 1302
\__tag_check_mc_if_nested: ....	\__tag_exclude_headfoot_begin: .
..... 174, 312, <u>396</u> , 396	..... 486, 547, 548
\__tag_check_mc_if_open: .....	\__tag_exclude_headfoot_end: ...
..... 220, 371, <u>396</u> , 404	..... 500, 549, 550
\__tag_check_mc_in_galley:TF ... <u>507</u>	\__tag_exclude_struct_headfoot_-
\__tag_check_mc_in_galley_p: ... <u>507</u>	begin:n ..... 513, 554, 555
\__tag_check_mc_pushed_popped:nn	\__tag_exclude_struct_headfoot_-
..... 90, 97, 110, 113, 118, <u>411</u> , 411	end: ..... 529, 556, 557
\__tag_check_mc_tag:N .....	\__tag_fakespace ..... <u>493</u>
..... 193, 330, <u>423</u> , 423	\__tag_fakespace: ..... 72, 74, 307
\__tag_check_mc_used:n .....	\__tag_finish_structure: .....
..... 145, 268, <u>442</u> , 442	..... 13, 16, <u>348</u> , 349
\g__tag_check_mc_used_intarray .	\l__tag_get_child_tmpa_tl .....
..... 437, 447, 449, 452	..... 60, 571, 576, 643, 645, 655,
\__tag_check_no_open_struct: ...	658, 669, 1323, 1327, 1329, 1335, 1345
..... 344, 344, 1283, 1291	\l__tag_get_child_tmpp_tl .....
\__tag_check_para_begin_show:nn 404	..... 60, 572, 577, 644, 656
\__tag_check_para_end_show:nn .. 415	\l__tag_get_child_tmpp_tl .....
\c__tag_check_pdfversion_tl ....	..... 60, 145, 157, 159
..... 237, 240, 242, 243	\__tag_get_data_mc_counter: .... <u>9</u> , 9
\__tag_check_show_MCID_by_page:	\__tag_get_data_mc_tag: .....
..... <u>461</u> , 461	..... <u>237</u> , 237, <u>293</u> , 293
\__tag_check_struct_forbidden_-	\__tag_get_data_struct_counter:
parent_child:nnn ... 137, 163, 630	..... 549, 550
\__tag_check_struct_used:n ....	\__tag_get_data_struct_id: <u>538</u> , 538
..... 348, 348, 1322	\__tag_get_data_struct_num: <u>543</u> , 544
\__tag_check_structure_has_tag:n	\__tag_get_data_struct_tag: <u>530</u> , 530
..... <u>318</u> , 318, 1152	\__tag_get_mathsubtype ..... <u>304</u>
\__tag_check_structure_tag:N ...	\__tag_get_mc_abs_cnt: .....
..... <u>328</u> , 328, 698, 721, 772	..... <u>14</u> , 15, 19, 20, 102,



126, 155, 166, 183, 210, 246, 254,  
 272, 286, 307, 321, 331, 400, 408, 428  
 \_\_tag\_get\_mc\_cnt\_type\_tag ..... [298](#)  
 \_\_tag\_get\_num\_from ..... [323](#)  
 \l\_tag\_get\_parent\_tmpa\_tl .....  
     . [60](#), 127, 132, 136, 139, 149, 152,  
     162, 165, 175, 566, 574, 587, 593,  
     597, 600, 667, 669, 727, 730, 740, 743  
 \l\_tag\_get\_parent\_tmpb\_tl .....  
     ..... [60](#), 150,  
     153, 163, 166, 175, 567, 575, 588,  
     604, 608, 611, 668, 728, 731, 741, 744  
 \l\_tag\_get\_parent\_tmpc\_tl .....  
     ..... [60](#), 144, 152, 154  
 \_\_tag\_get\_tag\_from ..... [342](#)  
 \l\_tag\_get\_tmpc\_tl ..... [60](#),  
     183, 188, 206, 208, 209, 236, 238,  
     239, 1201, 1207, 1426, 1428, 1432, 1438  
 \\_\_tag\_gincr\_para\_begin\_int: ...  
     ..... [329](#), 333, 351, 367  
 \\_\_tag\_gincr\_para\_end\_int: ...  
     ..... [329](#), 341, 359, 369  
 \\_\_tag\_gincr\_para\_main\_begin\_  
     int: ..... [329](#), 329, 347, 366  
 \\_\_tag\_gincr\_para\_main\_end\_int:  
     ..... [329](#), 337, 355, 368  
 \\_\_tag\_headfoot\_tagged\_begin:n .  
     ..... 584, 616, 617  
 \\_\_tag\_headfoot\_tagged\_end: ....  
     ..... 600, 618, 619  
 \\_\_tag\_hook\_kernel\_after\_foot: .  
     ..... 469, 482, 550, 557, 564, 619  
 \\_\_tag\_hook\_kernel\_after\_head: .  
     ..... 467, 474, 549, 556, 563, 618  
 \\_\_tag\_hook\_kernel\_before\_foot:  
     ..... 468, 480, 548, 555, 562, 617  
 \\_\_tag\_hook\_kernel\_before\_head:  
     ..... 466, 472, 547, 554, 561, 616  
 \g\_tag\_in\_mc\_bool ..... [16](#),  
     18, 175, 221, 228, 313, 372, 494,  
     495, 509, 521, 522, 539, 594, 595, 608  
 \_\_tag\_insert\_bdc\_node ..... [438](#)  
 \_\_tag\_insert\_bmc\_node ..... [409](#)  
 \_\_tag\_insert\_emc\_node ..... [380](#)  
 \_\_tag\_log ..... [226](#)  
 \l\_tag\_loglevel\_int .....  
     ..... [82](#), 125, 132, 170, 173, 261,  
     264, 267, 268, 269, 337, 367, 386,  
     414, 417, 444, 512, 537, 544, 551,  
     558, 562, 578, 585, 586, 593, 601, 608  
 \_\_tag\_mark\_spaces ..... [498](#)  
 \\_\_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 .....  
     ..... [94](#), 21, 90, 111,  
     161, 206, 208, 216, 221, 223, 509, 526  
 \\_\_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 .....  
     ..... [93](#), 21, 87, 110, 196, 199,  
     200, 205, 207, 208, 222, 509, 517, 519  
 \g\_tag\_mc\_footnote\_marks\_seq ... [14](#)  
 \\_\_tag\_mc\_get\_marks: . [84](#), 84, 197, 218  
 \\_\_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](#), 398, 406  
 \\_\_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, 311  
 \\_\_tag\_mc\_insert\_mcid\_single\_  
     kids:n ..... [131](#), 136, 312  
 \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](#), [420](#)  
 \\_\_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](#), [124](#), [135](#), [248](#), [303](#), [319](#),  
         360, [369](#), [489](#), [504](#), [516](#), [534](#), [589](#), [603](#)  
 \l\_\_tag\_name\_link\_tl . . . [625](#), [627](#), [628](#)  
 \_\_tag\_pairs\_prop . . . . . [243](#)  
 \l\_\_tag\_para\_attr\_class\_tl . . . . [311](#)  
 \g\_\_tag\_para\_begin\_int . . . . .  
     . . . . . [311](#), [335](#), [353](#), [410](#), [445](#), [450](#)  
 \l\_\_tag\_para\_bool . . . . . [311](#), [377](#),  
     386, [393](#), [399](#), [458](#), [459](#), [488](#), [515](#), [588](#)  
 \g\_\_tag\_para\_end\_int . . . . .  
     . . . . . [311](#), [343](#), [361](#), [421](#), [445](#), [451](#)  
 \l\_\_tag\_para\_flattened\_bool . . . .  
     . . . . . [311](#), [382](#), [389](#), [402](#)  
 \l\_\_tag\_para\_main\_attr\_class\_tl [311](#)  
 \g\_\_tag\_para\_main\_begin\_int . . . .  
     . . . . . [311](#), [331](#), [349](#), [436](#), [441](#)  
 \g\_\_tag\_para\_main\_end\_int . . . . .  
     . . . . . [311](#), [339](#), [357](#), [436](#), [442](#)  
 \\_\_tag\_para\_main\_store\_struct: .  
     . . . . . [371](#), [371](#)  
 \g\_\_tag\_para\_main\_struct\_tl [311](#), [373](#)  
 \l\_\_tag\_para\_main\_tag\_tl . . . . .  
     . . . . . [311](#), [381](#), [388](#), [401](#)  
 \l\_\_tag\_para\_show\_bool . . . . .  
     . . . . . [311](#), [378](#), [379](#), [394](#), [407](#), [418](#)  
 \l\_\_tag\_para\_tag\_default\_tl . . . . [311](#)  
 \l\_\_tag\_para\_tag\_tl . . . . .  
     . . . . . [311](#), [380](#), [387](#), [395](#), [400](#)

\l\_\_tag\_parent\_child\_check\_tl . .  
     . . . . . [156](#), [157](#), [169](#), [172](#), [500](#),  
         620, [621](#), [628](#), [631](#), [733](#), [734](#), [746](#), [748](#)  
 \\_\_tag\_parenttree\_add\_objr:nn . .  
     . . . . . [163](#), [163](#), [493](#), [521](#)  
 \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 . . . . . [107](#), [107](#)  
 \_\_tag\_pdf\_object\_ref . . . . . [468](#)  
 \\_\_tag\_prop\_gput:Nnn . . . . .  
     . . . . . [9](#), [29](#), [89](#), [98](#), [99](#), [111](#),  
         120, [121](#), [128](#), [132](#), [140](#), [143](#), [146](#),  
         149, [151](#), [201](#), [205](#), [217](#), [220](#), [282](#),  
         285, [314](#), [315](#), [384](#), [1328](#), [1469](#), [1476](#)  
 \\_\_tag\_prop\_item:Nn . . . [9](#), [52](#), [140](#), [147](#)  
 \\_\_tag\_prop\_new:N . . . . . [9](#), [9](#),  
     11, [19](#), [24](#), [32](#), [110](#), [140](#), [140](#), [154](#), [1124](#)  
 \\_\_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](#), [479](#), [730](#)  
 \\_\_tag\_property\_ref\_lastpage:nn  
     . . . . . [83](#), [119](#), [119](#), [160](#), [174](#), [177](#), [465](#), [479](#)  
 \c\_\_tag\_property\_struct\_clist [80](#), [732](#)  
 \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, [573](#), [590](#), [592](#), [640](#), [717](#), [729](#), [742](#)  
 \l\_\_tag\_role\_debug\_prop . . . . . [11](#)  
 \\_\_tag\_role\_get:nnNN . . . . . [154](#),  
     156, [164](#), [227](#), [229](#), [253](#), [714](#), [765](#), [1163](#)  
 \\_\_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](#)  
 \g\_\_tag\_role\_NS\_<ns>\_prop . . . . [181](#)  
 \g\_\_tag\_role\_NS\_mathml\_prop [267](#), [486](#)

\__tag_role_NS_new:nnn	.....	__tag_show_spacemark	.....	479
. 183, 20, 22, 30, 74, 75, 76, 77, 78, 80		\l__tag_showspaces_bool	...	7, 16, 17
\g__tag_role_NS_prop	.....	\g__tag_softthyphen_bool	.....	
..... 181, 9, 26, 56, 183, 326, 344, 786		..... 95, 277, 278, 281, 296, 311, 326		
\g__tag_role_parent_child-		__tag_space_chars_shipout	.....	596
intarray	390, 397, 406, 421, 425, 555	\__tag_start_para_ints:	.....	
\__tag_role_read_namespace:n	337,	..... 178, 203, 345, 345		
337, 341, 342, 343, 345, 347, 349, 350		\__tag_stop_para_ints:	.....	
\__tag_role_read_namespace:nn	..	..... 168, 192, 345, 364		
..... 318, 318, 339, 348		\__tag_store_parent_child-		
\__tag_role_read_namespace-		rule:nnn	.....	391, 393, 417, 462
line:nw	..... 255, 259, 292, 328	g__tag_struct_1_prop	.....	109
\l__tag_role_role_namespace-		\__tag_struct_add_AF:nn	.....	
tmpa_tl	..... 12,	..... 950, 967, 987, 994, 1014, 1059		
757, 778, 783, 787, 790, 794, 809		\__tag_struct_add_inline_AF:nn	..	
\l__tag_role_role_tmpa_tl	.....	..... 939, 966, 1028, 1032, 1039, 1049		
..... 12, 756, 776, 782, 802, 808		\l__tag_struct_addkid_tl	88, 774, 1238	
\g__tag_role_rolemap_prop	.....	\g__tag_struct_AFobj_int	937, 945, 948	
..... 181, 18, 144, 146, 149, 158,		\__tag_struct_check_parent-		
214, 217, 220, 269, 272, 385, 604, 614		child:nn	580, 580, 635, 671, 680, 1225	
\c__tag_role_rule_checkparent_tl		\__tag_struct_check_parent-		
..... 157, 173, 621, 734		child_aux:nnnnN	.. 555, 556, 615, 623	
\c__tag_role_rules_num_prop	....	\g__tag_struct_cont_mc_prop	....	
..... 391, 514, 564		..... 11, 95, 96, 98, 101, 246		
\c__tag_role_rules_prop	391, 395, 419	\g__tag_struct_dest_num_prop	...	
\l__tag_role_tag_namespace_tmpa-		..... 90, 880, 893		
tl	..... 12, 755, 807	\l__tag_struct_elem_stash_bool	..	
\l__tag_role_tag_namespace_tmpb-		..... 87, 734, 1188, 1221, 1251		
tl	..... 14	\__tag_struct_exchange_kid-		
\l__tag_role_tag_namespace_tmpb-		command:N	..... 307, 307, 316, 347	
tluuuuu%	..... 12	\__tag_struct_fill_kid_key:n	...	
\l__tag_role_tag_tmpa_tl	.....	..... 136, 317, 317, 449		
..... 12, 754, 775, 801, 806		\__tag_struct_format_P:nnN	....	411
\g__tag_role_tags_class_prop	...	\__tag_struct_format_parentnum:nnN	.....	414, 414
..... 181, 8, 90, 99, 112, 121, 137, 268		..... 411, 412		
\g__tag_role_tags_NS_prop	.....	\__tag_struct_format_Ref	.....	135
181, 7, 88, 97, 110, 119, 130, 330,		\__tag_struct_format_Ref:nnN	418, 418	
365, 383, 431, 687, 703, 754, 781, 1297		\__tag_struct_format_rolemap:nnN	.....	411, 411
\l__tag_role_tmpa_seq	..... 12	\__tag_struct_format_tag:nnN	411, 413	
\l__tag_role_update_bool	.....	\__tag_struct_get_dict_content:nn	.....	138, 397, 397, 450
..... 208, 255, 256, 264, 344, 346		\__tag_struct_get_id:n	.....	
\c__tag_role_userNS_id_str	....	. 96, 101, 114, 115, 150, 151, 456, 540		
..... 182, 59, 80		\__tag_struct_get_role:nnNN	....	
\g__tag_root_default_tl	..... 274	..... 146, 159, 197, 197,		
\g__tag_saved_in_mc_bool	.....	216, 563, 568, 640, 652, 664, 724, 737		
..... 485, 494, 509, 521, 539, 594, 608		\__tag_struct_gput_data_attribute:nn	.....	1462, 1462
\__tag_seq_gput_left:Nn	.....	\__tag_struct_gput_data_ref:nn	..	
..... 9, 40, 145, 153, 270		..... 1440, 1461		
\__tag_seq_gput_right:Nn	..... 9,			
35, 140, 144, 152, 233, 243, 254, 293				
\__tag_seq_item:Nn	... 9, 47, 140, 146			
\__tag_seq_new:N	.....			
..... 9, 9, 22, 111, 140, 142, 155, 1126				
\__tag_seq_show:N	9, 58, 140, 148, 156			

```

\__tag_struct_gput_data_ref_-
  aux:nnn ..... 1419,
    1420, 1442, 1446, 1450, 1454, 1458
\__tag_struct_gput_data_ref_-
  dest:nn ..... 1448
\__tag_struct_gput_data_ref_-
  dest_parent:nn ..... 1452
\__tag_struct_gput_data_ref_-
  label:nn ..... 1444
\__tag_struct_gput_data_ref_-
  num:nn ..... 1456
\__tag_struct_insert_annot:nn ..
  ..... 464, 464, 1486
\__tag_struct_insert_annot_-
  shipout:nnn ..... 505, 505
\__tag_struct_kid_mc_gput_-
  right:nn ... 217, 229, 230, 249, 269
\__tag_struct_kid_OBJR_gput_-
  right:nnn 282, 282, 285, 306, 480, 508
\__tag_struct_kid_struct_gput_-
  left:nn ..... 266, 266, 267, 281
\__tag_struct_kid_struct_gput_-
  right:nn .....
    ..... 250, 250, 251, 265, 1325, 1370
g__tag_struct_kids_1_seq ..... 109
\g__tag_struct_label_num_prop ..
  ..... 86, 728, 867
\l__tag_struct_lang_tl .....
  ..... 464, 1110, 1134, 1139
\c__tag_struct_link_bin_tl 226, 234
\__tag_struct_mcid_dict:n .....
  ..... 98, 101, 217, 236
\c__tag_struct_null_tl ..... 10, 351
\g__tag_struct_objR_seq ..... 8
\l__tag_struct_parenttag_NS_tl .
  ..... 78, 764, 767, 771, 1194
\l__tag_struct_parenttag_tl ....
  ..... 78, 763, 766, 770, 772, 1194
\__tag_struct_prop_gput:nnn .. 95,
  96, 97, 103, 113, 118, 123, 128,
  133, 140, 166, 170, 179, 185, 190,
  353, 366, 380, 786, 798, 812, 828,
  843, 851, 931, 953, 996, 1015, 1060,
  1130, 1136, 1141, 1175, 1190, 1203,
  1213, 1229, 1373, 1435, 1549, 1600
\g__tag_struct_ref_by_dest_prop . 93
\__tag_struct_Ref_dest:nN . 857, 878
\__tag_struct_Ref_dest_parent:nN
  ..... 857, 891
\__tag_struct_Ref_label:nN 857, 865
\__tag_struct_Ref_num:nN .. 857, 906
\__tag_struct_Ref_obj:nN .. 857, 857
\g__tag_struct_roletag_NS_tl .... 78

\l__tag_struct_roletag_NS_tl ...
  ..... 81, 1167, 1179, 1217
\g__tag_struct_roletag_stack_seq
  ..... 15, 17, 292, 1170, 1277
\l__tag_struct_roletag_tl .....
  78, 1166, 1169, 1171, 1179, 1181, 1217
\__tag_struct_set_attribute: ...
  ..... 25, 39, 1173, 1288
\__tag_struct_set_tag_info:nnn .
  ..... 161, 163, 177, 196, 1148
\g__tag_struct_stack_current_tl
  .. 18, 29, 33, 38, 69, 75, 106, 148,
  154, 162, 208, 270, 274, 299, 344,
  535, 540, 546, 1172, 1236, 1240,
  1241, 1262, 1281, 1287, 1326, 1332,
  1338, 1344, 1371, 1377, 1383, 1389
\l__tag_struct_stack_parent_-
  tmpa_tl .. 18, 473, 482, 499, 744,
  1146, 1153, 1157, 1199, 1226, 1233,
  1237, 1239, 1242, 1254, 1255, 1263
\g__tag_struct_stack_seq .....
  ..... 12, 22, 25, 472, 723,
  736, 1156, 1162, 1174, 1273, 1279, 1285
\c__tag_struct_StructElem_-
  entries_seq ..... 41
\c__tag_struct_StructTreeRoot_-
  entries_seq ..... 41
\g__tag_struct_tag_NS_tl 78, 697,
  713, 716, 720, 1151, 1165, 1261, 1299
\g__tag_struct_tag_stack_seq ...
  ..... 14, 50, 238,
  239, 581, 596, 610, 1168, 1278, 1293
\g__tag_struct_tag_tl ..... 78,
  179, 180, 183, 314, 315, 427, 428,
  696, 698, 712, 715, 719, 721, 1150,
  1164, 1169, 1295, 1297, 1339, 1384
\__tag_struct_use_check_parent_-
  child:nn . 636, 636, 683, 1343, 1388
\__tag_struct_write_obj ..... 135
\__tag_struct_write_obj:n .....
  ..... 151, 430, 430
\l__tag_tag_stop_int 159, 163, 164,
  172, 173, 180, 187, 188, 197, 198, 206
\g__tag_tagunmarked_bool 94, 273, 275
\l__tag_tmp_unused_tl 63, 130, 323,
  330, 395, 398, 402, 419, 422, 431,
  687, 690, 703, 706, 754, 757, 788, 1575
\l__tag_tmp_unused_tl\l__tag_-
  tag_Ref_tmpa_tl ..... 60
\l__tag_tmpa_box .....
  ..... 60, 171, 177, 178, 182, 193, 194
\l__tag_tmpa_clist .....
  .... 60, 1527, 1528, 1561, 1562, 1564

```

\l__tag_tmpa_int .....	60,	\g__tag_tree_openaction_struct_-	
90, 93, 98, 101, 105, 114, 430, 442, 444		tl .....	32, 38, 57
\l__tag_tmpa_prop 60, 176, 189, 203, 205		\__tag_tree_parenttree_rerun_-	
\l__tag_tmpa_seq 51, 58, 59, 60, 321,		msg: .....	171, 220, 255
323, 325, 326, 327, 328, 443, 446,		\__tag_tree_update_openaction: .	
454, 455, 457, 458, 459, 474, 494,		.....	42, 75
504, 689, 693, 696, 697, 705, 709,		\__tag_tree_write_classmap: ....	
712, 713, 756, 760, 763, 764, 774,		.....	286, 286, 369
775, 776, 1529, 1533, 1543, 1544,		\__tag_tree_write_idtree: ..	86, 361
1545, 1547, 1565, 1571, 1573, 1597		\__tag_tree_write_namespaces: ..	
\l__tag_tmpa_str .....		.....	322, 322, 373
.... 42, 43, 48, 60, 262, 267, 272,		\__tag_tree_write_parenttree: ..	
297, 302, 309, 399, 404, 416, 421,		.....	246, 246, 357
782, 789, 794, 801, 808, 815, 819,		\__tag_tree_write_rolemap: ....	
820, 824, 825, 831, 839, 846, 927, 934		.....	263, 263, 365
\l__tag_tmpa_tl ... 42, 43, 47, 49,		\__tag_tree_write_structelements:	
50, 51, 56, 60, 86, 88, 93, 94, 96, 98,		.....	147, 147, 377
102, 106, 106, 108, 109, 113, 114,		\__tag_tree_write_structtreeroot:	
116, 118, 119, 137, 138, 139, 141,		.....	126, 126, 381
143, 144, 146, 177, 178, 180, 183,		\g__tag_unique_cnt_int .....	
184, 186, 191, 198, 199, 205, 205,		97, 1081, 1085, 1088, 1098, 1102, 1106	
206, 209, 211, 214, 215, 220, 268,		\__tag_whatsits: .....	
269, 271, 275, 277, 288, 297, 299,		.....	36, 43, 48, 49, 52, 295, 296
300, 302, 306, 308, 308, 309, 310,		tag-namespace_(rolemap-key) .....	752
313, 314, 349, 351, 353, 365, 384,		tag/check/parent-child .....	183
450, 455, 455, 456, 457, 458, 463,		tag/check/parent-child-end .....	183
468, 469, 470, 471, 475, 481, 483,		tag/struct/1 internal commands:	
488, 514, 516, 525, 546, 549, 556,		__tag/struct/1 .....	31
564, 566, 575, 599, 601, 604, 606,		tag/tree/namespaces internal commands:	
610, 614, 618, 620, 624, 645, 653,		__tag/tree/namespaces .....	321
655, 656, 658, 662, 667, 698, 702,		tag/tree/parenttree internal commands:	
717, 719, 723, 725, 736, 738, 768,		__tag/tree/parenttree .....	154
770, 771, 772, 774, 894, 949, 952,		tag/tree/rolemap internal commands:	
1277, 1278, 1279, 1285, 1287, 1293,		__tag/tree/rolemap .....	262
1296, 1297, 1299, 1366, 1465, 1467,		tagabspage .....	8, 123
1468, 1472, 1535, 1541, 1552, 1579		tagmcabs .....	8, 123
\l__tag_tmpb_box .....		\tagmcbegin .....	42, 181, 22
..... 60, 172, 179, 180, 184, 186		\tagmcend .....	42, 22
\l__tag_tmpb_seq .....		tagmcid .....	8, 123
..... 60, 1528, 1529, 1564, 1565		\tagmcifinTF .....	42, 39
\l__tag_tmpb_tl .....	193,	\tagmcuse .....	42, 22
60, 89, 104, 118, 120, 295, 301,		\tagpdfparaOff .....	44, 455
434, 456, 462, 484, 490, 518, 547,		\tagpdfparaOn .....	44, 455
550, 556, 568, 609, 611, 614, 616,		\tagpdfsetup .....	8, 42, 66, 113, 180, 6
621, 625, 672, 680, 682, 683, 685,		\tagpdfsuppressmarks .....	44, 460
689, 694, 699, 703, 718, 720, 769,		\tagstart .....	7, 183, 214
771, 867, 871, 880, 884, 893, 896, 899		\tagstop .....	7, 182, 213
\l__tag_tmpc_tl .....	60, 485, 491	tagstruct .....	8, 123
\__tag_tree_fill_parenttree: ...		\tagstructbegin .....	
.....	171, 172, 253	.. 42, 147, 180, 181, 45, 225, 233, 277	
\__tag_tree_final_checks: 20, 20, 354		\tagstructend .....	42, 45, 227, 235, 278
\g__tag_tree_id_pad_int .. 78, 82, 156		tagstructobj .....	8, 123
\__tag_tree_lua_fill_parenttree:		\tagstructuse .....	42, 45
.....	233, 233, 250		

tagtag@LastPage internal commands:		
\r__tagtag@LastPage	57	
\tagtool	14, 20	
\tagtool␣(deprecated)	13	
tagunmarked (deprecated) (key)	1, 273	
test/lang␣(setup-key)	462	
TeX and L <sup>A</sup> T <sub>E</sub> X 2 <sub>ε</sub> commands:		
\@M	168	
\@bsphack	111	
\@esphack	113	
\@gobble	31, 55	
\@ifpackageloaded	22	
\@maxdepth	181	
\@secondoftwo	31, 55	
\c@chapter	360, 378	
tex commands:		
\tex_botmarks:D	91	
\tex_firstmarks:D	88	
\tex_kern:D	184	
\tex_marks:D	25, 34, 47, 54, 65, 71	
\tex_special:D	52	
\tex_splitbotmarks:D	217	
\tex_splitfirstmarks:D	197	
texsource (key)	1, 937	
\tiny	410, 421	
title (key)	1, 724	
title-o (key)	1, 724	
tl commands:		
\c_empty_tl	365, 385	
\c_space_tl	55, 56, 58, 60, 100, 104, 116, 167, 191, 197, 198, 216, 220, 222, 224, 259, 299, 390, 407, 427, 455, 861, 871, 884, 899, 910, 977, 1254, 1338, 1383, 1472, 1544, 1590	
\tl_clear:N	88, 89, 106, 177, 212, 213, 288, 399	
\tl_const:Nn	10, 226, 237	
\tl_count:n	79, 83, 156	
\tl_gput_left:Nn	866	
\tl_gput_right:Nn	165, 975	
\tl_gset:Nn	20, 33, 38, 106, 225, 243, 275, 287, 320, 373, 373, 390, 696, 697, 712, 713, 719, 720, 982, 1172, 1287, 1295, 1299	
\tl_gset_eq:NN	180, 315	
\tl_head:N	655, 682	
\tl_if_empty:NTF	43, 43, 109, 185, 198, 289, 307, 316, 335, 425, 656, 683, 772, 778, 820, 1134	
\tl_if_empty:nTF	51, 69, 77, 89, 142, 196, 210, 259, 262, 266, 279, 294, 295, 297, 360, 379, 413, 440, 629, 637, 643, 670, 805, 821, 836, 882, 942, 1012	
\tl_if_empty_p:n	310, 659	
\tl_if_eq:NNTF	351, 509, 669	
\tl_if_eq:NnTF	108	
\tl_if_eq:nnTF	212, 240, 274, 278	
\tl_if_exist:NTF	277, 970	
\tl_if_head_eq_charcode:nNTF	49	
\tl_if_in:nnTF	185	
\tl_new:N	11, 12, 12, 13, 14, 15, 16, 19, 19, 22, 22, 23, 24, 25, 26, 32, 33, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 78, 79, 80, 81, 82, 84, 88, 162, 170, 274, 319, 321, 322, 324, 327, 328, 500, 627, 980, 1110, 1502	
\tl_put_right:Nn	94, 104, 118, 195, 207, 226, 251, 256, 266, 267, 281, 297, 301, 302, 394, 403, 404, 407, 409, 416, 420, 420, 421, 425, 859, 869, 882, 897, 908, 1428, 1588, 1595	
\tl_remove_once:Nn	1467, 1468	
\tl_replace_once:Nnn	310	
\tl_set:Nn	42, 83, 85, 86, 89, 118, 139, 160, 162, 180, 183, 189, 193, 197, 201, 205, 208, 209, 209, 213, 217, 224, 235, 238, 239, 242, 243, 244, 249, 250, 271, 275, 298, 302, 306, 316, 323, 325, 326, 349, 389, 427, 463, 508, 516, 518, 552, 560, 566, 568, 601, 606, 611, 616, 628, 629, 645, 655, 658, 662, 667, 672, 682, 685, 689, 694, 707, 744, 763, 764, 770, 771, 774, 775, 776, 790, 794, 1146, 1323, 1432, 1541, 1569	
\tl_set_eq:NN	179, 314	
\tl_show:N	1236, 1237, 1593, 1599	
\tl_tail:n	533	
\tl_to_str:n	33, 48, 149, 202, 217, 532, 565	
\tl_trim_spaces:n	49	
\tl_use:N	279, 958, 1001, 1020, 1065	
tree-mcid-index-wrong	23, 226	
tree-statistic	23, 54	
tree-struct-still-open	23, 47	
U		
uncompress (deprecated) (key)	260	
unittag␣(deprecated)	375	
\unskip	42	
use commands:		
\use:N	235, 461, 1238	
\use:n	41, 350	
\use_i:nn	99, 102, 109, 138, 154, 159, 208, 238, 365, 385, 599, 610, 614, 618, 1296	

