

# tagpdf – A package to experiment with pdf tagging\*

Ulrike Fischer<sup>†</sup>

Released 2024-09-11

## Contents

<b>1</b>	<b>Initialization and test if pdfmanagement is active.</b>	<b>7</b>
<b>2</b>	<b>base package</b>	<b>7</b>
<b>3</b>	<b>Package options</b>	<b>8</b>
<b>4</b>	<b>Packages</b>	<b>8</b>
	4.1 Patches related to Ref improvement . . . . .	9
	4.2 a LastPage label . . . . .	9
<b>5</b>	<b>Variables</b>	<b>10</b>
<b>6</b>	<b>Variants of l3 commands</b>	<b>11</b>
<b>7</b>	<b>Label and Reference commands</b>	<b>11</b>
<b>8</b>	<b>Setup label attributes</b>	<b>12</b>
<b>9</b>	<b>Commands to fill seq and prop</b>	<b>12</b>
<b>10</b>	<b>General tagging commands</b>	<b>13</b>
<b>11</b>	<b>Keys for tagpdfsetup</b>	<b>14</b>
<b>12</b>	<b>loading of engine/more dependent code</b>	<b>16</b>
<b>I</b>	<b>The tagpdf-checks module</b>	
	Messages and check code	
	Part of the tagpdf package	<b>17</b>
<b>1</b>	<b>Commands</b>	<b>17</b>

---

\*This file describes v0.99e, last revised 2024-09-11.

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

<b>2</b>	<b>Description of log messages</b>	<b>17</b>
2.1	\ShowTagging command . . . . .	17
2.2	Messages in checks and commands . . . . .	18
2.3	Messages from the ptagging code . . . . .	18
2.4	Warning messages from the lua-code . . . . .	18
2.5	Info messages from the lua-code . . . . .	18
2.6	Debug mode messages and code . . . . .	19
2.7	Messages . . . . .	19
<b>3</b>	<b>Messages</b>	<b>21</b>
3.1	Messages related to mc-chunks . . . . .	21
3.2	Messages related to structures . . . . .	22
3.3	Attributes . . . . .	24
3.4	Roles . . . . .	24
3.5	Miscellaneous . . . . .	25
<b>4</b>	<b>Retrieving data</b>	<b>25</b>
<b>5</b>	<b>User conditionals</b>	<b>25</b>
<b>6</b>	<b>Internal checks</b>	<b>26</b>
6.1	checks for active tagging . . . . .	26
6.2	Checks related to structures . . . . .	27
6.3	Checks related to roles . . . . .	28
6.4	Check related to mc-chunks . . . . .	29
6.5	Checks related to the state of MC on a page or in a split stream . . . . .	32
6.6	Benchmarks . . . . .	35
<b>II The tagpdf-user module</b>		
<b>Code related to L<sup>A</sup>T<sub>E</sub>X<sub>2</sub>ε user commands and document commands</b>		
<b>Part of the tagpdf package</b>		<b>36</b>
<b>1</b>	<b>Setup commands</b>	<b>36</b>
<b>2</b>	<b>Commands related to mc-chunks</b>	<b>36</b>
<b>3</b>	<b>Commands related to structures</b>	<b>37</b>
<b>4</b>	<b>Debugging</b>	<b>37</b>
<b>5</b>	<b>Extension commands</b>	<b>38</b>
5.1	Fake space . . . . .	38
5.2	Tagging of paragraphs . . . . .	38
5.3	Header and footer . . . . .	39
5.4	Link tagging . . . . .	39
<b>6</b>	<b>Socket support</b>	<b>39</b>
<b>7</b>	<b>User commands and extensions of document commands</b>	<b>40</b>

<b>8</b>	<b>Setup and preamble commands</b>	<b>40</b>
<b>9</b>	<b>Commands for the mc-chunks</b>	<b>40</b>
<b>10</b>	<b>Commands for the structure</b>	<b>41</b>
<b>11</b>	<b>Socket support</b>	<b>42</b>
<b>12</b>	<b>Debugging</b>	<b>42</b>
<b>13</b>	<b>Commands to extend document commands</b>	<b>46</b>
	13.1 Document structure . . . . .	46
	13.2 Structure destinations . . . . .	47
	13.3 Fake space . . . . .	47
	13.4 Paratagging . . . . .	47
	13.5 Language support . . . . .	55
	13.6 Header and footer . . . . .	55
	13.7 Links . . . . .	57
<b>III The tagpdf-tree module</b>		
<b>Commands trees and main dictionaries</b>		
<b>Part of the tagpdf package</b>		
<b>1</b>	<b>Trees, pdfmanagement and finalization code</b>	<b>59</b>
	1.1 Check structure . . . . .	59
	1.2 Catalog: MarkInfo and StructTreeRoot and OpenAction . . . . .	60
	1.3 Writing the IDtree . . . . .	61
	1.4 Writing structure elements . . . . .	62
	1.5 ParentTree . . . . .	63
	1.6 Rolemap dictionary . . . . .	66
	1.7 Classmap dictionary . . . . .	66
	1.8 Namespaces . . . . .	67
	1.9 Finishing the structure . . . . .	68
	1.10 StructParents entry for Page . . . . .	69
<b>IV The tagpdf-mc-shared module</b>		
<b>Code related to Marked Content (mc-chunks), code shared by</b>		
<b>all modes</b>		
<b>Part of the tagpdf package</b>		
<b>1</b>	<b>Public Commands</b>	<b>70</b>
<b>2</b>	<b>Public keys</b>	<b>71</b>
<b>3</b>	<b>Marked content code – shared</b>	<b>72</b>
	3.1 Variables and counters . . . . .	72
	3.2 Functions . . . . .	73
	3.3 Keys . . . . .	76

<b>V</b>	<b>The tagpdf-mc-generic module</b>	
	Code related to Marked Content (mc-chunks), generic mode	
	Part of the tagpdf package	<b>77</b>
<b>1</b>	<b>Marked content code – generic mode</b>	<b>77</b>
1.1	Variables . . . . .	77
1.2	Functions . . . . .	78
1.3	Looking at MC marks in boxes . . . . .	81
1.4	Keys . . . . .	89
<b>VI</b>	<b>The tagpdf-mc-luacode module</b>	
	Code related to Marked Content (mc-chunks), luamode-specific	
	Part of the tagpdf package	<b>91</b>
<b>1</b>	<b>Marked content code – luamode code</b>	<b>91</b>
1.1	Commands . . . . .	93
1.2	Key definitions . . . . .	97
<b>VII</b>	<b>The tagpdf-struct module</b>	
	Commands to create the structure	
	Part of the tagpdf package	<b>100</b>
<b>1</b>	<b>Public Commands</b>	<b>100</b>
<b>2</b>	<b>Public keys</b>	<b>101</b>
2.1	Keys for the structure commands . . . . .	101
2.2	Setup keys . . . . .	103
<b>3</b>	<b>Variables</b>	<b>103</b>
3.1	Variables used by the keys . . . . .	105
3.2	Variables used by tagging code of basic elements . . . . .	106
<b>4</b>	<b>Commands</b>	<b>106</b>
4.1	Initialization of the StructTreeRoot . . . . .	107
4.2	Adding the /ID key . . . . .	108
4.3	Filling in the tag info . . . . .	109
4.4	Handlings kids . . . . .	110
4.5	Output of the object . . . . .	113
<b>5</b>	<b>Keys</b>	<b>117</b>
<b>6</b>	<b>User commands</b>	<b>124</b>
<b>7</b>	<b>Attributes and attribute classes</b>	<b>133</b>
7.1	Variables . . . . .	133
7.2	Commands and keys . . . . .	133

<b>VIII</b>	<b>The tagpdf-luatex.def</b>	
	Driver for luatex	
	Part of the tagpdf package	<b>137</b>
<b>1</b>	Loading the lua	<b>137</b>
<b>2</b>	Logging functions	<b>141</b>
<b>3</b>	Helper functions	<b>143</b>
	3.1 Retrieve data functions . . . . .	143
	3.2 Functions to insert the pdf literals . . . . .	146
<b>4</b>	Function for the real space chars	<b>148</b>
<b>5</b>	Function for the tagging	<b>151</b>
<b>6</b>	Parenttree	<b>156</b>
<b>IX</b>	<b>The tagpdf-roles module</b>	
	Tags, roles and namespace code	
	Part of the tagpdf package	<b>159</b>
<b>1</b>	Code related to roles and structure names	<b>159</b>
	1.1 Variables . . . . .	160
	1.2 Namespaces . . . . .	162
	1.3 Adding a new tag . . . . .	163
	1.3.1 pdf 1.7 and earlier . . . . .	164
	1.3.2 The pdf 2.0 version . . . . .	166
	1.4 Helper command to read the data from files . . . . .	168
	1.5 Reading the default data . . . . .	170
	1.6 Parent-child rules . . . . .	171
	1.6.1 Reading in the csv-files . . . . .	171
	1.6.2 Retrieving the parent-child rule . . . . .	173
	1.7 Remapping of tags . . . . .	178
	1.8 Key-val user interface . . . . .	178
<b>X</b>	<b>The tagpdf-space module</b>	
	Code related to real space chars	
	Part of the tagpdf package	<b>181</b>
<b>1</b>	Code for interword spaces	<b>181</b>
	<b>Index</b>	<b>185</b>

---

`\tag_stop:` We need commands to stop tagging in some places. They switches three local booleans  
`\tag_start:` and also stop the counting of paragraphs. If they are nested an inner `\tag_start:` will  
`\tagstop` not restart tagging.  
`\tagstart`

---

`\tag_stop:n` `\tag_stop:n{<label>}`  
`\tag_start:n` `\tag_start:n{<label>}`

---

The commands with argument allow to give a label. This is only used in debugging messages to allow to follow the nesting.

---

`activate/spaces_< >(setup-key)`

---

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

---

`activate/mc_< >(setup-key)`  
`activate/tree_< >(setup-key)`  
`activate/struct_< >(setup-key)`  
`activate/all_< >(setup-key)`  
`activate-mc_< >(deprecated)`  
`activate-tree_< >(deprecated)`  
`activate-struct_< >(deprecated)`  
`activate-all_< >(deprecated)`

---

Keys to activate the various tagging steps.

---

`activate/struct-dest_< >(setup-key)`  
`no-struct-dest_< >(deprecated)`

---

The key allows to suppress the creation of structure destinations

---

`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)`  
`tagunmarked_< >(deprecated)`

---

This key allows to set if (in luamode) unmarked text should be marked up as artifact. The initial value is true.

---

`activate/softhyphen_< >(setup-key)`

---

This key allows to activates automatic handling of hyphens inserted by hyphenation. It only is used in luamode and replaces hyphens by U+00AD if the font supports this.

---

`page/tabsorder_< >(setup-key)` This sets the tabsorder on a page. The values are `row`, `column`, `structure` (default)  
`tabsorder_< >(deprecated)` 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} {2024-09-11} {0.99e}
4 { LaTeX kernel code for PDF tagging }
5
6 \bool_if:nF
7 {
8   \bool_lazy_and_p:nn
9     {\cs_if_exist_p:N \pdfmanagement_if_active_p:}
10    { \pdfmanagement_if_active_p: }
11 }
12 { %error for now, perhaps warning later.
13   \PackageError{tagpdf}
14     {
15       PDF-resource-management-is-no-active!\MessageBreak
16       tagpdf-will-no-work.
17     }
18     {
19       Activate-it-with \MessageBreak
20       \string\RequirePackage{pdfmanagement-testphase}\MessageBreak
21       \string\DocumentMetadata{<options>}\MessageBreak
22       before~\string\documentclass
23     }
24 }
25 </package>
<*debug>
26 \ProvidesExplPackage {tagpdf-debug} {2024-09-11} {0.99e}
27 { debug code for tagpdf }
28 \@ifpackageloaded{tagpdf}{\PackageWarning{tagpdf-debug}{tagpdf~not~loaded,~quitting}}\ending
</debug> We map the internal module name “tag” to “tagpdf” in messages.
29 <*package>
30 \prop_gput:Nnn \g_msg_module_name_prop { tag }{ tagpdf }
31 </package>
Debug mode has its special mapping:
32 <*debug>
33 \prop_gput:Nnn \g_msg_module_type_prop { tag / debug} {}
34 \prop_gput:Nnn \g_msg_module_name_prop { tag / debug }{tagpdf~DEBUG}
35 </debug>
```

## 2 base package

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

```

36 <*base>
37 \ProvidesExplPackage {tagpdf-base} {2024-09-11} {0.99e}
38   {part of tagpdf - provide base, no-op versions of the user commands }
39 </base>

```

### 3 Package options

There are only two documented options to switch for luatex between generic and luamode, TODO try to get rid of them. The option `disabledelayedshipout` is only temporary to be able to debug problem with the new shipout keyword if needed.

```

40 <*package>
41 \bool_new:N\g__tag_mode_lua_bool
42 \bool_new:N\g__tag_delayed_shipout_bool
43 \bool_lazy_and:nnT
44   { \bool_if_exist_p:N \l__pdfmanagement_delayed_shipout_bool }
45   { \l__pdfmanagement_delayed_shipout_bool }
46   {
47     \bool_gset_true:N\g__tag_delayed_shipout_bool
48   }
49 \DeclareOption {luamode} { \sys_if_engine luatex:T { \bool_gset_true:N \g__tag_mode_lua_bool }
50 \DeclareOption {genericmode}{ \bool_gset_false:N\g__tag_mode_lua_bool }
51 \DeclareOption {disabledelayedshipout}{ \bool_gset_false:N\g__tag_delayed_shipout_bool }
52 \ExecuteOptions{luamode}
53 \ProcessOptions

```

### 4 Packages

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

```

54 \RequirePackage{tagpdf-base}
55 </package>

```

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

```

56 <*base>
57 \cs_new_protected:Npn \__tag_whatsits: {}
58 \AddToHook{begindocument}
59   {
60     \str_case:VnF \c_sys_backend_str
61     {
62       { luatex } { \cs_set_protected:Npn \__tag_whatsits: {} }
63       { dvisvgm } { \cs_set_protected:Npn \__tag_whatsits: {} }
64     }
65     {
66       \cs_set_protected:Npn \__tag_whatsits: {\tex_special:D {} }
67     }
68   }
69 </base>

```



## 4.1 Patches related to Ref improvement

2024-09-09: Temporary code. Can be removed when the latex-lab-footnote and latex-lab-toc code have been adapted to the better Ref handling.

```
70 <*package>
71 \AddToHook{package/latex-lab-testphase-new-or-2/after}
72 {
73   \cs_set_protected:Npn \__fnote_gput_ref:nn #1 #2 %#1 the structure number receiving the r
74   {
75     \tag_struct_gput:nnn {#1}{ref_num}{#2}
76   }
77 }
78 \AddToHook{package/latex-lab-testphase-toc/after}
79 {
80   \cs_set_protected:Npn \g__tag_struct_ref_by_dest:
81   {
82     \prop_map_inline:Nn\g__tag_struct_ref_by_dest_prop
83     {
84       \tag_struct_gput:nnn {##1}{ref_dest}{##2}
85     }
86   }
87 }
88 </package>
```

## 4.2 a LastPage label

See also issue #2 in Accessible-xref

\\_\_tag\_lastpagelabel:

```
89 <*package>
90 \cs_new_protected:Npn \__tag_lastpagelabel:
91 {
92   \legacy_if:nT { @filesw }
93   {
94     \exp_args:NNne \exp_args:NNe\iow_now:Nn \@auxout
95     {
96       \token_to_str:N \new@label@record
97       { @tag@LastPage }
98       {
99         { abspage } { \int_use:N \g_shipout_readonly_int }
100        { tagmcabs } { \int_use:N \c@g__tag_MCID_abs_int }
101        { tagstruct } { \int_use:N \c@g__tag_struct_abs_int }
102      }
103    }
104  }
105 }
106
107 \AddToHook{enddocument/afterlastpage}
108 { \__tag_lastpagelabel: }
(End of definition for \__tag_lastpagelabel:.)
```

## 5 Variables

```

\l__tag_tmpa_tl A few temporary variables
\l__tag_tmpb_tl
\l__tag_Ref_tmpa_tl
\l__tag_get_tmpc_tl
\l__tag_get_parent_tmpb_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
109 \tl_new:N \l__tag_tmpa_tl
110 \tl_new:N \l__tag_tmpb_tl
111 \tl_new:N \l__tag_Ref_tmpa_tl
112 \tl_new:N \l__tag_get_tmpc_tl
113 \tl_new:N \l__tag_get_parent_tmpa_tl
114 \tl_new:N \l__tag_get_parent_tmpb_tl
115 \str_new:N \l__tag_tmpa_str
116 \prop_new:N \l__tag_tmpa_prop
117 \seq_new:N \l__tag_tmpa_seq
118 \seq_new:N \l__tag_tmpb_seq
119 \clist_new:N \l__tag_tmpa_clist
120 \int_new:N \l__tag_tmpa_int
121 \box_new:N \l__tag_tmpa_box
122 \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
123 \clist_const:Nn \c__tag_property_mc_clist {tagabspace,tagmcabs,tagmcid}
124 \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.

```

125 \int_new:N \l__tag_loglevel_int

```

*(End of definition for \l\_\_tag\_loglevel\_int.)*

```

\g__tag_active_space_bool
\g__tag_active_mc_bool
\g__tag_active_tree_bool
\g__tag_active_struct_bool
\g__tag_active_struct_dest_bool

```

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

```

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

```

132 \bool_new:N \l__tag_active_mc_bool
133 \bool_set_true:N \l__tag_active_mc_bool
134 \bool_new:N \l__tag_active_struct_bool
135 \bool_set_true:N \l__tag_active_struct_bool
136 \bool_new:N \l__tag_active_socket_bool

```

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

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

```

137 \bool_new:N \g__tag_tagunmarked_bool

```

(End of definition for `\g__tag_tagunmarked_bool`.)

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

```

138 \bool_new:N \g__tag_softhyphen_bool

```

(End of definition for `\g__tag_softhyphen_bool`.)

## 6 Variants of l3 commands

```

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

```

## 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`.

```

149 \cs_new_protected:Npn \__tag_property_record:nn #1#2
150 {
151 \@bsphack
152 \property_record:nn{#1}{#2}
153 \@esphack
154 }

```

155

And a few variants

```

156 \cs_generate_variant:Nn \property_ref:nnn {enn}
157 \cs_generate_variant:Nn \property_ref:nn {en}
158 \cs_generate_variant:Nn \__tag_property_record:nn {en,eV}

```

*(End of definition for \\_\_tag\_property\_record:nn.)*

`\__tag_property_ref_lastpage:nn`

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

```

159 \cs_new:Npn \__tag_property_ref_lastpage:nn #1 #2
160 {
161   \property_ref:nnn {@tag@LastPage}{#1}{#2}
162 }

```

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

## 8 Setup label attributes

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

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

```

163 \property_new:nmmm
164 { tagstruct } { now }
165 {1} { \int_use:N \c@g__tag_struct_abs_int }
166 \property_new:nmmm { tagstructobj } { now } {}
167 {
168   \pdf_object_ref_indexed:nn { __tag/struct } { \c@g__tag_struct_abs_int }
169 }
170 \property_new:nmmm
171 { tagabspage } { shipout }
172 {0} { \int_use:N \g_shipout_readonly_int }
173 \property_new:nmmm { tagmcabs } { now }
174 {0} { \int_use:N \c@g__tag_MCID_abs_int }
175
176 \flag_new:n { __tag/mcid }
177 \property_new:nmmm {tagmcid} { shipout }
178 {0} { \flag_height:n { __tag/mcid } }
179

```

*(End of definition for tagstruct and others. These functions are documented on page 7.)*

## 9 Commands to fill seq and prop

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

```

    \_tag_prop_new:N
\_tag_prop_new_linked:N 180 \cs_set_eq:NN \_tag_prop_new:N \prop_new:N
    \_tag_seq_new:N      181 \cs_set_eq:NN \_tag_prop_new_linked:N \prop_new_linked:N
    \_tag_prop_gput:Nnn  182 \cs_set_eq:NN \_tag_seq_new:N \seq_new:N
\_tag_seq_gput_right:Nn 183 \cs_set_eq:NN \_tag_prop_gput:Nnn \prop_gput:Nnn
    \_tag_seq_item:cn    184 \cs_set_eq:NN \_tag_seq_gput_right:Nn \seq_gput_right:Nn
    \_tag_prop_item:cn   185 \cs_set_eq:NN \_tag_seq_item:cn \seq_item:cn
    \_tag_seq_show:N     186 \cs_set_eq:NN \_tag_prop_item:cn \prop_item:cn
    \_tag_prop_show:N    187 \cs_set_eq:NN \_tag_seq_show:N \seq_show:N
    \_tag_prop_show:N    188 \cs_set_eq:NN \_tag_prop_show:N \prop_show:N
189 % cnx temporary needed for latex-lab-graphic code
190 \cs_generate_variant:Nn \_tag_prop_gput:Nnn { Nen , Nee, Nne , cnn, cen, cne, cno, cnx}
191 \cs_generate_variant:Nn \_tag_seq_gput_right:Nn { Ne , No, cn, ce }
192 \cs_generate_variant:Nn \_tag_prop_new:N { c }
193 \cs_generate_variant:Nn \_tag_seq_new:N { c }
194 \cs_generate_variant:Nn \_tag_seq_show:N { c }
195 \cs_generate_variant:Nn \_tag_prop_show:N { c }
196 \endpackage

```

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

## 10 General tagging commands

`\tag_stop:` We need commands to stop tagging in some places. They switch local booleans and also  
`\tag_start:` stop the counting of paragraphs. The commands keep track of the nesting with a local  
`\tag_stop:n` counter. Tagging only is only restarted at the outer level, if the current level is 1. The  
`\tag_start:n` commands with argument allow to give a label. This is only used in debugging messages  
to allow to follow the nesting.

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.

```

\l__tag_tag_stop_int 197 \<package | debug>
198 \<package>\int_new:N \l__tag_tag_stop_int

199 \cs_set_protected:Npn \tag_stop:
200 {
201 \<debug> \msg_note:nne {tag / debug }{tag-stop}{ \int_use:N \l__tag_tag_stop_int }
202 \int_incr:N \l__tag_tag_stop_int
203 \bool_set_false:N \l__tag_active_struct_bool
204 \bool_set_false:N \l__tag_active_mc_bool
205 \bool_set_false:N \l__tag_active_socket_bool
206 \_tag_stop_para_ints:
207 }
208 \cs_set_protected:Npn \tag_start:
209 {
210 \int_if_zero:nF { \l__tag_tag_stop_int } { \int_decr:N \l__tag_tag_stop_int }
211 \int_if_zero:nT { \l__tag_tag_stop_int }
212 {
213 \bool_set_true:N \l__tag_active_struct_bool
214 \bool_set_true:N \l__tag_active_mc_bool
215 \bool_set_true:N \l__tag_active_socket_bool

```

```

216     \tag_start_para_ints:
217   }
218 <debug> \msg_note:nne {tag / debug }{tag-start}{ \int_use:N \l__tag_tag_stop_int }
219   }
220 \cs_set_eq:NN\tagstop\tag_stop:
221 \cs_set_eq:NN\tagstart\tag_start:
222 \cs_set_protected:Npn \tag_stop:n #1
223   {
224 <debug> \msg_note:nnee {tag / debug }{tag-stop}
225 <debug> { \int_use:N \l__tag_tag_stop_int }{\exp_not:n{#1}}
226   \int_incr:N \l__tag_tag_stop_int
227   \bool_set_false:N \l__tag_active_struct_bool
228   \bool_set_false:N \l__tag_active_mc_bool
229   \bool_set_false:N \l__tag_active_socket_bool
230   \tag_stop_para_ints:
231   }
232 \cs_set_protected:Npn \tag_start:n #1
233   {
234   \int_if_zero:nF { \l__tag_tag_stop_int } { \int_decr:N \l__tag_tag_stop_int }
235   \int_if_zero:nT { \l__tag_tag_stop_int }
236     {
237       \bool_set_true:N \l__tag_active_struct_bool
238       \bool_set_true:N \l__tag_active_mc_bool
239       \bool_set_true:N \l__tag_active_socket_bool
240       \tag_start_para_ints:
241     }
242 <debug> \msg_note:nnee {tag / debug }{tag-start}
243 <debug> { \int_use:N \l__tag_tag_stop_int }{\exp_not:n{#1}}
244   }
245 </package | debug>
246 <*base>
247 \cs_new_protected:Npn \tag_stop: {}
248 \cs_new_protected:Npn \tag_start: {}
249 \cs_new_protected:Npn \tagstop {}
250 \cs_new_protected:Npn \tagstart {}
251 \cs_new_protected:Npn \tag_stop:n #1 {}
252 \cs_new_protected:Npn \tag_start:n #1 {}
253 </base>

```

(End of definition for \tag\_stop: and others. These functions are documented on page 6.)

## 11 Keys for tagpdfsetup

TODO: the log-levels must be sorted

**activate/mc<sub>□</sub>(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.

```

254 <*package>
255 \keys_define:nn { __tag / setup }
256   {
257     activate/mc      .bool_gset:N = \g__tag_active_mc_bool,
258     activate/tree    .bool_gset:N = \g__tag_active_tree_bool,

```

```

259 activate/struct .bool_gset:N = \g__tag_active_struct_bool,
260 activate/all .meta:n =
261 {activate/mc={#1},activate/tree={#1},activate/struct={#1}},
262 activate/all .default:n = true,
263 activate/struct-dest .bool_gset:N = \g__tag_active_struct_dest_bool,

```

old, deprecated names

```

264 activate-mc .bool_gset:N = \g__tag_active_mc_bool,
265 activate-tree .bool_gset:N = \g__tag_active_tree_bool,
266 activate-struct .bool_gset:N = \g__tag_active_struct_bool,
267 activate-all .meta:n =
268 {activate/mc={#1},activate/tree={#1},activate/struct={#1}},
269 activate-all .default:n = true,
270 no-struct-dest .bool_gset_inverse:N = \g__tag_active_struct_dest_bool,

```

(End of definition for activate/mc (setup-key) and others. These functions are documented on page 6.)

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

```

271 debug/show .choice:,

```

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

`debug/log` (setup-key)

The log takes currently the values none, v, vv, vvv, all. The description of the log levels is in tagpdf-checks.

`debug/uncompress` (setup-key)

`log` (deprecated)

`uncompress` (deprecated)

```

272 debug/log .choice:,
273 debug/log / none .code:n = {\int_set:Nn \l__tag_loglevel_int { 0 }},
274 debug/log / v .code:n =
275 {
276 \int_set:Nn \l__tag_loglevel_int { 1 }
277 \cs_set_protected:Nn \__tag_check_typeout_v:n { \iow_term:e {##1} }
278 },
279 debug/log / vv .code:n = {\int_set:Nn \l__tag_loglevel_int { 2 }},
280 debug/log / vvv .code:n = {\int_set:Nn \l__tag_loglevel_int { 3 }},
281 debug/log / all .code:n = {\int_set:Nn \l__tag_loglevel_int { 10 }},
282 debug/uncompress .code:n = { \pdf_uncompress: },

```

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

```

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

```

(End of definition for debug/log (setup-key) and others. These functions are documented on page 6.)

`activate/tagunmarked` (setup-key)

`tagunmarked` (deprecated)

This key allows to set if (in luamode) unmarked text should be marked up as artifact. The initial value is true.

```

285 activate/tagunmarked .bool_gset:N = \g__tag_tagunmarked_bool,
286 activate/tagunmarked .initial:n = true,

```

deprecated name

```

287 tagunmarked .bool_gset:N = \g__tag_tagunmarked_bool,

```

(End of definition for activate/tagunmarked (setup-key) and tagunmarked (deprecated). These functions are documented on page 6.)

`activate/softhyphen` (setup-key)

This key activates (in luamode) the handling of soft hyphens.

```

288 activate/softhyphen .bool_gset:N = \g__tag_softhyphen_bool,
289 activate/softhyphen .initial:n = true,

```

(End of definition for `activate/softhyphen` (setup-key). This function is documented on page 6.)

`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  
`tabsorder_□(deprecated)` needed.

```
290 page/tabsorder .choice:,
291 page/tabsorder / row .code:n =
292 \pdfmanagement_add:nnn { Page } {Tabs}{/R},
293 page/tabsorder / column .code:n =
294 \pdfmanagement_add:nnn { Page } {Tabs}{/C},
295 page/tabsorder / structure .code:n =
296 \pdfmanagement_add:nnn { Page } {Tabs}{/S},
297 page/tabsorder / none .code:n =
298 \pdfmanagement_remove:nn {Page} {Tabs},
299 page/tabsorder .initial:n = structure,
```

deprecated name

```
300 tabsorder .meta:n = {page/tabsorder={#1}},
301 }
```

(End of definition for `page/tabsorder` (setup-key) and `tabsorder` (deprecated). These functions are documented on page 6.)

## 12 loading of engine/more dependent code

```
302 \sys_if_engine luatex:T
303 {
304   \file_input:n {tagpdf-luatex.def}
305 }
306 </package>
307 <*mcloding>
308 \bool_if:NTF \g__tag_mode_lua_bool
309 {
310   \RequirePackage {tagpdf-mc-code-lua}
311 }
312 {
313   \RequirePackage {tagpdf-mc-code-generic} %
314 }
315 </mcloding>
316 <*debug>
317 \bool_if:NTF \g__tag_mode_lua_bool
318 {
319   \RequirePackage {tagpdf-debug-lua}
320 }
321 {
322   \RequirePackage {tagpdf-debug-generic} %
323 }
324 </debug>
```



## Part I

# The `tagpdf-checks` module

## Messages and check code

### Part of the `tagpdf` package

## 1 Commands

---

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

---

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

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

---

`\tag_if_box_tagged_p:N` \* `\tag_if_box_tagged:N{<box>}`

`\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_t1` to a positive value. The LaTeX commands will do that automatically at some time but it is in the responsibility of the user to ensure that when using low-level code. If the internal command doesn't exist the box is assumed to be untagged.

## 2 Description of log messages

### 2.1 `\ShowTagging` command

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

## 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
in enddocument/info-hook	para-hook-count-wrong	error (warning?)

## 2.3 Messages from the ptgging 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

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

---

`struct-unknown` Various messages related to structure. Check the definition in the code for their meaning and the arguments they take.  
`struct-no-objnum`  
`struct-orphan`  
`struct-faulty-nesting`  
`struct-missing-tag`  
`struct-used-twice`  
`struct-label-unknown`  
`struct-show-closing`

---

---

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

---

---

`tree-statistic` Message issued at the end of the compilation showing the number of objects to write

---

---

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

---

---

`attr-unknown` Message if an attribute is unknown.

---

---

`role-missing` Messages related to role mapping.  
`role-unknown`  
`role-unknown-tag`  
`role-unknown-NS`  
`role-tag`  
`new-tag`  
`role-parent-child`  
`role-remapping`

---

---

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

---

---

`sys-no-interwordspace` Message if an engine doesn't support inter word spaces

---

---

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

```
1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-checks-code} {2024-09-11} {0.99e}
4 {part of tagpdf - code related to checks, conditionals, debugging and messages}
5 </header>
```

## 3 Messages

### 3.1 Messages related to mc-chunks

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

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

*(End of definition for mc-nested. This function is documented on page 19.)*

**mc-tag-missing** If the tag is missing

```
8 \msg_new:nnn { tag } {mc-tag-missing} { required~tag~missing~--~mcid~#1 }
```

*(End of definition for mc-tag-missing. This function is documented on page 19.)*

**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 19.)*

**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 19.)*

**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 19.)*

**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 19.)*

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

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

*(End of definition for mc-current. This function is documented on page 19.)*

## 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 20.)*

**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 20.)*

**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 20.)*

**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 20.)*

**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 20.)*

**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 20.)*

**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 20.)*

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

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

*(End of definition for struct-show-closing. This function is documented on page 20.)*

**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 20.)*

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

```

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

```

*(End of definition for tree-statistic. This function is documented on page 20.)*

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 20.)

### 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 20.)

### 3.4 Roles

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

```

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

```

(End of definition for `role-missing` and others. These functions are documented on page 20.)

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

```

90 \msg_new:nnn { tag } {role-parent-child}
91   { Parent-Child-#1'--->#2'.\Relation-is-#3-\msg_line_context:}

```

(End of definition for `role-parent-child`. This function is documented on page 20.)

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

```

92 \msg_new:nnn { tag } {role-remapping}
93   { remapping-tag-to-#1 }

```

(End of definition for `role-remapping`. This function is documented on page 20.)

`role-tag` Info messages.

```

new-tag          94 \msg_new:nnn { tag } {role-tag}          { mapping-tag-#1-to-role-#2 }
                95 \msg_new:nnn { tag } {new-tag}          { adding-new-tag-#1 }
                96 \msg_new:nnn { tag } {read-namespace}  { reading-namespace-definitions-tagpdf-
                ns-#1.def }
                97 \msg_new:nnn { tag } {namespace-missing}{ namespace-definitions-tagpdf-ns-#1.def-not-found }
                98 \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 20.)



### 3.5 Miscellaneous

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

```
99 \msg_new:nnn { tag } {tree-mcid-index-wrong}
100   {something-is-wrong-with-the-mcid--rerun}
```

(End of definition for `tree-mcid-index-wrong`. This function is documented on page 20.)

`sys-no-interwordspace` Currently only pdf<sub>l</sub>atex and lua<sub>l</sub>atex have some support for real spaces.

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

(End of definition for `sys-no-interwordspace`. This function is documented on page 20.)

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

```
103 \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 and breaks the structure.

```
104 \msg_new:nmmm { tag } {para-hook-count-wrong}
105   {The-number-of-automatic-begin-#1-and-end-#2-#3-para-hooks-differ!}
106   {This-quite-probably-a-coding-error-and-the-structure-will-be-wrong!}
107 </package>
```

(End of definition for `para-hook-count-wrong`. This function is documented on page 20.)

## 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`.

```
108 <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 17.)

## 5 User conditionals

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

`\tag_if_active:TF`

```
109 <*base>
110 \prg_new_conditional:Npnn \tag_if_active: { p , T , TF , F }
111   { \prg_return_false: }
112 </base>
113 <*package>
114 \prg_set_conditional:Npnn \tag_if_active: { p , T , TF , F }
115   {
116     \bool_lazy_all:nTF
117       {
118         {\g__tag_active_struct_bool}
119         {\g__tag_active_mc_bool}
```

```

120     {\g__tag_active_tree_bool}
121     {\l__tag_active_struct_bool}
122     {\l__tag_active_mc_bool}
123   }
124   {
125     \prg_return_true:
126   }
127   {
128     \prg_return_false:
129   }
130 }
131 \endpackage

```

(End of definition for `\tag_if_active:TF`. This function is documented on page 17.)

`\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.

```

132 \begin{base}
133 \prg_new_conditional:Npnn \tag_if_box_tagged:N #1 {p,T,F,TF}
134 {
135   \tl_if_exist:cTF {l_tag_box_\int_use:N #1_tl}
136   {
137     \int_compare:nNnTF {0\tl_use:c{l_tag_box_\int_use:N #1_tl}}>{0}
138     { \prg_return_true: }
139     { \prg_return_false: }
140   }
141   {
142     \prg_return_false:
143     % warning??
144   }
145 }
146 \end{base}

```

(End of definition for `\tag_if_box_tagged:NTF`. This function is documented on page 17.)

## 6 Internal checks

These are checks used in various places in the code.

### 6.1 checks for active tagging

This checks if mc are active.

```

\__tag_check_if_active_mc:TF
\__tag_check_if_active_struct:TF
147 \begin{package}
148 \prg_new_conditional:Npnn \__tag_check_if_active_mc: {T,F,TF}
149 {
150   \bool_lazy_and:nnTF { \g__tag_active_mc_bool } { \l__tag_active_mc_bool }
151   {
152     \prg_return_true:
153   }
154   {

```

```

155         \prg_return_false:
156     }
157 }
158 \prg_new_conditional:Npnn \__tag_check_if_active_struct: {T,F,TF}
159 {
160     \bool_lazy_and:nnTF { \g__tag_active_struct_bool } { \l__tag_active_struct_bool }
161     {
162         \prg_return_true:
163     }
164     {
165         \prg_return_false:
166     }
167 }

```

(End of definition for \\_\_tag\_check\_if\_active\_mc:TF and \\_\_tag\_check\_if\_active\_struct:TF.)

## 6.2 Checks related to structures

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

```

168 \cs_new_protected:Npn \__tag_check_structure_has_tag:n #1 %#1 struct num
169 {
170     \prop_if_in:cnF { g__tag_struct_#1_prop }
171     {S}
172     {
173         \msg_error:nn { tag } {struct-missing-tag}
174     }
175 }

```

(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.

```

176 \cs_new_protected:Npn \__tag_check_structure_tag:N #1
177 {
178     \prop_if_in:NoF \g__tag_role_tags_NS_prop {#1}
179     {
180         \msg_warning:nne { tag } {role-unknown-tag} {#1}
181     }
182 }

```

(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.

```

183 \cs_new_protected:Npn \__tag_check_info_closing_struct:n #1 %#1 struct num
184 {
185     \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
186     {
187         \msg_info:nnn { tag } {struct-show-closing} {#1}
188     }
189 }
190
191 \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.

```
192 \cs_new_protected:Npn __tag_check_no_open_struct:
193   {
194     \msg_error:nn { tag } {struct-faulty-nesting}
195   }
```

(End of definition for `__tag_check_no_open_struct:.`)

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

```
196 \cs_new_protected:Npn __tag_check_struct_used:n #1 % #1 label
197   {
198     \prop_get:cnNT
199       {__tag_struct\_property_ref:enn{tagpdfstruct-#1}{tagstruct}{unknown}_prop}
200       {P}
201     \l__tag_tmpa_tl
202     {
203       \msg_warning:nnn { tag } {struct-used-twice} {#1}
204     }
205   }
```

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

### 6.3 Checks related to roles

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

```
206 \cs_new_protected:Npn __tag_check_add_tag_role:nn #1 #2 % #1 tag, #2 role
207   {
208     \tl_if_empty:nTF {#2}
209     {
210       \msg_error:nnn { tag } {role-missing} {#1}
211     }
212     {
213       \prop_get:NnNTF \g__tag_role_tags_NS_prop {#2} \l_tmpa_tl
214       {
215         \int_compare:nNtT {\l__tag_loglevel_int} > { 0 }
216         {
217           \msg_info:nnnn { tag } {role-tag} {#1} {#2}
218         }
219       }
220       {
221         \msg_error:nnn { tag } {role-unknown} {#2}
222       }
223     }
224   }
```

Similar with a namespace

```
225 \cs_new_protected:Npn __tag_check_add_tag_role:nnn #1 #2 #3 % #1 tag/NS, #2 role #3 namespace
226   {
227     \tl_if_empty:nTF {#2}
228     {
229       \msg_error:nnn { tag } {role-missing} {#1}
```

```

230     }
231     {
232     \prop_get:cnNTF { g__tag_role_NS_#3_prop } {#2} \l_tmpa_tl
233     {
234     \int_compare:nNtT {\l__tag_loglevel_int} > { 0 }
235     {
236     \msg_info:nnnn { tag } {role-tag} {#1} {#2/#3}
237     }
238     }
239     {
240     \msg_error:nnn { tag } {role-unknown} {#2/#3}
241     }
242     }
243 }

```

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

## 6.4 Check related to mc-chunks

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

`\__tag_check_mc_if_open:`

```

244 \cs_new_protected:Npn \__tag_check_mc_if_nested:
245 {
246   \__tag_mc_if_in:T
247   {
248     \msg_warning:nne { tag } {mc-nested} { \__tag_get_mc_abs_cnt: }
249   }
250 }
251
252 \cs_new_protected:Npn \__tag_check_mc_if_open:
253 {
254   \__tag_mc_if_in:F
255   {
256     \msg_warning:nne { tag } {mc-not-open} { \__tag_get_mc_abs_cnt: }
257   }
258 }

```

(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.

```

259 \cs_new_protected:Npn \__tag_check_mc_pushed_popped:nn #1 #2
260 {
261   \int_compare:nNtT
262   { \l__tag_loglevel_int } = { 2 }
263   { \msg_info:nne {tag}{mc-#1}{#2} }
264   \int_compare:nNtT
265   { \l__tag_loglevel_int } > { 2 }
266   {
267     \msg_info:nne {tag}{mc-#1}{#2}
268     \seq_log:N \g__tag_mc_stack_seq
269   }
270 }

```

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

`\__tag_check_mc_tag:N` This checks if the mc has a (known) tag.

```
271 \cs_new_protected:Npn \__tag_check_mc_tag:N #1 %#1 is var with a tag name in it
272   {
273     \tl_if_empty:NT #1
274     {
275       \msg_error:nne { tag } {mc-tag-missing} { \__tag_get_mc_abs_cnt: }
276     }
277     \prop_if_in:NoF \g__tag_role_tags_NS_prop {#1}
278     {
279       \msg_warning:nne { tag } {role-unknown-tag} {#1}
280     }
281   }
```

(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??

```
282 \cs_new_protected:Npn \__tag_check_init_mc_used:
283   {
284     \intarray_new:Nn \g__tag_check_mc_used_intarray { 65536 }
285     \cs_gset_eq:NN \__tag_check_init_mc_used: \prg_do_nothing:
286   }
```

(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.

```
287 \cs_new_protected:Npn \__tag_check_mc_used:n #1 %#1 mcid absnt
288   {
289     \int_compare:nNnT {\l__tag_loglevel_int} > { 2 }
290     {
291       \__tag_check_init_mc_used:
292       \intarray_gset:Nnn \g__tag_check_mc_used_intarray
293         {#1}
294         { \intarray_item:Nn \g__tag_check_mc_used_intarray {#1} + 1 }
295       \int_compare:nNnT
296         {
297           \intarray_item:Nn \g__tag_check_mc_used_intarray {#1}
298         }
299         >
300         { 1 }
301         {
302           \msg_warning:nnn { tag } {mc-used-twice} {#1}
303         }
304     }
305   }
```

(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.

```
306 \cs_new_protected:Npn \_tag_check_show_MCID_by_page:
307 {
308   \tl_set:Nc \l__tag_tmpa_tl
309     {
310       \_tag_property_ref_lastpage:nn
311         {abspage}
312         {-1}
313     }
314   \int_step_inline:nnnn {1}{1}
315     {
316       \l__tag_tmpa_tl
317     }
318     {
319       \seq_clear:N \l_tmpa_seq
320       \int_step_inline:nnnn
321         {1}
322         {1}
323         {
324           \_tag_property_ref_lastpage:nn
325             {tagmcabs}
326             {-1}
327         }
328         {
329           \int_compare:nT
330             {
331               \property_ref:enn
332                 {mcid-###1}
333                 {tagabspage}
334                 {-1}
335             }
336             =
337             ##1
338         }
339         {
340           \seq_gput_right:Nc \l_tmpa_seq
341             {
342               Page##1-###1-
343               \property_ref:enn
344                 {mcid-###1}
345                 {tagmcid}
346                 {-1}
347             }
348         }
349       \seq_show:N \l_tmpa_seq
350     }
351 }
```

*(End of definition for `\_tag_check_show_MCID_by_page:.`)*

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

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

`\_tag\_check\_mc\_in\_galley\_p:` At first we need a test to decide if `\tag\_mc\_begin:n` (tmb) and `\tag\_mc\_end:` (tme) has been used at all on the current galley. As each command issues two slightly different marks we can do it by comparing firstmarks and botmarks. The test assumes that the marks have been already mapped into the sequence with `\@@\_mc\_get\_marks:`. As `\seq\_if\_eq:NNTF` doesn't exist we use the `tl`-test.

```

352 \prg_new_conditional:Npnn \_tag\_check\_if\_mc\_in\_galley: { T,F,TF }
353 {
354   \tl\_if\_eq:NNTF \l\_tag\_mc\_firstmarks\_seq \l\_tag\_mc\_botmarks\_seq
355   { \prg\_return\_false: }
356   { \prg\_return\_true: }
357 }

```

*(End of definition for \\_tag\\_check\\_if\\_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.

```

358 \prg_new_conditional:Npnn \_tag\_check\_if\_mc\_tmb\_missing: { T,F,TF }
359 {
360   \bool\_if:nTF
361   {
362     \str\_if\_eq\_p:ee {\seq\_item:Nn \l\_tag\_mc\_firstmarks\_seq {1}}{e-}
363     ||
364     \str\_if\_eq\_p:ee {\seq\_item:Nn \l\_tag\_mc\_firstmarks\_seq {1}}{b+}
365   }
366   { \prg\_return\_true: }
367   { \prg\_return\_false: }
368 }

```

*(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.

```

369 \prg_new_conditional:Npnn \_tag\_check\_if\_mc\_tme\_missing: { T,F,TF }
370 {
371   \str\_if\_eq:eeTF {\seq\_item:Nn \l\_tag\_mc\_botmarks\_seq {1}}{b+}
372   { \prg\_return\_true: }
373   { \prg\_return\_false: }
374 }

```

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

```

375 </package>

```

```

376 <*debug>

```



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

```

377 \msg_new:nnn { tag / debug } {mc-begin} { MC-begin~#1-with-options:~\tl_to_str:n{#2}~[\msg_line_context:] }
378 \msg_new:nnn { tag / debug } {mc-end} { MC-end~#1~[\msg_line_context:] }
379
380 \cs_new_protected:Npn \__tag_debug_mc_begin_insert:n #1
381 {
382   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
383   {
384     \msg_note:nnnn { tag / debug } {mc-begin} {inserted} { #1 }
385   }
386 }
387 \cs_new_protected:Npn \__tag_debug_mc_begin_ignore:n #1
388 {
389   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
390   {
391     \msg_note:nnnn { tag / debug } {mc-begin} {ignored} { #1 }
392   }
393 }
394 \cs_new_protected:Npn \__tag_debug_mc_end_insert:
395 {
396   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
397   {
398     \msg_note:nnn { tag / debug } {mc-end} {inserted}
399   }
400 }
401 \cs_new_protected:Npn \__tag_debug_mc_end_ignore:
402 {
403   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
404   {
405     \msg_note:nnn { tag / debug } {mc-end} {ignored}
406   }
407 }

```

And now something for the structures

```

408 \msg_new:nnn { tag / debug } {struct-begin}
409 {
410   Struct~\tag_get:n{struct_num}~begin~#1~with~options:~\tl_to_str:n{#2}~\[\msg_line_context:]
411 }
412 \msg_new:nnn { tag / debug } {struct-end}
413 {
414   Struct~end~#1~[\msg_line_context:]
415 }
416 \msg_new:nnn { tag / debug } {struct-end-wrong}
417 {
418   Struct~end~'~#1'~doesn't~fit~start~'~#2'~[\msg_line_context:]
419 }
420
421 \cs_new_protected:Npn \__tag_debug_struct_begin_insert:n #1
422 {
423   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
424   {
425     \msg_note:nnnn { tag / debug } {struct-begin} {inserted} { #1 }
426     \seq_log:N \g__tag_struct_tag_stack_seq

```

```

427     }
428   }
429   \cs_new_protected:Npn \__tag_debug_struct_begin_ignore:n #1
430   {
431     \int_compare:nNnT { \l__tag_loglevel_int } > {0}
432     {
433       \msg_note:nnnn { tag / debug } {struct-begin } {ignored} { #1 }
434     }
435   }
436   \cs_new_protected:Npn \__tag_debug_struct_end_insert:
437   {
438     \int_compare:nNnT { \l__tag_loglevel_int } > {0}
439     {
440       \msg_note:nnn { tag / debug } {struct-end} {inserted}
441       \seq_log:N \g__tag_struct_tag_stack_seq
442     }
443   }
444   \cs_new_protected:Npn \__tag_debug_struct_end_ignore:
445   {
446     \int_compare:nNnT { \l__tag_loglevel_int } > {0}
447     {
448       \msg_note:nnn { tag / debug } {struct-end } {ignored}
449     }
450   }
451   \cs_new_protected:Npn \__tag_debug_struct_end_check:n #1
452   {
453     \int_compare:nNnT { \l__tag_loglevel_int } > {0}
454     {
455       \seq_get:NNT \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
456       {
457         \str_if_eq:eeF
458         {#1}
459         {\exp_last_unbraced:NV\use_i:nn \l__tag_tmpa_tl}
460         {
461           \msg_warning:nnee { tag/debug }{ struct-end-wrong }
462           {#1}
463           {\exp_last_unbraced:NV\use_i:nn \l__tag_tmpa_tl}
464         }
465       }
466     }
467   }

```

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

```

468   \msg_new:nnn { tag / debug } {tag-stop}
469   {
470     \int_if_zero:nTF
471     {#1}
472     {Tagging~stopped}
473     {Tagging~(not)~stopped~(already~inactive)}\
474     level:~#1~==>~\int_eval:n{#1+1}\tl_if_empty:nF{#2}{,~label:~#2}~[\msg_line_context:]
475   }
476   \msg_new:nnn { tag / debug } {tag-start}
477   {

```

```

478 \int_if_zero:nTF
479   {#1}
480   {Tagging~restarted}
481   {Tagging~(not)~restarted}\
482   level:~\int_eval:n{#1+1}~==>~#1\tl_if_empty:nF{#2}{~,~label:~#2}~[\msg_line_context:]
483 }
484 </debug>

```

## 6.6 Benchmarks

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

```

485 <*package>
486 \cs_new_protected:Npn \__tag_check_benchmark_tic: {}
487 \cs_new_protected:Npn \__tag_check_benchmark_toc: {}
488 \cs_new_protected:Npn \tag_check_benchmark_on:
489 {
490   \cs_if_exist:NT \benchmark_tic:
491   {
492     \cs_set_eq:NN \__tag_check_benchmark_tic: \benchmark_tic:
493     \cs_set_eq:NN \__tag_check_benchmark_toc: \benchmark_toc:
494   }
495 }
496 </package>

```

## Part II

# The `tagpdf-user` module

## Code related to L<sup>A</sup>T<sub>E</sub>X2e user commands and document commands Part of the `tagpdf` package

### 1 Setup commands

---

`\tagpdfsetup` `\tagpdfsetup{⟨key val list⟩}`

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).

---

`activate_⟨setup-key⟩` And additional setup key which combine the other activate keys `activate/mc`, `activate/tree`, `activate/struct` and additionally adds a document structure.

---

`\tag_tool:n` `\tag_tool:n{⟨key val⟩}`  
`\tagtool`

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

### 2 Commands related to mc-chunks

---

`\tagmcbegin` `\tagmcbegin {⟨key-val⟩}`  
`\tagmcend` `\tagmcend`  
`\tagmcuse` `\tagmcuse{⟨label⟩}`

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`.

---

`\tagmcifinTF` `\tagmcifin {⟨true code⟩}{⟨false code⟩}`

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

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

### 3 Commands related to structures

---

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

---

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

### 4 Debugging

---

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

---

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

---

<code>mc-data_⟨show-key⟩</code>	<code>mc-data = ⟨number⟩</code>
---------------------------------	---------------------------------

---

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

---

<code>mc-current_⟨show-key⟩</code>	<code>mc-current</code>
------------------------------------	-------------------------

---

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

---

<code>mc-marks_⟨show-key⟩</code>	<code>mc-marks = show use</code>
----------------------------------	----------------------------------

---

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

---

<code>struct-stack_⟨show-key⟩</code>	<code>struct-stack = log show</code>
--------------------------------------	--------------------------------------

---

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

---

<code>debug/structures_⟨show-key⟩</code>	<code>debug/structures = ⟨structure number⟩</code>
--	--

---

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

## 5 Extension commands

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

The commands and keys should be view as experimental!

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

### 5.1 Fake space

---

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

### 5.2 Tagging of paragraphs

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

---

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

---

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

---

---

`\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{\tagmcbegin}\tagstructend}%
```

### 5.3 Header and footer

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

---

```
page/exclude-header-footer□(setup-key) page/exclude-header-footer = true|false|pagination
```

---

### 5.4 Link tagging

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

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

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

## 6 Socket support

---

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

---

The next L<sup>A</sup>T<sub>E</sub>X (2024-06-01) will use special sockets for the tagging.

These sockets will use names starting with `tagsupport/`. Usually, these sockets have exactly two plugs defined: `noop` (when no tagging is requested or tagging is not wanted for some reason) and a second plug that enables the tagging. There may be more, e.g., tagging with special debugging, etc., but right now it is usually just on or off.

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 the socket name is specified without `tagsupport/`, i.e.,

`\UseTaggingSocket{foo}` → `\UseSocket{tagsupport/foo}`

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

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`.

The L3 programming layer versions `\tag_socket_use:n` and `\tag_socket_use:nn` 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} {2024-09-11} {0.99e}
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 36.)*

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

`\tagtool`

```

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

```

*(End of definition for `\tag_tool:n` and `\tagtool`. These functions are documented on page 36.)*

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

```

*(End of definition for \tagmccbegin, \tagmccend, and \tagmccuse. These functions are documented on page 36.)*

**\tagmccifinTF** 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 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.

```

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

```

*(End of definition for \tagmccifinTF. This function is documented on page 36.)*

## 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 37.)*

## 11 Socket support

Until we can be sure that the kernel defines the commands we provide them before redefining them:

```
61 <*base>
62 \providecommand\tag_socket_use:n[1]{}
63 \providecommand\tag_socket_use:nn[2]{}
64 \providecommand\UseTaggingSocket[1]{}
65 </base>

\tag_socket_use:n
\tag_socket_use:nn
\UseTaggingSocket
66 <*package>
67 \cs_set_protected:Npn \tag_socket_use:n #1
68 {
69     \bool_if:NT \l__tag_active_socket_bool
70     { \UseSocket {tagsupport/#1} }
71 }
72 \cs_set_protected:Npn \tag_socket_use:nn #1#2
73 {
74     \bool_if:NT \l__tag_active_socket_bool
75     { \UseSocket {tagsupport/#1} {#2} }
76 }
77 \cs_set_protected:Npn \UseTaggingSocket #1
78 {
79     \bool_if:NTF \l__tag_active_socket_bool
80     { \UseSocket{tagsupport/#1} }
81     {
82         \int_case:nnF
83         { \int_use:c { c__socket_tagsupport/#1_args_int } }
84         {
85             0 \prg_do_nothing:
86             1 \use_none:n
87             2 \use_none:nn

```

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

```
88     }
89     \ERRORusetaggingsocket
90 }
91 }
92 </package>
```

*(End of definition for `\tag_socket_use:n`, `\tag_socket_use:nn`, and `\UseTaggingSocket`. These functions are documented on page 39.)*

## 12 Debugging

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

```
93 <*package>
94 \NewDocumentCommand\ShowTagging { m }
95 {
```

```

96   \keys_set:nn { __tag / show }{ #1}
97
98   }

```

(End of definition for `\ShowTagging`. This function is documented on page 37.)

**mc-data<sub>␣</sub>(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.

```

99 \keys_define:nn { __tag / show }
100 {
101   mc-data .code:n =
102   {
103     \sys_if_engine luatex:T
104     {
105       \lua_now:e{ltx.__tag.trace.show_all_mc_data(#1,\__tag_get_mc_abs_cnt:,0)}
106     }
107   }
108   ,mc-data .default:n = 1
109 }
110

```

(End of definition for `mc-data (show-key)`. This function is documented on page 37.)

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

```

111 \keys_define:nn { __tag / show }
112 { mc-current .code:n =
113   {
114     \bool_if:NTF \g__tag_mode_lua_bool
115     {
116       \sys_if_engine luatex:T
117       {
118         \int_compare:nNnTF
119         { -2147483647 }
120         =
121         {
122           \lua_now:e
123           {
124             tex.print
125             (tex.getattribute
126             (luatexbase.attributes.g__tag_mc_cnt_attr))
127           }
128         }
129         {
130           \lua_now:e
131           {
132             ltx.__tag.trace.log
133             (
134               "mc-current:~no~MC~open,~current~absent
135               =\__tag_get_mc_abs_cnt:"
136               ,0
137             )
138             texio.write_nl("")
139           }

```

```

140     }
141     {
142     \lua_now:e
143     {
144     ltx.__tag.trace.log
145     (
146     "mc-current:~absent=~\__tag_get_mc_abs_cnt:=="
147     ..
148     tex.getattribute(luatexbase.attributes.g__tag_mc_cnt_attr)
149     ..
150     "~=>tag="
151     ..
152     tostring
153     (ltx.__tag.func.get_tag_from
154     (tex.getattribute
155     (luatexbase.attributes.g__tag_mc_type_attr)))
156     ..
157     "="
158     ..
159     tex.getattribute
160     (luatexbase.attributes.g__tag_mc_type_attr)
161     ,0
162     )
163     texio.write_nl("")
164     }
165     }
166   }
167 }
168 {
169 \msg_note:nn{ tag }{ mc-current }
170 }
171 }
172 }

```

(End of definition for mc-current (show-key). This function is documented on page 37.)

**mc-marks<sub>l</sub>(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).

```

173 \keys_define:nn { __tag / show }
174 {
175   mc-marks .choice: ,
176   mc-marks / show .code:n =
177   {
178     \__tag_mc_get_marks:
179     \__tag_check_if_mc_in_galley:TF
180     {
181       \iow_term:n {Marks~from~this~page:~}
182     }
183     {
184       \iow_term:n {Marks~from~a~previous~page:~}
185     }
186     \seq_show:N \l__tag_mc_firstmarks_seq
187     \seq_show:N \l__tag_mc_botmarks_seq
188     \__tag_check_if_mc_tmb_missing:T

```

```

189     {
190       \iow_term:n {BDC-missing-on-this-page!}
191     }
192     \__tag_check_if_mc_tme_missing:T
193     {
194       \iow_term:n {EMC-missing-on-this-page!}
195     }
196   },
197   mc-marks / use .code:n =
198   {
199     \__tag_mc_get_marks:
200     \__tag_check_if_mc_in_galley:TF
201     { Marks-from-this-page:~}
202     { Marks-from-a-previous-page:~}
203     \seq_use:Nn \l__tag_mc_firstmarks_seq {,~}\quad
204     \seq_use:Nn \l__tag_mc_botmarks_seq {,~}\quad
205     \__tag_check_if_mc_tmb_missing:T
206     {
207       BDC-missing~
208     }
209     \__tag_check_if_mc_tme_missing:T
210     {
211       EMC-missing
212     }
213   },
214   mc-marks .default:n = show
215 }

```

(End of definition for `mc-marks` (`show-key`). This function is documented on page 37.)

#### `struct-stack_`(`show-key`)

```

216 \keys_define:nn { __tag / show }
217 {
218   struct-stack .choice:
219   ,struct-stack / log .code:n = \seq_log:N \g__tag_struct_tag_stack_seq
220   ,struct-stack / show .code:n = \seq_show:N \g__tag_struct_tag_stack_seq
221   ,struct-stack .default:n = show
222 }
223 </package>

```

(End of definition for `struct-stack` (`show-key`). This function is documented on page 37.)

#### `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.

```

224 <*debug>
225 \keys_define:nn { __tag / show }
226 {
227   ,debug/structures .code:n =
228   {
229     \int_step_inline:nnn{#1}{\c@g__tag_struct_abs_int}
230     {
231       \msg_term:nneeee
232       { tag/debug } { show-struct }
233       { ##1 }

```

```

234         {
235             \prop_map_function:cN
236             {g__tag_struct_debug_##1_prop}
237             \msg_show_item_unbraced:nn
238         }
239         { } { }
240     \msg_term:nneeee
241     { tag/debug } { show-kids }
242     { ##1 }
243     {
244         \seq_map_function:cN
245         {g__tag_struct_debug_kids_##1_seq}
246         \msg_show_item_unbraced:n
247     }
248     { } { }
249 }
250 }
251 ,debug/structures .default:n = 1
252 }
253 </debug>

```

(End of definition for `debug/structures (show-key)`. This function is documented on page 37.)

## 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.

```
254 <*package>
```

### 13.1 Document structure

```

\g__tag_root_default_tl
  activate_␣(setup-key)
activate/socket_␣(setup-key)
255 \tl_new:N\g__tag_root_default_tl
256 \tl_gset:Nn\g__tag_root_default_tl {Document}
257
258 \hook_gput_code:nnn{begindocument}{tagpdf}{\tagstructbegin{tag=\g__tag_root_default_tl}}
259 \hook_gput_code:nnn{tagpdf/finish/before}{tagpdf}{\tagstructend}
260
261 \keys_define:nn { __tag / setup}
262 {
263     activate/socket .bool_set:N = \l__tag_active_socket_bool,
264     activate .code:n =
265     {
266         \keys_set:nn { __tag / setup }
267         { activate/mc,activate/tree,activate/struct,activate/socket }
268         \tl_gset:Nn\g__tag_root_default_tl {#1}
269     },
270     activate .default:n = Document
271 }
272

```

(End of definition for `\g__tag_root_default_tl`, `activate (setup-key)`, and `activate/socket (setup-key)`. These functions are documented on page 36.)

## 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.

```

273 \AddToHook{begindocument/before}
274   {
275     \bool_lazy_and:nnT
276       { \g__tag_active_struct_dest_bool }
277       { \g__tag_active_struct_bool }
278     {
279       \tl_set:Nn \l_pdf_current_structure_destination_tl
280         { {__tag/struct}{\g__tag_struct_stack_current_tl } }
281       \pdf_activate_indexed_structure_destination:
282     }
283   }

```

## 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

```

284 \sys_if_engine_luatex:T
285   {
286     \NewDocumentCommand\pdffakespace { }
287     {
288       \__tag_fakespace:
289     }
290   }
291 \providecommand\pdffakespace{}

```

(End of definition for `\pdffakespace`. This function is documented on page 38.)

## 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.

<pre> \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 </pre>	<p>At first some variables.</p> <pre> 292 &lt;/package&gt; 293 &lt;base&gt;\bool_new:N \l__tag_para_flattened_bool 294 &lt;base&gt;\bool_new:N \l__tag_para_bool 295 &lt;*package&gt; 296 \int_new:N \g__tag_para_begin_int 297 \int_new:N \g__tag_para_end_int 298 \int_new:N \g__tag_para_main_begin_int 299 \int_new:N \g__tag_para_main_end_int </pre>
---	--

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

```
300 \tl_new:N \g__tag_para_main_struct_tl
301 \tl_gset:Nn \g__tag_para_main_struct_tl {1}
302 \tl_new:N \l__tag_para_tag_default_tl
303 \tl_set:Nn \l__tag_para_tag_default_tl { text }
304 \tl_new:N \l__tag_para_tag_tl
305 \tl_set:Nn \l__tag_para_tag_tl { \l__tag_para_tag_default_tl }
306 \tl_new:N \l__tag_para_main_tag_tl
307 \tl_set:Nn \l__tag_para_main_tag_tl {text-unit}
```

this is perhaps already defined by the block code

```
308 \tl_if_exist:NF \l__tag_para_attr_class_tl
309 {\tl_new:N \l__tag_para_attr_class_tl }
310 \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:
311 \cs_new_protected:Npn \__tag_gincr_para_main_begin_int:
312 {
313   \int_gincr:N \g__tag_para_main_begin_int
314 }
315 \cs_new_protected:Npn \__tag_gincr_para_begin_int:
316 {
317   \int_gincr:N \g__tag_para_begin_int
318 }
319 \cs_new_protected:Npn \__tag_gincr_para_main_end_int:
320 {
321   \int_gincr:N \g__tag_para_main_end_int
322 }
323 \cs_new_protected:Npn \__tag_gincr_para_end_int:
324 {
325   \int_gincr:N \g__tag_para_end_int
326 }
```

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

```
\__tag_start_para_ints:
\__tag_stop_para_ints:
327 \cs_new_protected:Npn \__tag_start_para_ints:
328 {
329   \cs_set_protected:Npn \__tag_gincr_para_main_begin_int:
330   {
331     \int_gincr:N \g__tag_para_main_begin_int
332   }
333   \cs_set_protected:Npn \__tag_gincr_para_begin_int:
334   {
335     \int_gincr:N \g__tag_para_begin_int
336   }
337   \cs_set_protected:Npn \__tag_gincr_para_main_end_int:
338   {
339     \int_gincr:N \g__tag_para_main_end_int
340   }
341   \cs_set_protected:Npn \__tag_gincr_para_end_int:
```



```

342   {
343     \int_gincr:N \g__tag_para_end_int
344   }
345 }
346 \cs_new_protected:Npn \__tag_stop_para_ints:
347 {
348   \cs_set_eq:NN \__tag_gincr_para_main_begin_int:\prg_do_nothing:
349   \cs_set_eq:NN \__tag_gincr_para_begin_int: \prg_do_nothing:
350   \cs_set_eq:NN \__tag_gincr_para_main_end_int: \prg_do_nothing:
351   \cs_set_eq:NN \__tag_gincr_para_end_int: \prg_do_nothing:
352 }

```

(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:`

```

353 \cs_new:Npn \__tag_para_main_store_struct:
354 {
355   \tl_gset:Ne \g__tag_para_main_struct_tl {\int_use:N \c@g__tag_struct_abs_int }
356 }

```

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

TEMPORARY FIX (2023-11-17). Until latex-lab is updated we must adapt a sec command:

```

357 \AddToHook{package/latex-lab-testphase-sec/after}
358 {
359   \cs_set_protected:Npn \@kernel@tag@hangfrom #1
360   {
361     \tagstructbegin{tag=\l__tag_para_tag_tl}
362     \__tag_gincr_para_begin_int:
363     \tagstructbegin{tag=Lbl}
364     \setbox\@tempboxa
365     \hbox
366     {
367       \bool_lazy_and:nnT
368       {\tag_if_active_p:}
369       {\g__tag_mode_lua_bool}
370       {\tagmcbegin{tag=Lbl}}
371       {#1}
372     }
373     \tag_stop:n{hangfrom}
374     \hangindent \wd\@tempboxa\noindent
375     \tag_start:n{hangfrom}
376     \tagmcbegin{\box\@tempboxa\tagmcbegin\tagmcbegin{}}
377   }
378 }

```

and one temporary adaptations for the block module:

```

379 \AddToHook{package/latex-lab-testphase-block/after}
380 {
381   \tl_if_exist:NT \l__tag_para_attr_class_tl
382   {
383     \tl_set:Nn \l__tag_para_attr_class_tl { \l__tag_para_attr_class_tl }
384   }

```

```
385 }
386
```

`para/tagging␣(setup-key)` These keys enable/disable locally paratagging. Paragraphs are typically tagged with two structure: A main structure around the whole paragraph, and inner structures around the various chunks. Debugging can be activated locally with `debug/show=para`, this can affect the typesetting as the small numbers are boxes and they have a (small) height. `para/tag␣(setup-key)` 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`

```
387 \keys_define:nn { __tag / setup }
388 {
para-flattened␣(deprecated)
  para/tagging      .bool_set:N = \l__tag_para_bool,
389 debug/show/para  .code:n = {\bool_set_true:N \l__tag_para_show_bool},
para-tagging␣(deprecated)
390 debug/show/paraOff .code:n = {\bool_set_false:N \l__tag_para_show_bool},
  para/tagging-show␣(deprecated)
391 debug/show/paraOff .code:n = {\bool_set_false:N \l__tag_para_show_bool},
  paratagging␣(deprecated)
392 para/tag          .tl_set:N = \l__tag_para_tag_tl,
  paratagging␣(deprecated)
393 para/maintag      .tl_set:N = \l__tag_para_main_tag_tl,
  paratagging␣(deprecated)
394 para/flattened    .bool_set:N = \l__tag_para_flattened_bool
395 }
396 \keys_define:nn { tag / tool}
397 {
  para/tagging      .bool_set:N = \l__tag_para_bool,
398 para/tag          .tl_set:N = \l__tag_para_tag_tl,
  para/maintag      .tl_set:N = \l__tag_para_main_tag_tl,
400 para/flattened    .bool_set:N = \l__tag_para_flattened_bool
401 }
402 }
```

the deprecated names

```
403 \keys_define:nn { __tag / setup }
404 {
405   paratagging      .bool_set:N = \l__tag_para_bool,
406   paratagging-show .bool_set:N = \l__tag_para_show_bool,
407   paratag          .tl_set:N = \l__tag_para_tag_tl
408 }
409 \keys_define:nn { tag / tool}
410 {
411   para      .bool_set:N = \l__tag_para_bool,
412   paratag .tl_set:N = \l__tag_para_tag_tl,
413   unittag .tl_set:N = \l__tag_para_main_tag_tl,
414   para-flattened .bool_set:N = \l__tag_para_flattened_bool
415 }
```

*(End of definition for `para/tagging` (setup-key) and others. These functions are documented on page 38.)*

Helper command for debugging:

```
416 \cs_new_protected:Npn \__tag_check_para_begin_show:nn #1 #2
417   %#1 color, #2 prefix
418   {
419     \bool_if:NT \l__tag_para_show_bool
420     {
421       \tag_mc_begin:n{artifact}
422       \llap{\color_select:n{#1}\tiny#2\int_use:N\g__tag_para_begin_int\ }
423       \tag_mc_end:
424     }
```

```

425 }
426
427 \cs_new_protected:Npn \__tag_check_para_end_show:nn #1 #2
428 %#1 color, #2 prefix
429 {
430   \bool_if:NT \l__tag_para_show_bool
431   {
432     \tag_mc_begin:n{artifact}
433     \rlap{\color_select:n{#1}\tiny\ #2\int_use:N\g__tag_para_end_int}
434     \tag_mc_end:
435   }
436 }

```

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

```

437 \socket_new:nn      {tagsupport/para/begin}{0}
438 \socket_new:nn      {tagsupport/para/end}{0}
439
440 \socket_new_plug:nnn{tagsupport/para/begin}{plain}
441 {
442   \bool_if:NT \l__tag_para_bool
443   {
444     \bool_if:NF \l__tag_para_flattened_bool
445     {
446       \__tag_gincr_para_main_begin_int:
447       \tag_struct_begin:n
448       {
449         tag=\l__tag_para_main_tag_tl,
450       }
451       \__tag_para_main_store_struct:
452     }
453     \__tag_gincr_para_begin_int:
454     \tag_struct_begin:n {tag=\l__tag_para_tag_tl}
455     \__tag_check_para_begin_show:nn {green}{}
456     \tag_mc_begin:n {}
457   }
458 }
459 \socket_new_plug:nnn{tagsupport/para/begin}{block}
460 {
461   \bool_if:NT \l__tag_para_bool
462   {
463     \legacy_if:nF { @inlabel }
464     {
465       \__tag_check_typeout_v:n
466       {==>~ @endpe = \legacy_if:nTF { @endpe }{true}{false} \on@line }
467     \legacy_if:nF { @endpe }
468     {
469       \bool_if:NF \l__tag_para_flattened_bool
470       {
471         \__tag_gincr_para_main_begin_int:
472         \tag_struct_begin:n

```

```

473         {
474             tag=\l__tag_para_main_tag_tl,
475             attribute-class=\l__tag_para_main_attr_class_tl,
476         }
477         \__tag_para_main_store_struct:
478     }
479 }
480 \__tag_gincr_para_begin_int:
481 \__tag_check_typeout_v:n {==>~increment~ P \on@line }
482 \tag_struct_begin:n
483 {
484     tag=\l__tag_para_tag_tl
485     ,attribute-class=\l__tag_para_attr_class_tl
486 }
487 \__tag_check_para_begin_show:nn {green}{\PARALABEL}
488 \tag_mc_begin:n {}
489 }
490 }
491 }

```

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

```

492 \socket_new_plug:nnn{tagsupport/para/end}{plain}
493 {
494     \bool_if:NT \l__tag_para_bool
495     {
496         \__tag_gincr_para_end_int:
497         \__tag_check_typeout_v:n {==>~increment~ /P \on@line }
498         \tag_mc_end:
499         \__tag_check_para_end_show:nn {red}{}
500         \tag_struct_end:
501         \bool_if:NF \l__tag_para_flattened_bool
502         {
503             \__tag_gincr_para_main_end_int:
504             \tag_struct_end:
505         }
506     }
507 }

```

By default we assign the plain plug:

```

508 \socket_assign_plug:nn { tagsupport/para/begin}{plain}
509 \socket_assign_plug:nn { tagsupport/para/end}{plain}

```

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

```

510 \AddToHook{para/begin}{ \socket_use:n { tagsupport/para/begin }
511 }
512 \AddToHook{para/end} { \socket_use:n { tagsupport/para/end } }

```

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

```

513 \AddToHook{package/latex-lab-testphase-block/after}
514 {
515     \RemoveFromHook{para/begin}[tagpdf]
516     \RemoveFromHook{para/end}[latex-lab-testphase-block]

```

```

517 \AddToHook{para/begin}[tagpdf]
518 {
519   \socket_use:n { tagsupport/para/begin }
520 }
521 \AddToHook{para/end}[tagpdf]
522 {
523   \socket_use:n { tagsupport/para/end }
524 }
525 \socket_assign_plug:nn { tagsupport/para/begin}{block}
526 }
527

```

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

```

528 \AddToHook{enddocument/info}
529 {
530   \tag_if_active:F
531   {
532     \msg_redirect_name:nnn { tag } { para-hook-count-wrong } { warning }
533   }
534   \int_compare:nNnF {\g__tag_para_main_begin_int}={\g__tag_para_main_end_int}
535   {
536     \msg_error:nneee
537     {tag}
538     {para-hook-count-wrong}
539     {\int_use:N\g__tag_para_main_begin_int}
540     {\int_use:N\g__tag_para_main_end_int}
541     {text-unit}
542   }
543   \int_compare:nNnF {\g__tag_para_begin_int}={\g__tag_para_end_int}
544   {
545     \msg_error:nneee
546     {tag}
547     {para-hook-count-wrong}
548     {\int_use:N\g__tag_para_begin_int}
549     {\int_use:N\g__tag_para_end_int}
550     {text}
551   }
552 }

```

We need at least the new-or-1 code. In generic mode we also must insert the code to finish the MC-chunks

```

553 \ifpackageloaded{footmisc}
554 {\PackageWarning{tagpdf}{tagpdf~has~been~loaded~too~late!}} %
555 {\RequirePackage{latex-lab-testphase-new-or-1}}
556
557 \AddToHook{begindocument/before}
558 {
559   \providecommand\@kernel@tagsupport@@makecol{}
560   \providecommand\@kernel@before@cclv{}
561   \bool_if:NF \g__tag_mode_lua_bool
562   {
563     \cs_if_exist:NT \@kernel@before@footins
564     {

```

```

565     \tl_put_right:Nn \@kernel@before@footins
566       { \__tag_add_missing_mcs_to_stream:Nn \footins {footnote} }
567   \tl_put_right:Nn \@kernel@before@ccclv
568     {
569       \__tag_check_typeout_v:n {====>~In~\token_to_str:N \@makecol\c_space_tl\the\c@
570       \__tag_add_missing_mcs_to_stream:Nn \@ccclv {main}
571     }
572   \tl_put_right:Nn \@kernel@tagsupport@makecol
573     {
574       \__tag_check_typeout_v:n {====>~In~\token_to_str:N \@makecol\c_space_tl\the\c@
575       \__tag_add_missing_mcs_to_stream:Nn \@outputbox {main}
576     }
577   \tl_put_right:Nn \@mult@ptagging@hook
578     {
579       \__tag_check_typeout_v:n {====>~In~\string\page@sofar}
580       \process@cols\mult@firstbox
581       {
582         \__tag_add_missing_mcs_to_stream:Nn \count@ {multicol}
583       }
584       \__tag_add_missing_mcs_to_stream:Nn \mult@rightbox {multicol}
585     }
586   }
587 }
588 }
589 \endpackage

```

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

`\tagpdfparaOff`

```

590 \newcommand\tagpdfparaOn {}
591 \newcommand\tagpdfparaOff{}
592 \endpackage
593 \renewcommand\tagpdfparaOn {\bool_set_true:N \l__tag_para_bool}
594 \renewcommand\tagpdfparaOff{\bool_set_false:N \l__tag_para_bool}

```

*(End of definition for `\tagpdfparaOn` and `\tagpdfparaOff`. These functions are documented on page 38.)*

`\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
}
{\tagpdfsuppressmarks{\tagmccend}\tagstructend}%

595 \NewDocumentCommand\tagpdfsuppressmarks{m}
596   {{\use:c{__tag_mc_disable_marks:} #1}}

```

*(End of definition for `\tagpdfsuppressmarks`. This function is documented on page 38.)*

## 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<sub>□</sub>(setup-key)

```
597 \keys_define:nn { __tag / setup }
598   {
599     text / lang .tl_set:N = \l__tag_struct_lang_tl
600   }
```

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

## 13.6 Header and footer

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

```
601 \cs_new_protected:Npn \__tag_hook_kernel_before_head: {}
602 \cs_new_protected:Npn \__tag_hook_kernel_after_head: {}
603 \cs_new_protected:Npn \__tag_hook_kernel_before_foot: {}
604 \cs_new_protected:Npn \__tag_hook_kernel_after_foot: {}
605
606 \AddToHook{begindocument}
607   {
608     \cs_if_exist:NT \@kernel@before@head
609     {
610       \tl_put_right:Nn \@kernel@before@head {\__tag_hook_kernel_before_head:}
611       \tl_put_left:Nn \@kernel@after@head {\__tag_hook_kernel_after_head:}
612       \tl_put_right:Nn \@kernel@before@foot {\__tag_hook_kernel_before_foot:}
613       \tl_put_left:Nn \@kernel@after@foot {\__tag_hook_kernel_after_foot:}
614     }
615   }
616
617 \bool_new:N \g__tag_saved_in_mc_bool
618 \cs_new_protected:Npn \__tag_exclude_headfoot_begin:
619   {
620     \bool_set_false:N \l__tag_para_bool
621     \bool_if:NTF \g__tag_mode_lua_bool
622     {
623       \tag_mc_end_push:
624     }
625     {
626       \bool_gset_eq:NN \g__tag_saved_in_mc_bool \g__tag_in_mc_bool
627       \bool_gset_false:N \g__tag_in_mc_bool
628     }
629     \tag_mc_begin:n {artifact}
630     \tag_stop:n{headfoot}
631   }
632 \cs_new_protected:Npn \__tag_exclude_headfoot_end:
633   {
634     \tag_start:n{headfoot}
635     \tag_mc_end:
636     \bool_if:NTF \g__tag_mode_lua_bool
```

```

637     {
638       \tag_mc_begin_pop:n{ }
639     }
640     {
641       \bool_gset_eq:NN \g__tag_in_mc_bool\g__tag_saved_in_mc_bool
642     }
643   }

```

This version allows to use an Artifact structure

```

644 \__tag_attr_new_entry:nn {__tag/attr/pagination}{/0/Artifact/Type/Pagination}
645 \cs_new_protected:Npn \__tag_exclude_struct_headfoot_begin:n #1
646 {
647   \bool_set_false:N \l__tag_para_bool
648   \bool_if:NTF \g__tag_mode_lua_bool
649   {
650     \tag_mc_end_push:
651   }
652   {
653     \bool_gset_eq:NN \g__tag_saved_in_mc_bool \g__tag_in_mc_bool
654     \bool_gset_false:N \g__tag_in_mc_bool
655   }
656   \tag_struct_begin:n{tag=Artifact,attribute-class=__tag/attr/#1}
657   \tag_mc_begin:n {artifact=#1}
658   \tag_stop:n{headfoot}
659 }
660
661 \cs_new_protected:Npn \__tag_exclude_struct_headfoot_end:
662 {
663   \tag_start:n{headfoot}
664   \tag_mc_end:
665   \tag_struct_end:
666   \bool_if:NTF \g__tag_mode_lua_bool
667   {
668     \tag_mc_begin_pop:n{ }
669   }
670   {
671     \bool_gset_eq:NN \g__tag_in_mc_bool\g__tag_saved_in_mc_bool
672   }
673 }

```

And now the keys

[page/exclude-header-footer<sub>□</sub>\(setup-key\)](#)

[exclude-header-footer<sub>□</sub>\(deprecated\)](#)

```

674 \keys_define:nn { __tag / setup }
675 {
676   page/exclude-header-footer .choice:,
677   page/exclude-header-footer / true .code:n =
678   {
679     \cs_set_eq:NN \__tag_hook_kernel_before_head: \__tag_exclude_headfoot_begin:
680     \cs_set_eq:NN \__tag_hook_kernel_before_foot: \__tag_exclude_headfoot_begin:
681     \cs_set_eq:NN \__tag_hook_kernel_after_head: \__tag_exclude_headfoot_end:
682     \cs_set_eq:NN \__tag_hook_kernel_after_foot: \__tag_exclude_headfoot_end:
683   },
684   page/exclude-header-footer / pagination .code:n =
685   {

```



```

686     \cs_set:Nn \__tag_hook_kernel_before_head: { \__tag_exclude_struct_headfoot_begin:n {p
687     \cs_set:Nn \__tag_hook_kernel_before_foot: { \__tag_exclude_struct_headfoot_begin:n {p
688     \cs_set_eq:NN \__tag_hook_kernel_after_head: \__tag_exclude_struct_headfoot_end:
689     \cs_set_eq:NN \__tag_hook_kernel_after_foot: \__tag_exclude_struct_headfoot_end:
690   },
691   page/exclude-header-footer / false .code:n =
692   {
693     \cs_set_eq:NN \__tag_hook_kernel_before_head: \prg_do_nothing:
694     \cs_set_eq:NN \__tag_hook_kernel_before_foot: \prg_do_nothing:
695     \cs_set_eq:NN \__tag_hook_kernel_after_head: \prg_do_nothing:
696     \cs_set_eq:NN \__tag_hook_kernel_after_foot: \prg_do_nothing:
697   },
698   page/exclude-header-footer .default:n = true,
699   page/exclude-header-footer .initial:n = true,

```

deprecated name

```

700   exclude-header-footer .meta:n = { page/exclude-header-footer = {#1} }
701 }

```

*(End of definition for page/exclude-header-footer (setup-key) and exclude-header-footer (deprecated). These functions are documented on page 39.)*

## 13.7 Links

We need to close and reopen mc-chunks around links. Currently we handle URI and GoTo (internal) links. Links should have an alternative text in the Contents key. It is unclear which text this should be and how to get it.

```

702 \hook_gput_code:nnn
703   {pdfannot/link/URI/before}
704   {tagpdf}
705   {
706     \tag_mc_end_push:
707     \tag_struct_begin:n { tag=Link }
708     \tag_mc_begin:n { tag=Link }
709     \pdfannot_dict_put:nne
710       { link/URI }
711       { StructParent }
712     { \tag_struct_parent_int: }
713   }
714
715 \hook_gput_code:nnn
716   {pdfannot/link/URI/after}
717   {tagpdf}
718   {
719     \tag_struct_insert_annot:ee {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
720     \tag_mc_end:
721     \tag_struct_end:
722     \tag_mc_begin_pop:n{ }
723   }
724
725 \hook_gput_code:nnn
726   {pdfannot/link/GoTo/before}
727   {tagpdf}
728   {

```

```

729     \tag_mc_end_push:
730     \tag_struct_begin:n{tag=Link}
731     \tag_mc_begin:n{tag=Link}
732     \pdfannot_dict_put:nne
733     { link/GoTo }
734     { StructParent }
735     { \tag_struct_parent_int: }
736   }
737
738   \hook_gput_code:nnn
739   {pdfannot/link/GoTo/after}
740   {tagpdf}
741   {
742     \tag_struct_insert_annot:ee {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
743     \tag_mc_end:
744     \tag_struct_end:
745     \tag_mc_begin_pop:n{ }
746   }
747 }
748
749 % "alternative descriptions " for PAX3. How to get better text here??
750 \pdfannot_dict_put:nnn
751 { link/URI }
752 { Contents }
753 { (url) }
754
755 \pdfannot_dict_put:nnn
756 { link/GoTo }
757 { Contents }
758 { (ref) }
759
</package>

```

## Part III

# The tagpdf-tree module

## Commands trees and main dictionaries

### Part of the tagpdf package

```
1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-tree-code} {2024-09-11} {0.99e}
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   \msg_note:nn {tag}{tree-statistic}
29 }
```

(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
30 \pdf_object_new_indexed:nm { __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).
31 \tl_new:N \g__tag_tree_openaction_struct_tl
32 \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.
33 \keys_define:nm { __tag / setup }
34 {
35   viewer/startstructure .code:n =
36   {
37     \tl_gset:Ne \g__tag_tree_openaction_struct_tl {#1}
38   }
39   ,viewer/startstructure .default:n = { \int_use:N \c@g__tag_struct_abs_int }
40 }

```

(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:

```

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

```

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.

```

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

```

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

```

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

```

### 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

```

77 \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
78 \cs_generate_variant:Nn \tl_count:n {e}
79 \hook_gput_code:nnn{begindocument}{tagpdf}
80 {
81     \int_gset:Nn\g__tag_tree_id_pad_int
82     {\tl_count:e { \__tag_property_ref_lastpage:nn{tagstruct}{1000}}+1}
83 }
84

```

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.

```

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

```

```

94     {
95       \__tag_struct_get_id:n{##1}~\pdf_object_ref_indexed:nn {__tag/struct}{##1}~
96     }
97   \int_compare:nNnF {\l__tag_tmpa_int}<{50} %
98   {
99     \pdf_object_unnamed_write:ne {dict}
100     { /Limits~[\__tag_struct_get_id:n{##1}~\l__tag_tmpa_int+1}~\__tag_struct_get_id:
101       /Names~[\l__tag_tmpa_tl]
102     }
103     \tl_put_right:Ne\l__tag_tmpb_tl {\pdf_object_ref_last:\c_space_tl}
104     \int_zero:N \l__tag_tmpa_int
105     \tl_clear:N \l__tag_tmpa_tl
106   }
107 }
108 \tl_if_empty:NF \l__tag_tmpa_tl
109 {
110   \pdf_object_unnamed_write:ne {dict}
111   {
112     /Limits~
113     [\__tag_struct_get_id:n{\c@g__tag_struct_abs_int~\l__tag_tmpa_int+1}~
114     \__tag_struct_get_id:n{\c@g__tag_struct_abs_int}]
115     /Names~[\l__tag_tmpa_tl]
116   }
117   \tl_put_right:Ne\l__tag_tmpb_tl {\pdf_object_ref_last:}
118 }
119 \pdf_object_unnamed_write:ne {dict}{/Kids~[\l__tag_tmpb_tl]}
120 \__tag_prop_gput:cne
121   { g__tag_struct_1_prop }
122   { IDTree }
123   { \pdf_object_ref_last: }
124 }

```

## 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.

```

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

```

```

142         \l__tag_tmpa_tl
143     }

```

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

```

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

```

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

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

```

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

```

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

## 1.5 ParentTree

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

```

153 \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`.

```

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

```

(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`

```

161 \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.

```

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

```

(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.

```

169 \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.

```

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

```



```

204         \pdf_object_ref_indexed:nn { __tag/struct }{ \l__tag_tmpa_tl }
205     }
206     {
207         null
208     }
209     \c_space_tl
210 }
211 }
212 {
213     \cs_set_protected:Npn \__tag_tree_parenttree_rerun_msg:
214     {
215         \msg_warning:nn { tag } {tree-mcid-index-wrong}
216     }
217 }
218 }
219 \tl_put_right:Nn
220 \l__tag_parenttree_content_tl
221 {%[
222 ]^^J
223 }
224 }
225 }

```

(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.

```

226 \cs_new_protected:Npn \__tag_tree_lua_fill_parenttree:
227 {
228     \tl_set:Nn \l__tag_parenttree_content_tl
229     {
230         \lua_now:e
231         {
232             ltx.__tag.func.output_parenttree
233             (
234                 \int_use:N\g_shipout_readonly_int
235             )
236         }
237     }
238 }

```

(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?

```

239 \cs_new_protected:Npn \__tag_tree_write_parenttree:
240 {
241     \bool_if:NTF \g__tag_mode_lua_bool
242     {
243         \__tag_tree_lua_fill_parenttree:
244     }
245     {
246         \__tag_tree_fill_parenttree:
247     }
248     \__tag_tree_parenttree_rerun_msg:

```

```

249 \tl_put_right:NV \l__tag_parenttree_content_tl\g__tag_parenttree_objr_tl
250 \pdf_object_write:nne { __tag/tree/parenttree }{dict}
251 {
252   /Nums\c_space_tl [\l__tag_parenttree_content_tl]
253 }
254 }

```

(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.

```

255 \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.

```

256 \cs_new_protected:Npn \__tag_tree_write_rolemap:
257 {
258   \bool_if:NT \g__tag_role_add_mathml_bool
259   {
260     \prop_map_inline:Nn \g__tag_role_NS_mathml_prop
261     {
262       \prop_gput:Nnn \g__tag_role_rolemap_prop {##1}{Span}
263     }
264   }
265   \prop_map_inline:Nn\g__tag_role_rolemap_prop
266   {
267     \tl_if_eq:nnF {##1}{##2}
268     {
269       \pdfdict_gput:nne {g__tag_role/RoleMap_dict}
270       {##1}
271       {\pdf_name_from_unicode_e:n{##2}}
272     }
273   }
274   \pdf_object_write:nne { __tag/tree/rolemap }{dict}
275   {
276     \pdfdict_use:n{g__tag_role/RoleMap_dict}
277   }
278 }

```

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

## 1.7 Classmap dictionary

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

```
__tag_tree_write_classmap:
```

```
279 \cs_new_protected:Npn __tag_tree_write_classmap:
280 {
281   \tl_clear:N \l__tag_tmpa_tl
282   \seq_map_inline:Nn \g__tag_attr_class_used_seq
283   {
284     \prop_gput:Nnn \g__tag_attr_class_used_prop {##1}{ }
285   }
286   \prop_map_inline:Nn \g__tag_attr_class_used_prop
287   {
288     \tl_put_right:Ne \l__tag_tmpa_tl
289     {
290       ##1\c_space_tl
291       <<
292       \prop_item:Nn
293       \g__tag_attr_entries_prop
294       {##1}
295       >>
296       \iow_newline:
297     }
298   }
299   \tl_if_empty:NF
300   \l__tag_tmpa_tl
301   {
302     \pdf_object_new:n { __tag/tree/classmap }
303     \pdf_object_write:nne
304     { __tag/tree/classmap }
305     {dict}
306     { \l__tag_tmpa_tl }
307     \__tag_prop_gput:cne
308     { g__tag_struct_1_prop }
309     { ClassMap }
310     { \pdf_object_ref:n { __tag/tree/classmap } }
311   }
312 }
```

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

## 1.8 Namespaces

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

```
__tag/tree/namespaces
```

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

(End of definition for \_\_tag/tree/namespaces.)

```
\__tag_tree_write_namespaces:
```

```
314 \cs_new_protected:Npn \__tag_tree_write_namespaces:
315 {
316   \pdf_version_compare:NnF < {2.0}
```

```

317 {
318   \prop_map_inline:Nn \g__tag_role_NS_prop
319   {
320     \pdfdict_if_empty:nF {g__tag_role/RoleMapNS_##1_dict}
321     {
322       \pdf_object_write:nne {__tag/RoleMapNS/##1}{dict}
323       {
324         \pdfdict_use:n {g__tag_role/RoleMapNS_##1_dict}
325       }
326       \pdfdict_gput:nne{g__tag_role/namespace_##1_dict}
327       {RoleMapNS}{\pdf_object_ref:n {__tag/RoleMapNS/##1}}
328     }
329     \pdf_object_write:nne{tag/NS/##1}{dict}
330     {
331       \pdfdict_use:n {g__tag_role/namespace_##1_dict}
332     }
333   }
334   \pdf_object_write:nne {__tag/tree/namespaces}{array}
335   {
336     \prop_map_tokens:Nn \g__tag_role_NS_prop{\use_ii:n}
337   }
338 }
339 }

```

(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:

```

340 \hook_new:n {tagpdf/finish/before}
341 \cs_new_protected:Npn \__tag_finish_structure:
342 {
343   \bool_if:NT\g__tag_active_tree_bool
344   {
345     \hook_use:n {tagpdf/finish/before}
346     \__tag_tree_final_checks:
347     \iow_term:n{Package~tagpdf~Info:~writing~ParentTree}
348     \__tag_check_benchmark_tic:
349     \__tag_tree_write_parenttree:
350     \__tag_check_benchmark_toc:
351     \iow_term:n{Package~tagpdf~Info:~writing~IDTree}
352     \__tag_check_benchmark_tic:
353     \__tag_tree_write_idtree:
354     \__tag_check_benchmark_toc:
355     \iow_term:n{Package~tagpdf~Info:~writing~RoleMap}
356     \__tag_check_benchmark_tic:
357     \__tag_tree_write_rolemap:
358     \__tag_check_benchmark_toc:
359     \iow_term:n{Package~tagpdf~Info:~writing~ClassMap}
360     \__tag_check_benchmark_tic:
361     \__tag_tree_write_classmap:

```

```

362     \__tag_check_benchmark_toc:
363     \iow_term:n{Package~tagpdf~Info:~writing~NameSpaces}
364     \__tag_check_benchmark_tic:
365     \__tag_tree_write_namespaces:
366     \__tag_check_benchmark_toc:
367     \iow_term:n{Package~tagpdf~Info:~writing~StructElems}
368     \__tag_check_benchmark_tic:
369     \__tag_tree_write_structelements: %this is rather slow!!
370     \__tag_check_benchmark_toc:
371     \iow_term:n{Package~tagpdf~Info:~writing~Root}
372     \__tag_check_benchmark_tic:
373     \__tag_tree_write_structtreeroot:
374     \__tag_check_benchmark_toc:
375   }
376 }
377 </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.

```

378 <*package>
379 \hook_gput_code:nnn{begindocument}{tagpdf}
380 {
381   \bool_if:NT\g__tag_active_tree_bool
382   {
383     \hook_gput_code:nnn{shipout/before} { tagpdf/structparents }
384     {
385       \pdfmanagement_add:nne
386       { Page }
387       { StructParents }
388       { \int_eval:n { \g_shipout_readonly_int} }
389     }
390   }
391 }
392 </package>

```

## Part IV

# The `tagpdf-mc-shared` module

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

### Part of the `tagpdf` package

#### 1 Public Commands

---

<code>\tag_mc_begin:n</code>	<code>\tag_mc_begin:n{&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.

---

---

`\tag_mc_reset_box:N`  $\star$  `\tag_mc_reset_box:N`  $\{<box>\}$

---

New: 2023-06-11

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

## 2 Public keys

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

---

`tag□(mc-key)`

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

---

`artifact□(mc-key)`

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

---

`raw□(mc-key)`

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

---

`alt□(mc-key)`

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

---

`actualtext□(mc-key)`

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

---

`label□(mc-key)`

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

---

`stash□(mc-key)`

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

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

### 3 Marked content code – shared

```

1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-mc-code-shared} {2024-09-11} {0.99e}
4   {part of tagpdf - code related to marking chunks -
5     code shared by generic and luamode }
6 </header>

```

#### 3.1 Variables and counters

MC chunks must be counted. I use a latex counter for the absolute count, so that it is added to `\c1@ckpt` and restored e.g. in `tabulars` and `align`. `\int_new:N \c@g_@@_MCID_abs_int` and `\tl_put_right:Nn\c1@ckpt{\@elt{g_@@_MCID_abs_int}}` would work too, but as the name is not `expl3` then too, why bother? The absolute counter can be used to label and to check if the page counter needs a reset.

`g__tag_MCID_abs_int`

```

7 <*base>
8 \newcounter { g__tag_MCID_abs_int }

```

*(End of definition for g\_\_tag\_MCID\_abs\_int.)*

`\__tag_get_data_mc_counter:`

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

```

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

```

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

`\__tag_get_mc_abs_cnt:`

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

```

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

```

*(End of definition for \\_\_tag\_get\_mc\_abs\_cnt:.)*

`\g__tag_in_mc_bool`

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

```

16 \bool_new:N \g__tag_in_mc_bool

```

*(End of definition for \g\_\_tag\_in\_mc\_bool.)*

`\g__tag_mc_parenttree_prop`

For every chunk we need to know the structure it is in, to record this in the parent tree. We store this in a property.

key: absolute number of the mc (tagmcabs)

value: the structure number the mc is in

```

17 \__tag_prop_new_linked:N \g__tag_mc_parenttree_prop

```

*(End of definition for \g\_\_tag\_mc\_parenttree\_prop.)*



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

```
18 \seq_new:N \g__tag_mc_stack_seq
```

(End of definition for `\g__tag_mc_parenttree_prop`.)

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

```
19 \tl_new:N \l__tag_mc_artifact_type_tl
```

(End of definition for `\l__tag_mc_artifact_type_tl`.)

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

```
\l__tag_mc_artifact_bool 20 \bool_new:N \l__tag_mc_key_stash_bool
```

```
21 \bool_new:N \l__tag_mc_artifact_bool
```

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

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

```
\g__tag_mc_key_tag_tl
```

```
\l__tag_mc_key_label_tl 22 \tl_new:N \l__tag_mc_key_tag_tl
```

```
\l__tag_mc_key_properties_tl 23 \tl_new:N \g__tag_mc_key_tag_tl
```

```
24 \tl_new:N \l__tag_mc_key_label_tl
```

```
25 \tl_new:N \l__tag_mc_key_properties_tl
```

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

## 3.2 Functions

`\__tag_mc_handle_mc_label:e` The commands labels a mc-chunk. It is used if the user explicitly labels the mc-chunk with the `label` key. The argument is the value provided by the user. It stores the attributes

**tagabspage:** the absolute page, `\g_shipout_readonly_int`,

**tagmcabs:** the absolute mc-counter `\c@g_@@_MCID_abs_int`. The reference command is based on `l3ref`.

```
26 \cs_new:Npn \__tag_mc_handle_mc_label:e #1
```

```
27 {
```

```
28   \__tag_property_record:en{tagpdf-#1}{tagabspage,tagmcabs}
```

```
29 }
```

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

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

```
30 \cs_new_protected:Npn \__tag_mc_set_label_used:n #1 %#1 labelname
```

```
31 {
```

```
32   \tl_new:c { g__tag_mc_label_\tl_to_str:n{#1}_used_tl }
```

```
33 }
```

```
34 </shared>
```

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

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

TODO: is testing for struct the right test?

```

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

```

(End of definition for `\tag_mc_use:n`. This function is documented on page 70.)

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

```

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

```

(End of definition for `\tag_mc_artifact_group_begin:n` and `\tag_mc_artifact_group_end:.` These functions are documented on page 70.)

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

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

(End of definition for `\tag_mc_reset_box:N`. This function is documented on page 71.)

`\tag_mc_end_push:`  
`\tag_mc_begin_pop:n`

```
79 <base>\cs_new_protected:Npn \tag_mc_end_push: {}
80 <base>\cs_new_protected:Npn \tag_mc_begin_pop:n #1 {}
81 <*shared>
82 \cs_set_protected:Npn \tag_mc_end_push:
83   {
84     \__tag_check_if_active_mc:T
85     {
86       \__tag_mc_if_in:TF
87       {
88         \seq_gpush:Ne \g__tag_mc_stack_seq { \tag_get:n {mc_tag} }
89         \__tag_check_mc_pushed_popped:nn
90           { pushed }
91           { \tag_get:n {mc_tag} }
92         \tag_mc_end:
93       }
94       {
95         \seq_gpush:Nn \g__tag_mc_stack_seq {-1}
96         \__tag_check_mc_pushed_popped:nn { pushed }{-1}
97       }
98     }
99   }
100
101 \cs_set_protected:Npn \tag_mc_begin_pop:n #1
102   {
103     \__tag_check_if_active_mc:T
104     {
105       \seq_gpop:NNTF \g__tag_mc_stack_seq \l__tag_tmpa_tl
106       {
107         \tl_if_eq:NnTF \l__tag_tmpa_tl {-1}
108         {
109           \__tag_check_mc_pushed_popped:nn {popped}{-1}
110         }
111         {
112           \__tag_check_mc_pushed_popped:nn {popped}{\l__tag_tmpa_tl}
113           \tag_mc_begin:n {tag=\l__tag_tmpa_tl,#1}
114         }
115       }
116       {
117         \__tag_check_mc_pushed_popped:nn {popped}{empty~stack,~nothing}
118       }
119     }
120   }
```

(End of definition for `\tag_mc_end_push:` and `\tag_mc_begin_pop:n`. These functions are documented on page 70.)

### 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.

```
121 \keys_define:n { __tag / mc }
122 {
123   stash .bool_set:N = \l__tag_mc_key_stash_bool,
124   __artifact-bool .bool_set:N = \l__tag_mc_artifact_bool,
125   __artifact-type .choice:,
126   __artifact-type / pagination .code:n =
127   {
128     \tl_set:Nn \l__tag_mc_artifact_type_tl { Pagination }
129   },
130   __artifact-type / pagination/header .code:n =
131   {
132     \tl_set:Nn \l__tag_mc_artifact_type_tl { Pagination/Subtype/Header }
133   },
134   __artifact-type / pagination/footer .code:n =
135   {
136     \tl_set:Nn \l__tag_mc_artifact_type_tl { Pagination/Subtype/Footer }
137   },
138   __artifact-type / layout .code:n =
139   {
140     \tl_set:Nn \l__tag_mc_artifact_type_tl { Layout }
141   },
142   __artifact-type / page .code:n =
143   {
144     \tl_set:Nn \l__tag_mc_artifact_type_tl { Page }
145   },
146   __artifact-type / background .code:n =
147   {
148     \tl_set:Nn \l__tag_mc_artifact_type_tl { Background }
149   },
150   __artifact-type / notype .code:n =
151   {
152     \tl_set:Nn \l__tag_mc_artifact_type_tl {}
153   },
154   __artifact-type / .code:n =
155   {
156     \tl_set:Nn \l__tag_mc_artifact_type_tl {}
157   },
158 }
```

(End of definition for `stash (mc-key)`, `__artifact-bool`, and `__artifact-type`. This function is documented on page 71.)

```
159 </shared>
```

## Part V

# The tagpdf-mc-generic module Code related to Marked Content (mc-chunks), generic mode Part of the tagpdf package

## 1 Marked content code – generic mode

```
1 <@@=tag>
2 <*generic>
3 \ProvidesExplPackage {tagpdf-mc-code-generic} {2024-09-11} {0.99e}
4 {part of tagpdf - code related to marking chunks - generic mode}
5 </generic>
6 <*debug>
7 \ProvidesExplPackage {tagpdf-debug-generic} {2024-09-11} {0.99e}
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  
`\g__tag_mc_footnote_marks_seq` previous chunk in the stream. We provide three by default: main, footnote and multicol.  
`\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`.)

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

```

17 \seq_new:N \l__tag_mc_firstmarks_seq
18 \seq_new:N \l__tag_mc_botmarks_seq

```

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

## 1.2 Functions

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

```

19 \cs_new_protected:Npn \__tag_mc_begin_marks:nn #1 #2 %#1 tag, #2 label
20 {
21   \tex_marks:D \g__tag_mc_marks
22   {
23     b-, %first of begin pair
24     \int_use:N\c@g__tag_MCID_abs_int, %mc-num
25     \g__tag_struct_stack_current_tl, %structure num
26     #1, %tag
27     \bool_if:NT \l__tag_mc_key_stash_bool{stash}, % stash info
28     #2, %label
29   }
30   \tex_marks:D \g__tag_mc_marks
31   {
32     b+, % second of begin pair
33     \int_use:N\c@g__tag_MCID_abs_int, %mc-num
34     \g__tag_struct_stack_current_tl, %structure num
35     #1, %tag
36     \bool_if:NT \l__tag_mc_key_stash_bool{stash}, % stash info
37     #2, %label
38   }
39 }
40 \cs_generate_variant:Nn \__tag_mc_begin_marks:nn {oo}
41 \cs_new_protected:Npn \__tag_mc_artifact_begin_marks:n #1 %#1 type
42 {
43   \tex_marks:D \g__tag_mc_marks
44   {
45     b-, %first of begin pair
46     \int_use:N\c@g__tag_MCID_abs_int, %mc-num
47     -1, %structure num
48     #1 %type
49   }
50   \tex_marks:D \g__tag_mc_marks
51   {
52     b+, %first of begin pair
53     \int_use:N\c@g__tag_MCID_abs_int, %mc-num
54     -1, %structure num
55     #1 %Type
56   }
57 }

```

```

58
59 \cs_new_protected:Npn \__tag_mc_end_marks:
60 {
61   \tex_marks:D \g__tag_mc_marks
62   {
63     e-, %first of end pair
64     \int_use:N\c@g__tag_MCID_abs_int, %mc-num
65     \g__tag_struct_stack_current_tl, %structure num
66   }
67   \tex_marks:D \g__tag_mc_marks
68   {
69     e+, %second of end pair
70     \int_use:N\c@g__tag_MCID_abs_int, %mc-num
71     \g__tag_struct_stack_current_tl, %structure num
72   }
73 }

```

(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.

```

74 \cs_new_protected:Npn \__tag_mc_disable_marks:
75 {
76   \cs_set_eq:NN \__tag_mc_begin_marks:nn \use_none:nn
77   \cs_set_eq:NN \__tag_mc_artifact_begin_marks:n \use_none:n
78   \cs_set_eq:NN \__tag_mc_end_marks: \prg_do_nothing:
79 }

```

(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.

```

80 \cs_new_protected:Npn \__tag_mc_get_marks:
81 {
82   \exp_args:NNe
83   \seq_set_from_clist:Nn \l__tag_mc_firstmarks_seq
84   { \tex_firstmarks:D \g__tag_mc_marks }
85   \exp_args:NNe
86   \seq_set_from_clist:Nn \l__tag_mc_botmarks_seq
87   { \tex_botmarks:D \g__tag_mc_marks }
88 }

```

(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.

```

89 \cs_new_protected:Npn \__tag_mc_store:nnn #1 #2 #3 %#1 mc-prev, #2 mc-num #3 structure-
90   num
91   {
92     \%prop_show:N \g__tag_struct_cont_mc_prop
93     \prop_get:NnNTF \g__tag_struct_cont_mc_prop {#1} \l__tag_tmpa_tl
94     {

```

```

94     \prop_gput:Nne \g__tag_struct_cont_mc_prop {#1}{ \l__tag_tmpa_tl \__tag_struct_mcid_c
95   }
96   {
97     \prop_gput:Nne \g__tag_struct_cont_mc_prop {#1}{ \__tag_struct_mcid_dict:n {#2}}
98   }
99   \prop_gput:Nee \g__tag_mc_parenttree_prop
100   {#2}
101   {#3}
102 }
103 \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.

```

104 \cs_new_protected:Npn \__tag_mc_insert_extra_tmb:n #1 % #1 stream: e.g. main or footnote
105   {
106     \__tag_check_typeout_v:n {=>~ first~ \seq_use:Nn \l__tag_mc_firstmarks_seq {,~}}
107     \__tag_check_typeout_v:n {=>~ bot~ \seq_use:Nn \l__tag_mc_botmarks_seq {,~}}
108     \__tag_check_if_mc_tmb_missing:TF
109     {
110       \__tag_check_typeout_v:n {=>~ TMB~ ~ missing~ --- inserted}
111       %test if artifact
112       \int_compare:nNnTF { \seq_item:cn { g__tag_mc_#1_marks_seq } {3} } = {-
113         1}
114       {
115         \tl_set:Nc \l__tag_tmpa_tl { \seq_item:cn { g__tag_mc_#1_marks_seq } {4} }
116         \__tag_mc_handle_artifact:N \l__tag_tmpa_tl
117       }
118       {
119         \exp_args:Nc
120         \__tag_mc_bdc_mcid:n
121         {
122           \seq_item:cn { g__tag_mc_#1_marks_seq } {4}
123         }
124         \str_if_eq:eeTF
125         {
126           \seq_item:cn { g__tag_mc_#1_marks_seq } {5}
127         }
128         {
129           %store
130           \__tag_mc_store:eee
131           {
132             \seq_item:cn { g__tag_mc_#1_marks_seq } {2}
133           }
134           { \int_eval:n{\c@g__tag_MCID_abs_int} }
135           {

```



```

136             \seq_item:cn { g__tag_mc_#1_marks_seq } {3}
137         }
138     }
139     {
140         %stashed -> warning!!
141     }
142 }
143 }
144 {
145     \__tag_check_typeout_v:n {=>~ TMB~ not~ missing}
146 }
147 }
148
149 \cs_new_protected:Npn \__tag_mc_insert_extra_tme:n #1 % #1 stream, eg. main or footnote
150 {
151     \__tag_check_if_mc_tme_missing:TF
152     {
153         \__tag_check_typeout_v:n {=>~ TME~ ~ missing~ --- inserted}
154         \__tag_mc_emc:
155         \seq_gset_eq:cN
156         { g__tag_mc_#1_marks_seq }
157         \l__tag_mc_botmarks_seq
158     }
159     {
160         \__tag_check_typeout_v:n {=>~ TME~ not~ missing}
161     }
162 }

```

(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 text, or `multicol` for boxes produced for columns in that environment. Other streams may follow over time.

```

163 \cs_new_protected:Npn \__tag_add_missing_mcs:Nn #1 #2 {
164     \vbadness \@M
165     \vfuzz \c_max_dim
166     \vbox_set_to_ht:Nnn #1 { \box_ht:N #1 } {
167         \hbox_set:Nn \l__tag_tmpa_box { \__tag_mc_insert_extra_tmb:n {#2} }
168         \hbox_set:Nn \l__tag_tmpb_box { \__tag_mc_insert_extra_tme:n {#2} }
169         \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
170         {

```

```

171         \seq_log:c { g__tag_mc_#2_marks_seq}
172     }

```

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

```

173     \box_set_ht:Nn \l__tag_tmpa_box \c_zero_dim
174     \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.

```

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

```

177     \boxmaxdepth \@maxdepth
178     \box_use_drop:N     \l__tag_tmpa_box
179     \vbox_unpack_drop:N     #1

```

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

```

180     \tex_kern:D -\box_dp:N \l__tag_tmpb_box

```

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

```

181     \nointerlineskip
182     \box_use_drop:N \l__tag_tmpb_box
183 }
184 }

```

(End of definition for `\__tag_add_missing_mcs: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.

```

185 \cs_new_protected:Npn \__tag_add_missing_mcs_to_stream:Nn #1#2
186 {
187     \__tag_check_if_active_mc:T {

```

First set up a temp box for trial splitting.

```

188     \vbadness\maxdimen
189     \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).

```

190     \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`

```

191     \exp_args:NNe
192     \seq_set_from_clist:Nn \l__tag_mc_firstmarks_seq
193     { \tex_splitfirstmarks:D \g__tag_mc_marks }

```

Some debugging info:

```
194 % \iow_term:n { First~ mark~ from~ this~ box: }
195 % \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!

```
196 \seq_if_empty:NTF \l__tag_mc_firstmarks_seq
197 {
198 \__tag_check_typeout_v:n
199 {
200 No~ marks~ so~ use~ saved~ bot~ mark:~
201 \seq_use:cn {g__tag_mc_#2_marks_seq} {,~} \iow_newline:
202 }
203 \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).

```
204 \seq_set_eq:NN \l__tag_mc_botmarks_seq \l__tag_mc_firstmarks_seq
205 }
```

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.

```
206 {
207 \__tag_check_typeout_v:n
208 {
209 Pick~ up~ new~ bot~ mark!
210 }
211 \exp_args:NNe
212 \seq_set_from_clist:Nn \l__tag_mc_botmarks_seq
213 { \tex_splitbotmarks:D \g__tag_mc_marks }
214 }
```

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

```
215 \__tag_add_missing_mcs:Nn #1 {#2}
216 %%
217 \seq_gset_eq:cN {g__tag_mc_#2_marks_seq} \l__tag_mc_botmarks_seq
218 %%
219 }
220 }
```

(End of definition for `\__tag_add_missing_mcs_to_stream:Nn`.)

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

`\__tag_mc_if_in:TF`  
`\tag_mc_if_in_p:`  
`\tag_mc_if_in:TF`

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

```

221 \prg_new_conditional:Nnn \__tag_mc_if_in: {p,T,F,TF}
222   {
223     \bool_if:NTF \g__tag_in_mc_bool
224       { \prg_return_true: }
225       { \prg_return_false: }
226   }
227
228 \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 70.)

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

```

229 % #1 tag, #2 properties
230 \cs_set_eq:NN \__tag_mc_bmc:n \pdf_bmc:n
231 \cs_set_eq:NN \__tag_mc_emc: \pdf_emc:
232 \cs_set_eq:NN \__tag_mc_bdc:nn \pdf_bdc:nn
233 \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:VV 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.

```

234 \bool_if:NTF\g__tag_delayed_shipout_bool
235   {
236     \hook_gput_code:nnn {shipout/before}{tagpdf}{ \flag_clear:n { __tag/mcid } }
237     \cs_set_protected:Npn \__tag_mc_bdc_mcid:nn #1 #2
238       {
239         \int_gincr:N \c@g__tag_MCID_abs_int
240         \__tag_property_record:eV
241         {
242           mcid-\int_use:N \c@g__tag_MCID_abs_int
243         }
244         \__tag_property_mc_clist
245         \__tag_mc_bdc_shipout:ee
246         {#1}
247         {
248           /MCID-\flag_height:n { __tag/mcid }
249           \flag_raise:n { __tag/mcid }~ #2
250         }
251       }
252   }

```

if the engine is too old, we have to revert to earlier method.

```

253   {
254     \msg_new:nnn { tagpdf } { old-engine }

```

```

255 {
256   The~engine~or~the~PDF~management~is~too~old~or~\
257   delayed~shipout~has~been~disabled~\
258   Fast~numbering~of~MC~chunks~not~available~\
259   More~compilations~will~be~needed~in~generic~mode.
260 }
261 \msg_warning:nn { tagpdf } { old-engine }
262 \__tag_prop_new:N \g__tag_MCID_byabspage_prop
263 \int_new:N \g__tag_MCID_tmp_bypage_int
264 \cs_generate_variant:Nn \__tag_mc_bdc:nn {ne}
revert the attribute:
265 \property_gset:nxxx {tagmcid} { now }
266   {0} { \int_use:N \g__tag_MCID_tmp_bypage_int }
267 \cs_new_protected:Npn \__tag_mc_bdc_mcid:nn #1 #2
268 {
269   \int_gincr:N \c@g__tag_MCID_abs_int
270   \tl_set:Ne \l__tag_mc_ref_abspage_tl
271   {
272     \property_ref:enn %3 args
273     {
274       mcid-\int_use:N \c@g__tag_MCID_abs_int
275     }
276     { tagabspage }
277     {-1}
278   }
279   \prop_get:NoNTF
280     \g__tag_MCID_byabspage_prop
281     {
282       \l__tag_mc_ref_abspage_tl
283     }
284     \l__tag_mc_tmpa_tl
285     {
286       %key already present, use value for MCID and add 1 for the next
287       \int_gset:Nn \g__tag_MCID_tmp_bypage_int { \l__tag_mc_tmpa_tl }
288       \__tag_prop_gput:Nee
289         \g__tag_MCID_byabspage_prop
290         { \l__tag_mc_ref_abspage_tl }
291         { \int_eval:n { \l__tag_mc_tmpa_tl +1 } }
292     }
293     {
294       %key not present, set MCID to 0 and insert 1
295       \int_gzero:N \g__tag_MCID_tmp_bypage_int
296       \__tag_prop_gput:Nee
297         \g__tag_MCID_byabspage_prop
298         { \l__tag_mc_ref_abspage_tl }
299         { 1 }
300     }
301   \__tag_property_record:eV
302   {
303     mcid-\int_use:N \c@g__tag_MCID_abs_int
304   }
305   \c__tag_property_mc_clist
306   \__tag_mc_bdc:ne
307   { #1 }

```

```

308         { /MCID~\int_eval:n { \g__tag_MCID_tmp_bypage_int }~ \exp_not:n { #2 } }
309     }
310 }
311 \cs_new_protected:Npn \__tag_mc_bdc_mcid:n #1
312 {
313     \__tag_mc_bdc_mcid:nn {#1} {}
314 }
315
316 \cs_new_protected:Npn \__tag_mc_handle_mcid:nn #1 #2 %#1 tag, #2 properties
317 {
318     \__tag_mc_bdc_mcid:nn {#1} {#2}
319 }
320
321 \cs_generate_variant:Nn \__tag_mc_handle_mcid:nn {VV}

```

(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?

\\_\_tag\_mc\_handle\_stash:e

```

322 \cs_new_protected:Npn \__tag_mc_handle_stash:n #1 %1 mcidnum
323 {
324     \__tag_check_mc_used:n {#1}
325     \__tag_struct_kid_mc_gput_right:nn
326     { \g__tag_struct_stack_current_tl }
327     {#1}
328     \prop_gput:Nee \g__tag_mc_parenttree_prop
329     {#1}
330     { \g__tag_struct_stack_current_tl }
331 }
332 \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

```

333 \cs_new_protected:Npn \__tag_mc_bmc_artifact:
334 {
335     \__tag_mc_bmc:n {Artifact}
336 }
337 \cs_new_protected:Npn \__tag_mc_bmc_artifact:n #1
338 {
339     \__tag_mc_bdc:nn {Artifact}{/Type/#1}
340 }
341 \cs_new_protected:Npn \__tag_mc_handle_artifact:N #1
342 % #1 is a var containing the artifact type
343 {
344     \int_gincr:N \c@g__tag_MCID_abs_int
345     \tl_if_empty:NTF #1
346     { \__tag_mc_bmc_artifact: }
347     { \exp_args:NV\__tag_mc_bmc_artifact:n #1 }
348 }

```

(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.

```
349 \cs_new:Nn \__tag_get_data_mc_tag: { \g__tag_mc_key_tag_tl }
350 \</generic>
```

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

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

```
351 <base>\cs_new_protected:Npn \tag_mc_begin:n #1 { \__tag_whatsits: \int_gincr:N \c@g__tag_MCID
352 <base>\cs_new_protected:Nn \tag_mc_end:{ \__tag_whatsits: }
353 <*generic | debug>
354 <*generic>
355 \cs_set_protected:Npn \tag_mc_begin:n #1 %#1 keyval
356 {
357   \__tag_check_if_active_mc:T
358   {
359     \</generic>
360     <*debug>
361     \cs_set_protected:Npn \tag_mc_begin:n #1 %#1 keyval
362     {
363       \__tag_check_if_active_mc:TF
364       {
365         \__tag_debug_mc_begin_insert:n { #1 }
366       \</debug>
367       \group_begin: %hm
368       \__tag_check_mc_if_nested:
369       \bool_gset_true:N \g__tag_in_mc_bool
```

set default MC tags to structure:

```
370   \tl_set_eq:NN \l__tag_mc_key_tag_tl \g__tag_struct_tag_tl
371   \tl_gset_eq:NN\g__tag_mc_key_tag_tl \g__tag_struct_tag_tl
372   \keys_set:nn { __tag / mc } {#1}
373   \bool_if:NTF \l__tag_mc_artifact_bool
374   { %handle artifact
375     \__tag_mc_handle_artifact:N \l__tag_mc_artifact_type_tl
376     \exp_args:NV
377     \__tag_mc_artifact_begin_marks:n \l__tag_mc_artifact_type_tl
378   }
379   { %handle mcid type
380     \__tag_check_mc_tag:N \l__tag_mc_key_tag_tl
381     \__tag_mc_handle_mcid:VV
382     \l__tag_mc_key_tag_tl
383     \l__tag_mc_key_properties_tl
384     \__tag_mc_begin_marks:oo{\l__tag_mc_key_tag_tl}{\l__tag_mc_key_label_tl}
385     \tl_if_empty:NF {\l__tag_mc_key_label_tl}
386     {
387       \exp_args:NV
388       \__tag_mc_handle_mc_label:e \l__tag_mc_key_label_tl
389     }
```

```

390     \bool_if:NF \l__tag_mc_key_stash_bool
391     {
392         \exp_args:NV\__tag_struct_get_parentrole:nNN
393         \g__tag_struct_stack_current_tl
394         \l__tag_get_parent_tmpa_tl
395         \l__tag_get_parent_tmpb_tl
396         \__tag_check_parent_child:VVnnN
397         \l__tag_get_parent_tmpa_tl
398         \l__tag_get_parent_tmpb_tl
399         {MC}{ }
400         \l__tag_parent_child_check_tl
401     \int_compare:nNnT {\l__tag_parent_child_check_tl}<{0}
402     {
403         \prop_get:cnN
404         { g__tag_struct_ \g__tag_struct_stack_current_tl _prop}
405         {S}
406         \l__tag_tmpa_tl
407         \msg_warning:nneee
408         { tag }
409         {role-parent-child}
410         { \l__tag_get_parent_tmpa_tl/\l__tag_get_parent_tmpb_tl }
411         { MC~(real content) }
412         { not~allowed~
413           (struct~\g__tag_struct_stack_current_tl,~\l__tag_tmpa_tl)
414         }
415     }
416     \__tag_mc_handle_stash:e { \int_use:N \c@g__tag_MCID_abs_int }
417 }
418 }
419 \group_end:
420 }
421 <*debug>
422 {
423     \__tag_debug_mc_begin_ignore:n { #1 }
424 }
425 </debug>
426 }
427 <*generic>
428 \cs_set_protected:Nn \tag_mc_end:
429 {
430     \__tag_check_if_active_mc:T
431     {
432 </generic>
433 <*debug>
434 \cs_set_protected:Nn \tag_mc_end:
435 {
436     \__tag_check_if_active_mc:TF
437     {
438         \__tag_debug_mc_end_insert:
439 </debug>
440     \__tag_check_mc_if_open:
441     \bool_gset_false:N \g__tag_in_mc_bool
442     \tl_gset:Nn \g__tag_mc_key_tag_tl { }
443     \__tag_mc_emc:

```



```

444     \__tag_mc_end_marks:
445   }
446   <*debug>
447   {
448     \__tag_debug_mc_end_ignore:
449   }
450 </debug>
451   }
452 </generic | debug>

```

(End of definition for `\tag_mc_begin:n` and `\tag_mc_end:.` These functions are documented on page 70.)

## 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)
alt_␣(mc-key)
actualtext_␣(mc-key)
label_␣(mc-key)
artifact_␣(mc-key)
453 <*generic>
454 \keys_define:nn { __tag / mc }
455 {
456   tag .code:n = % the name (H,P,Span) etc
457   {
458     \tl_set:Ne \l__tag_mc_key_tag_tl { #1 }
459     \tl_gset:Ne \g__tag_mc_key_tag_tl { #1 }
460   },
461   raw .code:n =
462   {
463     \tl_put_right:Ne \l__tag_mc_key_properties_tl { #1 }
464   },
465   alt .code:n = % Alt property
466   {
467     \str_set_convert:Noon
468     \l__tag_tmpa_str
469     { #1 }
470     { default }
471     { utf16/hex }
472     \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
473     \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
474   },
475   alttext .meta:n = {alt=#1},
476   actualtext .code:n = % ActualText property
477   {
478     \tl_if_empty:oF{#1}
479     {
480       \str_set_convert:Noon
481       \l__tag_tmpa_str
482       { #1 }
483       { default }
484       { utf16/hex }
485       \tl_put_right:Nn \l__tag_mc_key_properties_tl { /ActualText~< }
486       \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
487     }

```

```

488     },
489     label .tl_set:N          = \l__tag_mc_key_label_tl,
490     artifact .code:n        =
491     {
492         \exp_args:Nne
493         \keys_set:nn
494         { __tag / mc }
495         { __artifact-bool, __artifact-type=#1 }
496     },
497     artifact .default:n     = {notype}
498 }
499 </generic>

```

*(End of definition for tag (mc-key) and others. These functions are documented on page 71.)*

## Part VI

# The `tagpdf-mc-luacode` module Code related to Marked Content (mc-chunks), luamode-specific Part of the `tagpdf` package

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

## 1 Marked content code – luamode code

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

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

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

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

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

`tag` : the type (a string)

`raw` : more properties (string)

`label`: a string.

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

`kids`: a array of tables

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

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

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

```
1 <@@=tag>
2 <*luamode>
3 \ProvidesExplPackage {tagpdf-mc-code-lua} {2024-09-11} {0.99e}
4   {tagpdf - mc code only for the luamode }
5 </luamode>
6 <*debug>
7 \ProvidesExplPackage {tagpdf-debug-lua} {2024-09-11} {0.99e}
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 {}

```

(End of definition for `\_tag_add_missing_mcs_to_stream:Nn`.)

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

```

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

```

```

67 {
68   \int_compare:nNnTF
69     { -2147483647 }
70     =
71     {\lua_now:e
72      {
73        tex.print(\int_use:N \c_document_cctab, tex.getattribute(luatexbase.attributes.g__tag_mc_if_in:))
74      }
75     }
76     { \prg_return_false: }
77     { \prg_return_true: }
78 }

```

```

80 \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 70.)

`\_tag_mc_lua_set_mc_type_attr:` This takes a tag name, and sets the attributes globally to the related number.

```

81 \cs_new:Nn \_tag_mc_lua_set_mc_type_attr:n % #1 is a tag name
82 {
83   %TODO ltx.__tag.func.get_num_from("#1") seems not to return a suitable number??
84   \tl_set:Nc\l__tag_tmpa_tl{\lua_now:e{ltx.__tag.func.output_num_from("#1")}}
85   \lua_now:e
86   {
87     tex.setattribute
88     (
89       "global",
90       luatexbase.attributes.g__tag_mc_type_attr,
91       \l__tag_tmpa_tl
92     )
93   }
94   \lua_now:e
95   {
96     tex.setattribute
97     (
98       "global",
99       luatexbase.attributes.g__tag_mc_cnt_attr,
100     \_tag_get_mc_abs_cnt:
101     )

```

```

102     }
103   }
104
105 \cs_generate_variant:Nn\__tag_mc_lua_set_mc_type_attr:n { o }
106
107 \cs_new:Nn \__tag_mc_lua_unset_mc_type_attr:
108 {
109   \lua_now:e
110   {
111     tex.setattribute
112     (
113       "global",
114       luatexbase.attributes.g__tag_mc_type_attr,
115       -2147483647
116     )
117   }
118   \lua_now:e
119   {
120     tex.setattribute
121     (
122       "global",
123       luatexbase.attributes.g__tag_mc_cnt_attr,
124       -2147483647
125     )
126   }
127 }
128

```

(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

```

129 \cs_new:Nn \__tag_mc_insert_mcid_kids:n
130 {
131   \lua_now:e { ltx.__tag.func.mc_insert_kids (#1,0) }
132 }
133
134 \cs_new:Nn \__tag_mc_insert_mcid_single_kids:n
135 {
136   \lua_now:e {ltx.__tag.func.mc_insert_kids (#1,1) }
137 }

```

(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.

```

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

```

```

146     { g__tag_struct_kids_\g__tag_struct_stack_current_tl _seq }
147     {
148         \__tag_mc_insert_mcid_kids:n {#1}%
149     }
150 <debug> \seq_gput_right:cn % Don't fill a lua table due to the command in the item,
151 <debug> % so use the kernel command
152 <debug> { g__tag_struct_debug_kids_\g__tag_struct_stack_current_tl _seq }
153 <debug> {
154 <debug>     MC~#1%
155 <debug> }
156 \lua_now:e
157 {
158     ltx.__tag.func.store_struct_mcab
159     (
160         \g__tag_struct_stack_current_tl,#1
161     )
162 }
163 \prop_gput:Nee
164 \g__tag_mc_parenttree_prop
165 { #1 }
166 { \g__tag_struct_stack_current_tl }
167 }
168 </luamode | debug>
169 <*luamode>
170 \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.

```

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

```

set the default tag to the structure:

```

181         \tl_set_eq:NN \l__tag_mc_key_tag_tl \g__tag_struct_tag_tl
182         \tl_gset_eq:NN\g__tag_mc_key_tag_tl \g__tag_struct_tag_tl
183         \lua_now:e
184         {
185             ltx.__tag.func.store_mc_data(\__tag_get_mc_abs_cnt:,"tag","\g__tag_struct_tag_tl
186         }
187         \keys_set:nn { __tag / mc }{ label={}, #1 }
188         %check that a tag or artifact has been used
189         \__tag_check_mc_tag:N \l__tag_mc_key_tag_tl
190         %set the attributes:
191         \__tag_mc_lua_set_mc_type_attr:o { \l__tag_mc_key_tag_tl }
192         \bool_if:NF \l__tag_mc_artifact_bool
193         { % store the absolute num name in a label:

```

```

194     \tl_if_empty:NF {\l__tag_mc_key_label_tl}
195     {
196         \exp_args:NV
197         \__tag_mc_handle_mc_label:e \l__tag_mc_key_label_tl
198     }
199     % if not stashed record the absolute number
200     \bool_if:NF \l__tag_mc_key_stash_bool
201     {
202         \exp_args:NV\__tag_struct_get_parentrole:nNN
203         \g__tag_struct_stack_current_tl
204         \l__tag_get_parent_tmpa_tl
205         \l__tag_get_parent_tmpb_tl
206         \__tag_check_parent_child:VVnnN
207         \l__tag_get_parent_tmpa_tl
208         \l__tag_get_parent_tmpb_tl
209         {MC}{ }
210         \l__tag_parent_child_check_tl
211         \int_compare:nNnT {\l__tag_parent_child_check_tl}<{0}
212         {
213             \prop_get:cnN
214             { g__tag_struct_ \g__tag_struct_stack_current_tl _prop}
215             {S}
216             \l__tag_tmpa_tl
217             \msg_warning:nneee
218             { tag }
219             {role-parent-child}
220             { \l__tag_get_parent_tmpa_tl/\l__tag_get_parent_tmpb_tl }
221             { MC~(real content) }
222             {
223                 not~allowed~
224                 (struct~\g__tag_struct_stack_current_tl,~\l__tag_tmpa_tl)
225             }
226         }
227         \__tag_mc_handle_stash:e { \__tag_get_mc_abs_cnt: }
228     }
229 }
230 \group_end:
231 }
232 }

```

(End of definition for `\tag_mc_begin:n`. This function is documented on page 70.)

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

```

233 \cs_set_protected:Nn \tag_mc_end:
234 {
235     \__tag_check_if_active_mc:T
236     {
237         %\__tag_check_mc_if_open:
238         \bool_gset_false:N \g__tag_in_mc_bool
239         \bool_set_false:N\l__tag_mc_artifact_bool
240         \__tag_mc_lua_unset_mc_type_attr:
241         \tl_set:Nn \l__tag_mc_key_tag_tl { }
242         \tl_gset:Nn \g__tag_mc_key_tag_tl { }
243     }
244 }

```



(End of definition for `\tag_mc_end:`. This function is documented on page 70.)

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

```

245 \cs_set_protected:Npn \tag_mc_reset_box:N #1
246   {
247     \lua_now:e
248     {
249       local~type=tex.getattribute(luatexbase.attributes.g__tag_mc_type_attr)
250       local~mc=tex.getattribute(luatexbase.attributes.g__tag_mc_cnt_attr)
251       ltx.__tag.func.update_mc_attributes(tex.getbox(\int_use:N #1),mc,type)
252     }
253   }

```

(End of definition for `\tag_mc_reset_box:N`. This function is documented on page 71.)

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

```

254 \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)  255 \keys_define:nn { __tag / mc }
alt_␣(mc-key)  256   {
actualtext_␣(mc-key) 257   tag .code:n = %
label_␣(mc-key)    258   {
artifact_␣(mc-key) 259     \tl_set:Ne \l__tag_mc_key_tag_tl { #1 }
260     \tl_gset:Ne \g__tag_mc_key_tag_tl { #1 }
261     \lua_now:e
262     {
263       ltx.__tag.func.store_mc_data(\__tag_get_mc_abs_cnt:,"tag", "#1")
264     }
265   },
266   raw .code:n =
267   {
268     \tl_put_right:Ne \l__tag_mc_key_properties_tl { #1 }
269     \lua_now:e
270     {
271       ltx.__tag.func.store_mc_data(\__tag_get_mc_abs_cnt:,"raw", "#1")
272     }
273   },
274   alt .code:n      = % Alt property
275   {
276     \tl_if_empty:oF{#1}
277     {
278       \str_set_convert:Noon
279       \l__tag_tmpa_str
280       { #1 }
281       { default }
282       { utf16/hex }
283       \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }

```

```

284         \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
285         \lua_now:e
286         {
287             ltx.__tag.func.store_mc_data
288             (
289                 \__tag_get_mc_abs_cnt:,"alt","/Alt~<\str_use:N \l__tag_tmpa_str>"
290             )
291         }
292     }
293 },
294 alttext .meta:n = {alt=#1},
295 actualtext .code:n = % Alt property
296 {
297     \tl_if_empty:oF{#1}
298     {
299         \str_set_convert:Noon
300         \l__tag_tmpa_str
301         { #1 }
302         { default }
303         { utf16/hex }
304         \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
305         \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
306         \lua_now:e
307         {
308             ltx.__tag.func.store_mc_data
309             (
310                 \__tag_get_mc_abs_cnt:,
311                 "actualtext",
312                 "/ActualText~<\str_use:N \l__tag_tmpa_str>"
313             )
314         }
315     }
316 },
317 label .code:n =
318 {
319     \tl_set:Nn\l__tag_mc_key_label_tl { #1 }
320     \lua_now:e
321     {
322         ltx.__tag.func.store_mc_data
323         (
324             \__tag_get_mc_abs_cnt:,"label", "#1"
325         )
326     }
327 },
328 __artifact-store .code:n =
329 {
330     \lua_now:e
331     {
332         ltx.__tag.func.store_mc_data
333         (
334             \__tag_get_mc_abs_cnt:,"artifact", "#1"
335         )
336     }
337 },

```

```

338 artifact .code:n      =
339   {
340     \exp_args:Nne
341     \keys_set:nn
342     { __tag / mc }
343     { __artifact-bool, __artifact-type=#1, tag=Artifact }
344     \exp_args:Nne
345     \keys_set:nn
346     { __tag / mc }
347     { __artifact-store=\l__tag_mc_artifact_type_tl }
348   },
349 artifact .default:n    = { notype }
350 }
351
352 </luamode>

```

*(End of definition for tag (mc-key) and others. These functions are documented on page 71.)*

## Part VII

# The tagpdf-struct module

## Commands to create the structure Part of the tagpdf package

### 1 Public Commands

---

<code>\tag_struct_begin:n</code>	<code>\tag_struct_begin:n{&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;struct 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.

---

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

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)

---

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

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

## 2 Public keys

### 2.1 Keys for the structure commands

---

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

---

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

---

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

---

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

---

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

---

alt<sub>□</sub>(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<sub>□</sub>(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<sub>□</sub>(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<sub>□</sub>(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<sub>□</sub>(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<sub>□</sub>(struct-key) AF = *<object name>*

AFref<sub>□</sub>(struct-key) AFref = *<object reference>*

AFinline<sub>□</sub>(struct-key) AF-inline = *<text content>*

AFinline-o<sub>□</sub>(struct-key) These keys allows to reference an associated file in the structure element. 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`.

texsource

mathml

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 file 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 file 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<sub>□</sub>(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<sub>□</sub>(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<sub>□</sub>(setup-key)** `role/new-attribute = {<name>}{<Content>}`  
**newattribute<sub>□</sub>(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<sub>□</sub>(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} {2024-09-11} {0.99e}
4 {part of tagpdf - code related to storing structure}
5 </header>
```

## 3 Variables

`\c@g__tag_struct_abs_int` Every structure will have a unique, absolute number. I will use a latex counter for the structure count to have a chance to avoid double structures in align etc.

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

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

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

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

(End of definition for `\g__tag_struct_objR_seq`.)

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

```
10 \tl_const:Nn\c__tag_struct_null_tl {null}
```

(End of definition for `\c__tag_struct_null_tl`.)

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

```
11 \__tag_prop_new:N \g__tag_struct_cont_mc_prop
```

(End of definition for `\g__tag_struct_cont_mc_prop`.)

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

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

(End of definition for `\g__tag_struct_stack_seq`.)

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

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

(End of definition for `\g__tag_struct_tag_stack_seq`.)

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

```
16 </package>
17 <base>\tl_new:N \g__tag_struct_stack_current_tl
18 <base>\tl_gset:Nn \g__tag_struct_stack_current_tl {\int_use:N\c@g__tag_struct_abs_int}
19 <*package>
20 \tl_new:N \l__tag_struct_stack_parent_tpa_tl
```

(End of definition for `\g__tag_struct_stack_current_tl` and `\l__tag_struct_stack_parent_tpa_tl`.)

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

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

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

All properties have at least the keys



## Type StructTreeRoot or StructElem

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

```
21 \seq_const_from_clist:Nn \c__tag_struct_StructTreeRoot_entries_seq
22   {%p 857/858
23     Type,           % always /StructTreeRoot
24     K,             % kid, dictionary or array of dictionaries
25     IDTree,       % currently unused
26     ParentTree,   % required,obj ref to the parent tree
27     ParentTreeNextKey, % optional
28     RoleMap,
29     ClassMap,
30     Namespaces,
31     AF            %pdf 2.0
32   }
33
34 \seq_const_from_clist:Nn \c__tag_struct_StructElem_entries_seq
35   {%p 858 f
36     Type,           %always /StructElem
37     S,             %tag/type
38     P,            %parent
39     ID,           %optional
40     Ref,          %optional, pdf 2.0 Use?
41     Pg,          %obj num of starting page, optional
42     K,           %kids
43     A,           %attributes, probably unused
44     C,           %class ""
45     %R,          %attribute revision number, irrelevant for us as we
46                 % don't update/change existing PDF and (probably)
47                 % deprecated in PDF 2.0
48     T,           %title, value in () or <>
49     Lang,        %language
50     Alt,         % value in () or <>
51     E,           % abbreviation
52     ActualText,
53     AF,          %pdf 2.0, array of dict, associated files
54     NS,          %pdf 2.0, dict, namespace
55     PhoneticAlphabet, %pdf 2.0
56     Phoneme      %pdf 2.0
57   }
```

(End of definition for \c\_\_tag\_struct\_StructTreeRoot\_entries\_seq and \c\_\_tag\_struct\_StructElem\_entries\_seq.)

### 3.1 Variables used by the keys

Use by the tag key to store the tag and the namespace. The role tag variables will hold locally rolemapping info needed for the parent-child checks

```
\g__tag_struct_tag_t1
\g__tag_struct_tag_NS_t1
\l__tag_struct_roletag_t1
\g__tag_struct_roletag_NS_t1
```

```

58 \tl_new:N \g__tag_struct_tag_tl
59 \tl_new:N \g__tag_struct_tag_NS_tl
60 \tl_new:N \l__tag_struct_roletag_tl
61 \tl_new:N \l__tag_struct_roletag_NS_tl

```

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

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

```

62 \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

```

63 \bool_new:N \l__tag_struct_elem_stash_bool

```

(End of definition for `\l__tag_struct_elem_stash_bool`.)

### 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.

```

64 \package
65 \base\prop_new_linked:N \g__tag_struct_dest_num_prop
66 \*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.

```

67 \prop_new_linked:N \g__tag_struct_ref_by_dest_prop

```

(End of definition for `\g__tag_struct_ref_by_dest_prop`.)

## 4 Commands

The properties must be in some places handled expandably. So I need an output handler for each prop, to get expandable output see <https://tex.stackexchange.com/questions/424208>. There is probably room here for a more efficient implementation. TODO check if this can now be implemented with the `pdfdict` commands. The property contains currently non pdf keys, but e.g. object numbers are perhaps no longer needed as we have named object anyway.

```

\__tag_struct_output_prop_aux:nn
\__tag_new_output_prop_handler:n
68 \cs_new:Npn \__tag_struct_output_prop_aux:nn #1 #2 %#1 num, #2 key
69 {
70   \prop_if_in:cnT
71     { g__tag_struct_#1_prop }
72     { #2 }
73     {
74       \c_space_tl/#2~ \prop_item:cn{ g__tag_struct_#1_prop } { #2 }

```

```

75     }
76   }
77
78   \cs_new_protected:Npn \__tag_new_output_prop_handler:n #1
79   {
80     \cs_new:cn { \__tag_struct_output_prop_#1:n }
81     {
82       \__tag_struct_output_prop_aux:nn {#1}{#1}
83     }
84   }
85 \end{package}

```

(End of definition for \\_\_tag\_struct\_output\_prop\_aux:nn and \\_\_tag\_new\_output\_prop\_handler:n.)

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

```

86 \begin{package} \debug
87 \package\cs_new_protected:Npn \__tag_struct_prop_gput:nnn #1 #2 #3
88 \debug\cs_set_protected:Npn \__tag_struct_prop_gput:nnn #1 #2 #3
89 {
90   \__tag_prop_gput:cnn
91   { g__tag_struct_#1_prop }{#2}{#3}
92 \debug\prop_gput:cnn { g__tag_struct_debug_#1_prop } {#2} {#3}
93 }
94 \cs_generate_variant:Nn \__tag_struct_prop_gput:nnn {onn,nne,nee,nno}
95 \end{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.

```

96 \begin{package}
97 \tl_gset:Nn \g__tag_struct_stack_current_tl {1}

```

\\_\_tag\_pdf\_name\_e:n

```

98 \cs_new:Npn \__tag_pdf_name_e:n #1{\pdf_name_from_unicode_e:n{#1}}
99 \end{package}

```

(End of definition for \\_\_tag\_pdf\_name\_e:n.)

g\_\_tag\_struct\_1\_prop  
g\_\_tag\_struct\_kids\_1\_seq

```

100 \begin{package}
101 \__tag_prop_new:c { g__tag_struct_1_prop }
102 \__tag_new_output_prop_handler:n {1}
103 \__tag_seq_new:c { g__tag_struct_kids_1_seq }
104
105 \__tag_struct_prop_gput:nne
106   { 1 }
107   { Type }

```

```

108 { \pdf_name_from_unicode_e:n {StructTreeRoot} }
109
110 \__tag_struct_prop_gput:nne
111 { 1 }
112 { S }
113 { \pdf_name_from_unicode_e:n {StructTreeRoot} }
114
115 \__tag_struct_prop_gput:nne
116 { 1 }
117 { rolemap }
118 { {StructTreeRoot}{pdf} }
119
120 \__tag_struct_prop_gput:nne
121 { 1 }
122 { parentrole }
123 { {StructTreeRoot}{pdf} }
124

```

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

```

125 \pdf_version_compare:NnF < {2.0}
126 {
127   \__tag_struct_prop_gput:nne
128   { 1 }
129   { Namespaces }
130   { \pdf_object_ref:n { __tag/tree/namespaces } }
131 }
132 </package>

```

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

```

133 <debug>\prop_new:c { g__tag_struct_debug_1_prop }
134 <debug>\seq_new:c { g__tag_struct_debug_kids_1_seq }
135 <debug>\prop_gset_eq:cc { g__tag_struct_debug_1_prop }{ g__tag_struct_1_prop }
136 <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
137 <*package>
138 \cs_new:Npn \__tag_struct_get_id:n #1 %#1=struct num
139 {
140   (
141     ID.
142     \prg_replicate:nn
143     { \int_abs:n{\g__tag_tree_id_pad_int - \tl_count:e { \int_to_arabic:n { #1 } }} }
144     { 0 }
145     \int_to_arabic:n { #1 }
146   )
147 }

```

(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.

```

148 \pdf_version_compare:NnTF < {2.0}
149 {
150   \cs_new_protected:Npn \_tag_struct_set_tag_info:nnn #1 #2 #3
151     %#1 structure number, #2 tag, #3 NS
152     {
153       \_tag_struct_prop_gput:nne
154         { #1 }
155         { S }
156         { \pdf_name_from_unicode_e:n {#2} } %
157     }
158 }
159 {
160   \cs_new_protected:Npn \_tag_struct_set_tag_info:nnn #1 #2 #3
161     {
162       \_tag_struct_prop_gput:nne
163         { #1 }
164         { S }
165         { \pdf_name_from_unicode_e:n {#2} } %
166       \prop_get:NnNT \g__tag_role_NS_prop {#3} \l__tag_get_tmpc_tl
167       {
168         \_tag_struct_prop_gput:nne
169           { #1 }
170           { NS }
171           { \l__tag_get_tmpc_tl } %
172       }
173     }
174 }
175 \cs_generate_variant:Nn \_tag_struct_set_tag_info:nnn {eVV}

```

(End of definition for `\_tag_struct_set_tag_info:nnn`.)

`\_tag_struct_get_parentrole:nnN` We also need a way to get the tag info needed for parent child check from parent structures.

```

176 \cs_new_protected:Npn \_tag_struct_get_parentrole:nnN #1 #2 #3
177   %#1 struct num, #2 tlvvar for tag , #3 tlvvar for NS
178   {
179     \prop_get:cnNTF
180       { g__tag_struct_#1_prop }
181       { parentrole }
182     \l__tag_get_tmpc_tl
183     {
184       \tl_set:Ne #2{\exp_last_unbraced:NV\use_i:nn \l__tag_get_tmpc_tl}
185       \tl_set:Ne #3{\exp_last_unbraced:NV\use_ii:nn \l__tag_get_tmpc_tl}
186     }
187     {
188       \tl_clear:N#2
189       \tl_clear:N#3
190     }
191   }
192 \cs_generate_variant:Nn \_tag_struct_get_parentrole:nnN {eNN}

```

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

## 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!

```
193 \cs_new:Npn \_tag_struct_mcid_dict:n #1 %#1 MCID absnum
194   {
195     <<
196     /Type \c_space_tl /MCR \c_space_tl
197     /Pg
198     \c_space_tl
199     \pdf_pageobject_ref:n { \property_ref:enn{mcid-#1}{tagabspage}{1} }
200     /MCID \c_space_tl \property_ref:enn{mcid-#1}{tagmcid}{1}
201     >>
202   }
203 </package>
204 <*package | debug>
205 <package>\cs_new_protected:Npn \_tag_struct_kid_mc_gput_right:nn #1 #2 %#1 structure num, #2
206 <debug>\cs_set_protected:Npn \_tag_struct_kid_mc_gput_right:nn #1 #2 %#1 structure num, #2 M
207   {
208     \_tag_seq_gput_right:ce
209     { g__tag_struct_kids_#1_seq }
210     {
211       \_tag_struct_mcid_dict:n {#2}
212     }
213 <debug> \seq_gput_right:cn
214 <debug> { g__tag_struct_debug_kids_#1_seq }
215 <debug> {
216 <debug>   MC~#2
217 <debug> }
218 \_tag_seq_gput_right:cn
219 { g__tag_struct_kids_#1_seq }
220 {
221 \prop_item:Nn \g__tag_struct_cont_mc_prop {#2}
222 }
223 }
224 <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.

```
225 <package>\cs_new_protected:Npn \_tag_struct_kid_struct_gput_right:nn #1 #2 %#1 num of parent s
```

```

226 <debug>\cs_set_protected:Npn\__tag_struct_kid_struct_gput_right:nn #1 #2 %#1 num of parent st
227 {
228   \__tag_seq_gput_right:ce
229   { g__tag_struct_kids_#1_seq }
230   {
231     \pdf_object_ref_indexed:nn { __tag/struct }{ #2 }
232   }
233 <debug>   \seq_gput_right:cn
234 <debug>   { g__tag_struct_debug_kids_#1_seq }
235 <debug>   {
236 <debug>     Struct~#2
237 <debug>   }
238 }
239
240 <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\_OBJR\_gput\_right:nn  
\\_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

```

241 <package>\cs_new_protected:Npn\__tag_struct_kid_OBJR_gput_right:nnn #1 #2 #3 %#1 num of parent
242 <package>                                     %#2 obj reference
243 <package>                                     %#3 page object referenc
244 <debug>\cs_set_protected:Npn\__tag_struct_kid_OBJR_gput_right:nnn #1 #2 #3
245 {
246   \pdf_object_unnamed_write:nn
247   { dict }
248   {
249     /Type/OBJR/Obj~#2/Pg~#3
250   }
251   \__tag_seq_gput_right:ce
252   { g__tag_struct_kids_#1_seq }
253   {
254     \pdf_object_ref_last:
255   }
256 <debug>   \seq_gput_right:ce
257 <debug>   { g__tag_struct_debug_kids_#1_seq }
258 <debug>   {
259 <debug>     OBJR~reference
260 <debug>   }
261 }
262 </package | debug>
263 <*package>
264 \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.

```

265 \cs_new_protected:Npn\__tag_struct_exchange_kid_command:N #1 %#1 = seq var
266 {

```

```

267 \seq_gpop_left:NN #1 \l__tag_tmpa_tl
268 \tl_replace_once:Nnn \l__tag_tmpa_tl
269   {\__tag_mc_insert_mcid_kids:n}
270   {\__tag_mc_insert_mcid_single_kids:n}
271 \seq_gput_left:NV #1 \l__tag_tmpa_tl
272 }
273
274 \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.

```

275 \cs_new_protected:Npn \__tag_struct_fill_kid_key:n #1 %#1 is the struct num
276   {
277     \bool_if:NF\g__tag_mode_lua_bool
278     {
279       \seq_clear:N \l__tag_tmpa_seq
280       \seq_map_inline:cn { g__tag_struct_kids_#1_seq }
281         { \seq_put_right:Ne \l__tag_tmpa_seq { ##1 } }
282       %\seq_show:c { g__tag_struct_kids_#1_seq }
283       %\seq_show:N \l__tag_tmpa_seq
284       \seq_remove_all:Nn \l__tag_tmpa_seq {}
285       %\seq_show:N \l__tag_tmpa_seq
286       \seq_gset_eq:cN { g__tag_struct_kids_#1_seq } \l__tag_tmpa_seq
287     }
288
289     \int_case:nnF
290     {
291       \seq_count:c
292       {
293         g__tag_struct_kids_#1_seq
294       }
295     }
296     {
297       { 0 }
298       { } %no kids, do nothing
299       { 1 } % 1 kid, insert
300       {
301         % in this case we need a special command in
302         % luamode to get the array right. See issue #13
303         \bool_if:NTF\g__tag_mode_lua_bool
304         {
305           \__tag_struct_exchange_kid_command:c
306           {g__tag_struct_kids_#1_seq}

```

check if we get null

```

307         \tl_set:Ne\l__tag_tmpa_tl
308         {\use:e{\seq_item:cn {g__tag_struct_kids_#1_seq} {1}}}
309         \tl_if_eq:NNF\l__tag_tmpa_tl \c__tag_struct_null_tl
310         {
311           \__tag_struct_prop_gput:nne
312           {#1}
313           {K}

```



```

314         {
315             \seq_item:cn
316             {
317                 g__tag_struct_kids_#1_seq
318             }
319             {1}
320         }
321     }
322 }
323 {
324     \__tag_struct_prop_gput:nne
325     {#1}
326     {K}
327     {
328         \seq_item:cn
329         {
330             g__tag_struct_kids_#1_seq
331         }
332         {1}
333     }
334 }
335 } %
336 }
337 { %many kids, use an array
338     \__tag_struct_prop_gput:nne
339     {#1}
340     {K}
341     {
342         [
343             \seq_use:cn
344             {
345                 g__tag_struct_kids_#1_seq
346             }
347             {
348                 \c_space_tl
349             }
350         ]
351     }
352 }
353 }
354

```

(End of definition for \\_\_tag\_struct\_fill\_kid\_key:n.)

## 4.5 Output of the object

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

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.

```

355 \cs_new_protected:Npn \__tag_struct_get_dict_content:nN #1 #2 %#1: structure num
356 {
357     \tl_clear:N #2
358     \prop_map_inline:cn { g__tag_struct_#1_prop }
359     {

```

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.

```

360     \cs_if_exist_use:cTF {__tag_struct_format_##1:nnN}
361     {
362       {##1}{##2}#2
363     }
364     {
365       \tl_put_right:Ne #2 { \c_space_tl/##1-##2 }
366     }
367   }
368 }

```

(End of definition for `\__tag_struct_get_dict_content:nnN`.)

`\__tag_struct_format_rolemap:nnN`  
`\__tag_struct_format_parentrole:nnN`

This two entries should not end in the PDF.

```

369 \cs_new:Nn\__tag_struct_format_rolemap:nnN{}
370 \cs_new:Nn\__tag_struct_format_parentrole:nnN{}

```

(End of definition for `\__tag_struct_format_rolemap:nnN` and `\__tag_struct_format_parentrole: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.

```

371 \cs_new_protected:Nn\__tag_struct_format_Ref:nnN
372 {
373   \tl_put_right:Nn #3 { ~/#1-[ ] %]
374   \clist_map_inline:nn{ #2 }
375   {
376     ##1 #3
377   }
378   \tl_put_right:Nn #3
379   { %[
380     \c_space_tl]
381   }
382 }

```

(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.

```

383 \cs_new_protected:Npn \__tag_struct_write_obj:n #1 % #1 is the struct num
384 {
385   \prop_if_exist:cTF { g__tag_struct_#1_prop }
386   {

```

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.

```

387     \prop_get:cnNF { g__tag_struct_#1_prop } {P}\l__tag_tmpb_tl
388     {
389       \prop_gput:cne { g__tag_struct_#1_prop } {P}{\pdf_object_ref_indexed:nn { __tag/
390       \prop_gput:cne { g__tag_struct_#1_prop } {S}{/Artifact}
391       \seq_if_empty:cF {g__tag_struct_kids_#1_seq}
392       {
393         \msg_warning:nnee
394         {tag}

```

```

395         {struct-orphan}
396         { #1 }
397         {\seq_count:c{g__tag_struct_kids_#1_seq}}
398     }
399 }
400 \__tag_struct_fill_kid_key:n { #1 }
401 \__tag_struct_get_dict_content:nN { #1 } \l__tag_tmpa_tl
402 \pdf_object_write_indexed:nne
403   { __tag/struct }{ #1 }
404   {dict}
405   {
406     \l__tag_tmpa_tl\c_space_tl
407     /ID-\__tag_struct_get_id:n{#1}
408   }
409
410 }
411 {
412   \msg_error:nnn { tag } { struct-no-objnum } { #1}
413 }
414 }

```

(End of definition for \\_\_tag\_struct\_write\_obj:n.)

\\_\_tag\_struct\_insert\_annot:mn 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

```

\tag_struct_begin:n { tag=Link }
\tag_mc_begin:n { tag=Link }
(1) \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:

```

```

415 \cs_new_protected:Npn \__tag_struct_insert_annot:nn #1 #2 %#1 object reference to the annotat
416                                     %#2 structparent number
417 {
418   \bool_if:NT \g__tag_active_struct_bool
419   {
420     %get the number of the parent structure:
421     \seq_get:NNF
422     \g__tag_struct_stack_seq
423     \l__tag_struct_stack_parent_tmpa_tl

```

```

424     {
425     \msg_error:nn { tag } { struct-faulty-nesting }
426     }
427     %put the obj number of the annot in the kid entry, this also creates
428     %the OBJR object
429     \__tag_property_record:nn {@tag@objr@page@#2 }{ tagabspage }
430     \__tag_struct_kid_OBJR_gput_right:eee
431     {
432     \l__tag_struct_stack_parent_tmpa_tl
433     }
434     {
435     #1 %
436     }
437     {
438     \pdf_pageobject_ref:n { \property_ref:nnn {@tag@objr@page@#2 }{ tagabspage }{1} }
439     }
440     % add the parent obj number to the parent tree:
441     \exp_args:Nne
442     \__tag_parenttree_add_objr:nn
443     {
444     #2
445     }
446     {
447     \pdf_object_ref_indexed:nn { __tag/struct }{ \l__tag_struct_stack_parent_tmpa_tl
448     }
449     % increase the int:
450     \int_gincr:N \c@g__tag_parenttree_obj_int
451     }
452 }

```

(End of definition for \\_\_tag\_struct\_insert\_annot:nn.)

\\_\_tag\_get\_data\_struct\_tag: this command allows \tag\_get:n to get the current structure tag with the keyword **struct\_tag**.

```

453 \cs_new:Npn \__tag_get_data_struct_tag:
454 {
455   \exp_args:Ne
456   \tl_tail:n
457   {
458     \prop_item:cn {g__tag_struct_\g__tag_struct_stack_current_tl _prop}{S}
459   }
460 }

```

(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**.

```

461 \cs_new:Npn \__tag_get_data_struct_id:
462 {
463   \__tag_struct_get_id:n {\g__tag_struct_stack_current_tl}
464 }
465 </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

```

466 <*base>
467 \cs_new:Npn \_tag_get_data_struct_num:
468   {
469     \g__tag_struct_stack_current_tl
470   }
471 </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.

```

472 <*base>
473 \cs_new:Npn \_tag_get_data_struct_counter:
474   {
475     \int_use:N \c@g__tag_struct_abs_int
476   }
477 </base>

```

*(End of definition for \\_tag\_get\_data\_struct\_counter:.)*

## 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.

```

478 <*package>
479 \socket_new:nn { tag/struct/tag }{1}
480 \socket_new_plug:nnn { tag/struct/tag }{ latex-tags }
481   {
482     \seq_set_split:Nne \l__tag_tmpa_seq { / } {#1/\prop_item:Ne\g__tag_role_tags_NS_prop{#1}}
483     \tl_gset:Ne \g__tag_struct_tag_tl { \seq_item:Nn\l__tag_tmpa_seq {1} }
484     \tl_gset:Ne \g__tag_struct_tag_NS_tl{ \seq_item:Nn\l__tag_tmpa_seq {2} }
485     \_tag_check_structure_tag:N \g__tag_struct_tag_tl
486   }
487
488 \socket_new_plug:nnn { tag/struct/tag }{ pdf-tags }
489   {
490     \seq_set_split:Nne \l__tag_tmpa_seq { / } {#1/\prop_item:Ne\g__tag_role_tags_NS_prop{#1}}
491     \tl_gset:Ne \g__tag_struct_tag_tl { \seq_item:Nn\l__tag_tmpa_seq {1} }
492     \tl_gset:Ne \g__tag_struct_tag_NS_tl{ \seq_item:Nn\l__tag_tmpa_seq {2} }
493     \_tag_role_get:VVNN \g__tag_struct_tag_tl\g__tag_struct_tag_NS_tl\l__tag_tmpa_tl\l__tag_t
494     \tl_gset:Ne \g__tag_struct_tag_tl {\l__tag_tmpa_tl}
495     \tl_gset:Ne \g__tag_struct_tag_NS_tl{\l__tag_tmpb_tl}
496     \_tag_check_structure_tag:N \g__tag_struct_tag_tl
497   }
498 \socket_assign_plug:nn { tag/struct/tag } {latex-tags}

```

```

label_(struct-key)
stash_(struct-key) 499 \keys_define:nn { __tag / struct }
parent_(struct-key) 500 {
tag_(struct-key) 501 label .code:n =
title_(struct-key) 502 {
title-o_(struct-key) 503 \prop_gput:Nee\g__tag_struct_label_num_prop
alt_(struct-key) 504 {#1}{\int_use:N \c@g__tag_struct_abs_int}
actualtext_(struct-key) 505 \__tag_property_record:eV
lang_(struct-key) 506 {tagpdfstruct-#1}
ref_(struct-key) 507 \c__tag_property_struct_clist
E_(struct-key) 508 },
509 stash .bool_set:N = \l__tag_struct_elem_stash_bool,
510 parent .code:n =
511 {
512 \bool_lazy_and:nnTF
513 {
514 \prop_if_exist_p:c { g__tag_struct_\int_eval:n {#1}_prop }
515 }
516 {
517 \int_compare_p:nNn {#1}<{\c@g__tag_struct_abs_int}
518 }
519 { \tl_set:Ne \l__tag_struct_stack_parent_tpa_tl { \int_eval:n {#1} } }
520 {
521 \msg_warning:nnee { tag } { struct-unknown }
522 { \int_eval:n {#1} }
523 { parent~key~ignored }
524 }
525 },
526 parent .default:n = {-1},
527 tag .code:n = % S property
528 {
529 \socket_use:nn { tag/struct/tag }{#1}
530 },
531 title .code:n = % T property
532 {
533 \str_set_convert:Nnnn
534 \l__tag_tpa_str
535 { #1 }
536 { default }
537 { utf16/hex }
538 \__tag_struct_prop_gput:nne
539 { \int_use:N \c@g__tag_struct_abs_int }
540 { T }
541 { <\l__tag_tpa_str> }
542 },
543 title-o .code:n = % T property
544 {
545 \str_set_convert:Nonn
546 \l__tag_tpa_str
547 { #1 }
548 { default }
549 { utf16/hex }
550 \__tag_struct_prop_gput:nne
551 { \int_use:N \c@g__tag_struct_abs_int }

```

```

552         { T }
553         { <\l__tag_tmpa_str> }
554     },
555     alt .code:n      = % Alt property
556     {
557         \tl_if_empty:oF{#1}
558         {
559             \str_set_convert:Noon
560             \l__tag_tmpa_str
561             { #1 }
562             { default }
563             { utf16/hex }
564             \__tag_struct_prop_gput:nne
565             { \int_use:N \c@g__tag_struct_abs_int }
566             { Alt }
567             { <\l__tag_tmpa_str> }
568         }
569     },
570     alttext .meta:n = {alt=#1},
571     actualtext .code:n = % ActualText property
572     {
573         \tl_if_empty:oF{#1}
574         {
575             \str_set_convert:Noon
576             \l__tag_tmpa_str
577             { #1 }
578             { default }
579             { utf16/hex }
580             \__tag_struct_prop_gput:nne
581             { \int_use:N \c@g__tag_struct_abs_int }
582             { ActualText }
583             { <\l__tag_tmpa_str>}
584         }
585     },
586     lang .code:n      = % Lang property
587     {
588         \__tag_struct_prop_gput:nne
589         { \int_use:N \c@g__tag_struct_abs_int }
590         { Lang }
591         { (#1) }
592     },
593 }

```

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.

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

```

594 \cs_new_protected:Npn \__tag_struct_Ref_obj:nN #1 #2 %#1 a object reference
595 {
596     \tl_put_right:Ne#2

```

```

597     {
598     \c_space_tl#1
599     }
600 }
601
602 \cs_new_protected:Npn \__tag_struct_Ref_label:nN #1 #2 %#1 a label
603 {
604   \prop_get:NnNTF \g__tag_struct_label_num_prop {#1} \l__tag_tmpb_tl
605   {
606     \tl_put_right:Ne#2
607     {
608       \c_space_tl\tag_struct_object_ref:e{ \l__tag_tmpb_tl }
609     }
610   }
611   {
612     \msg_warning:nnn {tag}{struct-Ref-unknown}{Label~'#1'}
613   }
614 }
615 \cs_new_protected:Npn \__tag_struct_Ref_dest:nN #1 #2 %#1 a dest name
616 {
617   \prop_get:NnNTF \g__tag_struct_dest_num_prop {#1} \l__tag_tmpb_tl
618   {
619     \tl_put_right:Ne#2
620     {
621       \c_space_tl\tag_struct_object_ref:e{ \l__tag_tmpb_tl }
622     }
623   }
624   {
625     \msg_warning:nnn {tag}{struct-Ref-unknown}{Destination~'#1'}
626   }
627 }
628 \cs_new_protected:Npn \__tag_struct_Ref_num:nN #1 #2 %#1 a structure number
629 {
630   \tl_put_right:Ne#2
631   {
632     \c_space_tl\tag_struct_object_ref:e{ #1 }
633   }
634 }
635
636 \keys_define:nn { __tag / struct }
637 {
638   ref .code:n      = % ref property
639   {
640     \clist_map_inline:on {#1}
641     {
642       \tag_struct_gput:nne
643       {\int_use:N \c@g__tag_struct_abs_int}{ref_label}{ ##1 }
644     }
645   },
646   E .code:n      = % E property
647   {
648     \str_set_convert:Nnon
649     \l__tag_tmpa_str
650     { #1 }

```



```

651         { default }
652         { utf16/hex }
653     \__tag_struct_prop_gput:nne
654     { \int_use:N \c@g__tag_struct_abs_int }
655     { E }
656     { <\l__tag_tmpa_str> }
657 },
658 }

```

(End of definition for label (struct-key) and others. These functions are documented on page 101.)

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

This variable is used to number the AF-object names

```

659 \int_new:N\g__tag_struct_AFobj_int

\g__tag_struct_AFobj_int
660 \cs_generate_variant:Nn \pdffile_embed_stream:nnN {neN}
661 \cs_new_protected:Npn \__tag_struct_add_inline_AF:nn #1 #2
662 % #1 content, #2 extension
663 {
664     \tl_if_empty:nF{#1}
665     {
666         \group_begin:
667         \int_gincr:N \g__tag_struct_AFobj_int
668         \pdffile_embed_stream:neN
669         {#1}
670         {tag-AFfile\int_use:N\g__tag_struct_AFobj_int.#2}
671         \l__tag_tmpa_tl
672         \__tag_struct_add_AF:ee
673         { \int_use:N \c@g__tag_struct_abs_int }
674         { \l__tag_tmpa_tl }
675         \__tag_struct_prop_gput:nne
676         { \int_use:N \c@g__tag_struct_abs_int }
677         { AF }
678         {
679             [
680                 \tl_use:c
681                 { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
682             ]
683         }
684         \group_end:
685     }
686 }

687
688 \cs_generate_variant:Nn \__tag_struct_add_inline_AF:nn {on}
689 \cs_new_protected:Npn \__tag_struct_add_AF:nn #1 #2 % #1 struct num #2 object reference
690 {
691     \tl_if_exist:cTF
692     {

```

```

693     g__tag_struct_#1_AF_tl
694   }
695   {
696     \tl_gput_right:ce
697     { g__tag_struct_#1_AF_tl }
698     { \c_space_tl #2 }
699   }
700   {
701     \tl_new:c
702     { g__tag_struct_#1_AF_tl }
703     \tl_gset:ce
704     { g__tag_struct_#1_AF_tl }
705     { #2 }
706   }
707 }
708 \cs_generate_variant:Nn \__tag_struct_add_AF:nn {en,ee}
709 \keys_define:nn { __tag / struct }
710 {
711   AF .code:n          = % AF property
712   {
713     \pdf_object_if_exist:eTF {#1}
714     {
715       \__tag_struct_add_AF:ee { \int_use:N \c@g__tag_struct_abs_int }{\pdf_object_ref:
716       \__tag_struct_prop_gput:nne
717       { \int_use:N \c@g__tag_struct_abs_int }
718       { AF }
719       {
720         [
721           \tl_use:c
722           { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
723         ]
724       }
725     }
726     {
727       % message?
728     }
729   },
730   AFref .code:n      = % AF property
731   {
732     \tl_if_empty:eF {#1}
733     {
734       \__tag_struct_add_AF:ee { \int_use:N \c@g__tag_struct_abs_int }{#1}
735       \__tag_struct_prop_gput:nne
736       { \int_use:N \c@g__tag_struct_abs_int }
737       { AF }
738       {
739         [
740           \tl_use:c
741           { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
742         ]
743       }
744     }
745   },
746   ,AFinline .code:n =

```

```

747     {
748       \__tag_struct_add_inline_AF:nn {#1}{txt}
749     }
750   ,AFinline-o .code:n =
751     {
752       \__tag_struct_add_inline_AF:on {#1}{txt}
753     }
754   ,texsource .code:n =
755     {
756       \group_begin:
757       \pdfdict_put:nnn { l_pdffile/Filespec } {Desc}{(TeX~source)}
758       \pdfdict_put:nnn { l_pdffile/Filespec }{AFRelationship} { /Source }
759       \__tag_struct_add_inline_AF:on {#1}{tex}
760       \group_end:
761     }
762   ,mathml .code:n =
763     {
764       \group_begin:
765       \pdfdict_put:nnn { l_pdffile/Filespec } {Desc}{(mathml~representation)}
766       \pdfdict_put:nnn { l_pdffile/Filespec }{AFRelationship} { /Supplement }
767       \__tag_struct_add_inline_AF:on {#1}{xml}
768       \group_end:
769     }
770 }

```

(End of definition for AF (struct-key) and others. These functions are documented on page 102.)

**root-AF<sub>□</sub>(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!

```

771 \keys_define:nn { __tag / setup }
772 {
773   root-AF .code:n =
774   {
775     \pdf_object_if_exist:nTF {#1}
776     {
777       \__tag_struct_add_AF:ee { 1 }{\pdf_object_ref:n {#1}}
778       \__tag_struct_prop_gput:nne
779         { 1 }
780         { AF }
781         {
782           [
783             \tl_use:c
784             { g__tag_struct_1_AF_tl }
785           ]
786         }
787       }
788     }
789   }
790 }
791 },
792 }

```

(End of definition for root-AF (setup-key). This function is documented on page 103.)

## 6 User commands

We allow to set a language by default

```
\l__tag_struct_lang_tl
```

```
793 \tl_new:N \l__tag_struct_lang_tl
794 </package>
```

(End of definition for \l\_\_tag\_struct\_lang\_tl.)

```
\tag_struct_begin:n
```

```
\tag_struct_end:
```

```
795 <base>\cs_new_protected:Npn \tag_struct_begin:n #1 {\int_gincr:N \c@g__tag_struct_abs_int}
796 <base>\cs_new_protected:Npn \tag_struct_end: {}
797 <base>\cs_new_protected:Npn \tag_struct_end:n {}
798 <*package | debug>
799 <package>\cs_set_protected:Npn \tag_struct_begin:n #1 %#1 key-val
800 <debug>\cs_set_protected:Npn \tag_struct_end:n #1 %#1 key-val
801 {
802 <package>\__tag_check_if_active_struct:T
803 <debug>\__tag_check_if_active_struct:TF
804 {
805     \group_begin:
806     \int_gincr:N \c@g__tag_struct_abs_int
807     \__tag_prop_new:c { g__tag_struct_\int_eval:n { \c@g__tag_struct_abs_int }_prop }
808 <debug>     \prop_new:c { g__tag_struct_debug_\int_eval:n { \c@g__tag_struct_abs_int }_prop
809     \__tag_new_output_prop_handler:n {\int_eval:n { \c@g__tag_struct_abs_int }}
810     \__tag_seq_new:c { g__tag_struct_kids_\int_eval:n { \c@g__tag_struct_abs_int }_seq}
811 <debug>     \seq_new:c { g__tag_struct_debug_kids_\int_eval:n { \c@g__tag_struct_abs_int }_
812     \pdf_object_new_indexed:nn { __tag/struct }
813     { \c@g__tag_struct_abs_int }
814     \__tag_struct_prop_gput:nnn
815     { \int_use:N \c@g__tag_struct_abs_int }
816     { Type }
817     { /StructElem }
818     \tl_if_empty:NF \l__tag_struct_lang_tl
819     {
820     \__tag_struct_prop_gput:nne
821     { \int_use:N \c@g__tag_struct_abs_int }
822     { Lang }
823     { (\l__tag_struct_lang_tl) }
824     }
825     \__tag_struct_prop_gput:nnn
826     { \int_use:N \c@g__tag_struct_abs_int }
827     { Type }
828     { /StructElem }
829
830     \tl_set:Nn \l__tag_struct_stack_parent_tmpa_tl {-1}
831     \keys_set:nn { __tag / struct } { #1 }
832
833     \__tag_struct_set_tag_info:eVV
834     { \int_use:N \c@g__tag_struct_abs_int }
835     \g__tag_struct_tag_tl
836     \g__tag_struct_tag_NS_tl
837     \__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.

```

837     \int_compare:nNtT { \l__tag_struct_stack_parent_tmpa_tl } = { -1 }
838     {
839         \seq_get:NNF
840         \g__tag_struct_stack_seq
841         \l__tag_struct_stack_parent_tmpa_tl
842         {
843             \msg_error:nn { tag } { struct-faulty-nesting }
844         }
845     }
846     \seq_gpush:NV \g__tag_struct_stack_seq          \c@g__tag_struct_abs_int
847     \__tag_role_get:VVNN
848     \g__tag_struct_tag_tl
849     \g__tag_struct_tag_NS_tl
850     \l__tag_struct_roletag_tl
851     \l__tag_struct_roletag_NS_tl

```

to target role and role NS

```

852     \__tag_struct_prop_gput:nne
853     { \int_use:N \c@g__tag_struct_abs_int }
854     { rolemap }
855     {
856         {\l__tag_struct_roletag_tl}{\l__tag_struct_roletag_NS_tl}
857     }

```

we also store which role to use for parent/child test. If the role is one of Part, Div, NonStruct we have to retrieve it from the parent. If the structure is stashed, this must be updated!

```

858     \str_case:VnTF \l__tag_struct_roletag_tl
859     {
860         {Part} {}
861         {Div} {}
862         {NonStruct} {}
863     }
864     {
865         \prop_get:cnNT
866         { g__tag_struct_ \l__tag_struct_stack_parent_tmpa_tl _prop }
867         { parentrole }
868         \l__tag_get_tmpc_tl
869         {
870             \__tag_struct_prop_gput:nno
871             { \int_use:N \c@g__tag_struct_abs_int }
872             { parentrole }
873             {
874                 \l__tag_get_tmpc_tl
875             }
876         }
877     }
878     {
879         \__tag_struct_prop_gput:nne
880         { \int_use:N \c@g__tag_struct_abs_int }
881         { parentrole }
882         {

```

```

883         {\l__tag_struct_roletag_tl}{\l__tag_struct_roletag_NS_tl}
884     }
885 }
886 \seq_gpush:Ne \g__tag_struct_tag_stack_seq
887   {\g__tag_struct_tag_tl}{\l__tag_struct_roletag_tl}}
888 \tl_gset:NV \g__tag_struct_stack_current_tl \c@g__tag_struct_abs_int
889 %\seq_show:N \g__tag_struct_stack_seq
890 \bool_if:NF
891   \l__tag_struct_elem_stash_bool
892   {

```

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. For now we ignore the namespace!

```

893   \__tag_struct_get_parentrole:eNN
894     {\l__tag_struct_stack_parent_tmpa_tl}
895     \l__tag_get_parent_tmpa_tl
896     \l__tag_get_parent_tmpb_tl
897   \__tag_check_parent_child:VVVVN
898     \l__tag_get_parent_tmpa_tl
899     \l__tag_get_parent_tmpb_tl
900     \g__tag_struct_tag_tl
901     \g__tag_struct_tag_NS_tl
902     \l__tag_parent_child_check_tl
903   \int_compare:nNnT {\l__tag_parent_child_check_tl}<0
904     {
905       \prop_get:cnN
906         { g__tag_struct_ \l__tag_struct_stack_parent_tmpa_tl _prop}
907         {S}
908         \l__tag_tmpa_tl
909       \quark_if_no_value:NT\l__tag_tmpa_tl{\tl_set:Nn \l__tag_tmpa_tl{UNKNOWN}}
910       \msg_warning:nneee
911         { tag }
912         {role-parent-child}
913         { \l__tag_get_parent_tmpa_tl/\l__tag_get_parent_tmpb_tl }
914         { \g__tag_struct_tag_tl/\g__tag_struct_tag_NS_tl }
915         { not~allowed~
916           (struct~\l__tag_struct_stack_parent_tmpa_tl,~\l__tag_tmpa_tl
917             \c_space_tl-->~struct~\int_eval:n {\c@g__tag_struct_abs_int})
918         }
919       \cs_set_eq:NN \l__tag_role_remap_tag_tl \g__tag_struct_tag_tl
920       \cs_set_eq:NN \l__tag_role_remap_NS_tl \g__tag_struct_tag_NS_tl
921       \__tag_role_remap:
922       \cs_gset_eq:NN \g__tag_struct_tag_tl \l__tag_role_remap_tag_tl
923       \cs_gset_eq:NN \g__tag_struct_tag_NS_tl \l__tag_role_remap_NS_tl
924       \__tag_struct_set_tag_info:eVV
925         { \int_use:N \c@g__tag_struct_abs_int }
926         \g__tag_struct_tag_tl
927         \g__tag_struct_tag_NS_tl
928     }

```

Set the Parent.

```

929   \__tag_struct_prop_gput:nne
930   { \int_use:N \c@g__tag_struct_abs_int }

```

```

931         { P }
932         {
933             \pdf_object_ref_indexed:nn { __tag/struct} { \l__tag_struct_stack_parent_tmpa_tl
934         }
935
936         %record this structure as kid:
937         %\tl_show:N \g__tag_struct_stack_current_tl
938         %\tl_show:N \l__tag_struct_stack_parent_tmpa_tl
939         \__tag_struct_kid_struct_gput_right:ee
940         { \l__tag_struct_stack_parent_tmpa_tl }
941         { \g__tag_struct_stack_current_tl }
942         %\prop_show:c { g__tag_struct_\g__tag_struct_stack_current_tl _prop }
943         %\seq_show:c {g__tag_struct_kids_\l__tag_struct_stack_parent_tmpa_tl _seq}
944     }

```

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.

```

944 <debug>         \prop_gset_eq:cc
945 <debug>         { g__tag_struct_debug_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
946 <debug>         { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
947 <debug>         \prop_gput:cne
948 <debug>         { g__tag_struct_debug_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
949 <debug>         { P }
950 <debug>         {
951             \bool_if:NTF \l__tag_struct_elem_stash_bool
952             {no-parent:~stashed}
953             {
954                 parent~structure:~\l__tag_struct_stack_parent_tmpa_tl\c_space_tl =~
955                 \prop_item:cn{ g__tag_struct_\l__tag_struct_stack_parent_tmpa_tl _p
956             }
957         }
958 <debug>         \prop_gput:cne
959 <debug>         { g__tag_struct_debug_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
960 <debug>         { NS }
961 <debug>         { \g__tag_struct_tag_NS_tl }
962
963         %\prop_show:c { g__tag_struct_\g__tag_struct_stack_current_tl _prop }
964         %\seq_show:c {g__tag_struct_kids_\l__tag_struct_stack_parent_tmpa_tl _seq}
965 <debug> \__tag_debug_struct_begin_insert:n { #1 }
966         \group_end:
967     }
968 <debug>{ \__tag_debug_struct_begin_ignore:n { #1 }}
969 }
970 <package>\cs_set_protected:Nn \tag_struct_end:
971 <debug>\cs_set_protected:Nn \tag_struct_end:
972     { %take the current structure num from the stack:
973       %the objects are written later, lua mode hasn't all needed info yet
974       %\seq_show:N \g__tag_struct_stack_seq
975 <package>\__tag_check_if_active_struct:T
976 <debug>\__tag_check_if_active_struct:TF
977     {
978         \seq_gpop:NN \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
979         \seq_gpop:NNTF \g__tag_struct_stack_seq \l__tag_tmpa_tl
980     }

```

```

980     \tag_struct_stack_current_tl }
981   }
982   { \tag_check_no_open_struct: }
983   % get the previous one, shouldn't be empty as the root should be there
984   \seq_get:NNTF \g__tag_struct_stack_seq \l__tag_tmpa_tl
985   {
986     \tl_gset:NV \g__tag_struct_stack_current_tl \l__tag_tmpa_tl
987   }
988   {
989     \tag_check_no_open_struct:
990   }
991   \seq_get:NNT \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
992   {
993     \tl_gset:Ne \g__tag_struct_tag_tl
994     { \exp_last_unbraced:NV\use_i:nn \l__tag_tmpa_tl }
995     \prop_get:NVNT\g__tag_role_tags_NS_prop \g__tag_struct_tag_tl\l__tag_tmpa_tl
996     {
997       \tl_gset:Ne \g__tag_struct_tag_NS_tl { \l__tag_tmpa_tl }
998     }
999   }
1000 <debug>\tag_debug_struct_end_insert:
1001   }
1002 <debug>{\tag_debug_struct_end_ignore:}
1003   }
1004
1005 \cs_set_protected:Npn \tag_struct_end:n #1
1006 {
1007 <debug> \tag_check_if_active_struct:T{\tag_debug_struct_end_check:n{#1}}
1008   \tag_struct_end:
1009 }
1010 </package | debug>

```

(End of definition for \tag\_struct\_begin:n and \tag\_struct\_end:. These functions are documented on page 100.)

**\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.

```

1011 <base>\cs_new_protected:Npn \tag_struct_use:n #1 {}
1012 <*package | debug>
1013 \cs_set_protected:Npn \tag_struct_use:n #1 %#1 is the label
1014 {
1015   \tag_check_if_active_struct:T
1016   {
1017     \prop_if_exist:cTF
1018     { g__tag_struct_\property_ref:enn{tagpdfstruct-#1}{tagstruct}{unknown}_prop } %
1019     {
1020       \tag_check_struct_used:n {#1}
1021       %add the label structure as kid to the current structure (can be the root)
1022       \tag_struct_kid_struct_gput_right:ee
1023       { \g__tag_struct_stack_current_tl }
1024       { \property_ref:enn{tagpdfstruct-#1}{tagstruct}{1} }
1025       %add the current structure to the labeled one as parents
1026       \tag_prop_gput:cne
1027       { g__tag_struct_\property_ref:enn{tagpdfstruct-#1}{tagstruct}{1}_prop }

```



```

1028         { P }
1029         {
1030         \pdf_object_ref_indexed:nn { __tag/struct } { \g__tag_struct_stack_current_tl
1031         }

```

debug code

```

1032 <debug>         \prop_gput:cne
1033 <debug>         { g__tag_struct_debug_ \property_ref:enn{tagpdfstruct-#1}{tagstruct}{1}_pr
1034 <debug>         { P }
1035 <debug>         {
1036 <debug>         parent~structure:~\g__tag_struct_stack_current_tl\c_space_tl=~
1037 <debug>         \g__tag_struct_tag_tl
1038 <debug>         }

```

check if the tag is allowed as child. Here we have to retrieve the tag info for the child, while the data for the parent is in the global tl-vars:

```

1039         \__tag_struct_get_parentrole:eNN
1040         {\property_ref:enn{tagpdfstruct-#1}{tagstruct}{1}}
1041         \l__tag_tmpa_tl
1042         \l__tag_tmpb_tl
1043         \__tag_check_parent_child:VVVVN
1044         \g__tag_struct_tag_tl
1045         \g__tag_struct_tag_NS_tl
1046         \l__tag_tmpa_tl
1047         \l__tag_tmpb_tl
1048         \l__tag_parent_child_check_tl
1049         \int_compare:nNnT {\l__tag_parent_child_check_tl}<0
1050         {
1051         \cs_set_eq:NN \l__tag_role_remap_tag_tl \g__tag_struct_tag_tl
1052         \cs_set_eq:NN \l__tag_role_remap_NS_tl \g__tag_struct_tag_NS_tl
1053         \__tag_role_remap:
1054         \cs_gset_eq:NN \g__tag_struct_tag_tl \l__tag_role_remap_tag_tl
1055         \cs_gset_eq:NN \g__tag_struct_tag_NS_tl \l__tag_role_remap_NS_tl
1056         \__tag_struct_set_tag_info:eVV
1057         { \int_use:N \c@g__tag_struct_abs_int }
1058         \g__tag_struct_tag_tl
1059         \g__tag_struct_tag_NS_tl
1060         }
1061         }
1062         {
1063         \msg_warning:nnn{ tag }{struct-label-unknown}{#1}
1064         }
1065     }
1066 }
1067 </package | debug>

```

(End of definition for `\tag_struct_use:n`. This function is documented on page 100.)

`\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.

```

1068 <base>\cs_new_protected:Npn \tag_struct_use_num:n #1 {}
1069 <*package | debug>
1070 \cs_set_protected:Npn \tag_struct_use_num:n #1 %#1 is structure number
1071 {

```

```

1072 \__tag_check_if_active_struct:T
1073 {
1074     \prop_if_exist:cTF
1075     { g__tag_struct_#1_prop } %
1076     {
1077         \prop_get:cnNT
1078         {g__tag_struct_#1_prop}
1079         {P}
1080         \l__tag_tmpa_tl
1081         {
1082             \msg_warning:nnn { tag } {struct-used-twice} {#1}
1083         }
1084         %add the \#1 structure as kid to the current structure (can be the root)
1085         \__tag_struct_kid_struct_gput_right:ee
1086         { \g__tag_struct_stack_current_tl }
1087         { #1 }
1088         %add the current structure to \#1 as parent
1089         \__tag_struct_prop_gput:nne
1090         { #1 }
1091         { P }
1092         {
1093             \pdf_object_ref_indexed:nn { __tag/struct }{ \g__tag_struct_stack_current_tl
1094         }
1095         <debug> \prop_gput:cne
1096         <debug> { g__tag_struct_debug_#1_prop }
1097         <debug> { P }
1098         <debug> {
1099             parent~structure:~\g__tag_struct_stack_current_tl\c_space_tl=~
1100         <debug> \g__tag_struct_tag_tl
1101         <debug> }

```

check if the tag is allowed as child. Here we have to retrieve the tag info for the child, while the data for the parent is in the global tl-vars:

```

1102     \__tag_struct_get_parentrole:eNN
1103     {#1}
1104     \l__tag_tmpa_tl
1105     \l__tag_tmpb_tl
1106     \__tag_check_parent_child:VVVVN
1107     \g__tag_struct_tag_tl
1108     \g__tag_struct_tag_NS_tl
1109     \l__tag_tmpa_tl
1110     \l__tag_tmpb_tl
1111     \l__tag_parent_child_check_tl
1112     \int_compare:nNnT {\l__tag_parent_child_check_tl}<0
1113     {
1114         \cs_set_eq:NN \l__tag_role_remap_tag_tl \g__tag_struct_tag_tl
1115         \cs_set_eq:NN \l__tag_role_remap_NS_tl \g__tag_struct_tag_NS_tl
1116         \__tag_role_remap:
1117         \cs_gset_eq:NN \g__tag_struct_tag_tl \l__tag_role_remap_tag_tl
1118         \cs_gset_eq:NN \g__tag_struct_tag_NS_tl \l__tag_role_remap_NS_tl
1119         \__tag_struct_set_tag_info:eVV
1120         { \int_use:N \c@g__tag_struct_abs_int }
1121         \g__tag_struct_tag_tl
1122         \g__tag_struct_tag_NS_tl

```

```

1123     }
1124   }
1125   {
1126     \msg_warning:nnn{ tag }{struct-label-unknown}{#1}
1127   }
1128 }
1129 }
1130 \</package | debug>

```

(End of definition for `\tag_struct_use_num:n`. This function is documented on page 100.)

`\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.

```

1131 <*package>
1132 \cs_new:Npn \tag_struct_object_ref:n #1
1133 {
1134   \pdf_object_ref_indexed:nn {__tag/struct}{ #1 }
1135 }
1136 \cs_generate_variant:Nn \tag_struct_object_ref:n {e}
1137 \</package>

```

(End of definition for `\tag_struct_object_ref:n`. This function is documented on page 100.)

`\tag_struct_gput:nnn` This is a command that allows to update the data of a structure. This often can't done simply by replacing the value, as we have to preserve and extend existing content. We use therefore dedicated functions adjusted to the key in question. The first argument is the number of the structure, the second a keyword referring to a function, the third the value. Currently the existing keywords are all 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. At last the keyword `ref_num` expects a structure number.

```

1138 <base>\cs_new_protected:Npn \tag_struct_gput:nnn #1 #2 #3{ }
1139 <*package>
1140 \cs_set_protected:Npn \tag_struct_gput:nnn #1 #2 #3
1141 {
1142   \cs_if_exist_use:cF {__tag_struct_gput_data_#2:nn}
1143   { %warning??
1144     \use_none:nn
1145   }
1146   {#1}{#3}
1147 }
1148 \cs_generate_variant:Nn \tag_struct_gput:nnn {ene,nne}
1149 \</package>

```

(End of definition for `\tag_struct_gput:nnn`. This function is documented on page 101.)

`\__tag_struct_gput_data_ref_aux:nnn`

```

1150 <*package>
1151 \cs_new_protected:Npn \__tag_struct_gput_data_ref_aux:nnn #1 #2 #3
1152 % #1 receiving struct num, #2 key word #3 value
1153 {

```

```

1154 \prop_get:cnNTF
1155   { g__tag_struct_#1_prop }
1156   {Ref}
1157   \l__tag_get_tmpc_tl
1158   {
1159     \tl_put_right:No \l__tag_get_tmpc_tl
1160     {\cs:w __tag_struct_Ref_#2:nN \cs_end: {#3},}
1161   }
1162   {
1163     \tl_set:No \l__tag_get_tmpc_tl
1164     {\cs:w __tag_struct_Ref_#2:nN \cs_end: {#3},}
1165   }
1166   \__tag_struct_prop_gput:nno
1167     { #1 }
1168     { Ref }
1169     { \l__tag_get_tmpc_tl }
1170   }
1171 \cs_new_protected:Npn \__tag_struct_gput_data_ref:nn #1 #2
1172   {
1173     \__tag_struct_gput_data_ref_aux:nnn {#1}{obj}{#2}
1174   }
1175 \cs_new_protected:Npn \__tag_struct_gput_data_ref_label:nn #1 #2
1176   {
1177     \__tag_struct_gput_data_ref_aux:nnn {#1}{label}{#2}
1178   }
1179 \cs_new_protected:Npn \__tag_struct_gput_data_ref_dest:nn #1 #2
1180   {
1181     \__tag_struct_gput_data_ref_aux:nnn {#1}{dest}{#2}
1182   }
1183 \cs_new_protected:Npn \__tag_struct_gput_data_ref_num:nn #1 #2
1184   {
1185     \__tag_struct_gput_data_ref_aux:nnn {#1}{num}{#2}
1186   }
1187
1188 \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\_insert\_annot:nn** This are the user command to insert annotations. They must be used together to get the numbers right. They use a counter to the StructParent and \tag\_struct\_insert\_annot:nn increases the counter given back by \tag\_struct\_parent\_int:.

**\tag\_struct\_insert\_annot:ee** It must be used together with \tag\_struct\_parent\_int: to insert an annotation. TODO: decide how it should be guarded if tagging is deactivated.

**\tag\_struct\_parent\_int:**

```

1189 \cs_new_protected:Npn \tag_struct_insert_annot:nn #1 #2 %#1 should be an object reference
1190                                     %#2 struct parent num
1191   {
1192     \__tag_check_if_active_struct:T
1193     {
1194       \__tag_struct_insert_annot:nn {#1}{#2}
1195     }
1196   }
1197
1198 \cs_generate_variant:Nn \tag_struct_insert_annot:nn {xx,ee}
1199 \cs_new:Npn \tag_struct_parent_int: {\int_use:c { c@g__tag_parenttree_obj_int }}

```

```

1200
1201 </package>
1202

```

(End of definition for `\tag_struct_insert_annot:nn` and `\tag_struct_parent_int:.`. These functions are documented on page 100.)

## 7 Attributes and attribute classes

```

1203 <*header>
1204 \ProvidesExplPackage {tagpdf-attr-code} {2024-09-11} {0.99e}
1205   {part of tagpdf - code related to attributes and attribute classes}
1206 </header>

```

### 7.1 Variables

`\g__tag_attr_entries_prop` will store attribute names and their dictionary content.  
`\g__tag_attr_class_used_prop` will hold the attributes which have been used as class name.  
`\l__tag_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__tag_attr_objref_prop`

```

1207 <*package>
1208 \prop_new:N \g__tag_attr_entries_prop
1209 \prop_new_linked:N \g__tag_attr_class_used_prop
1210 \tl_new:N \l__tag_attr_value_tl
1211 \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.

```

1212 \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

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},
}

1213 \cs_new_protected:Npn \__tag_attr_new_entry:nn #1 #2 %#1:name, #2: content
1214   {
1215     \prop_gput:Nen \g__tag_attr_entries_prop
1216       {\pdf_name_from_unicode_e:n{#1}}{#2}
1217   }
1218
1219 \cs_generate_variant:Nn \__tag_attr_new_entry:nn {ee}

```

```

1220 \keys_define:nn { __tag / setup }
1221 {
1222   role/new-attribute .code:n =
1223   {
1224     \__tag_attr_new_entry:nn #1
1225   }

```

deprecated name

```

1226   ,newattribute .code:n =
1227   {
1228     \__tag_attr_new_entry:nn #1
1229   },
1230 }

```

(End of definition for `\__tag_attr_new_entry:nn`, `role/new-attribute (setup-key)`, and `newattribute (deprecated)`. These functions are documented on page 103.)

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

```

1231 \keys_define:nn { __tag / struct }
1232 {
1233   attribute-class .code:n =
1234   {
1235     \clist_set:Ne \l__tag_tmpa_clist { #1 }
1236     \seq_set_from_clist:NN \l__tag_tmpb_seq \l__tag_tmpa_clist

```

we convert the names into pdf names with slash

```

1237     \seq_set_map_e:NNn \l__tag_tmpa_seq \l__tag_tmpb_seq
1238     {
1239       \pdf_name_from_unicode_e:n {##1}
1240     }
1241     \seq_map_inline:Nn \l__tag_tmpa_seq
1242     {
1243       \prop_if_in:NnF \g__tag_attr_entries_prop {##1}
1244       {
1245         \msg_error:nnn { tag } { attr-unknown } { ##1 }
1246       }
1247       \prop_gput:Nnn \g__tag_attr_class_used_prop { ##1 } {}
1248     }
1249     \tl_set:Ne \l__tag_tmpa_tl
1250     {
1251       \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}
1252       \seq_use:Nn \l__tag_tmpa_seq { \c_space_tl }
1253       \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}
1254     }
1255     \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 0 }
1256     {
1257       \__tag_struct_prop_gput:nne
1258       { \int_use:N \c@g__tag_struct_abs_int }
1259       { C }
1260       { \l__tag_tmpa_tl }
1261       %\prop_show:c { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
1262     }
1263   }
1264 }

```

(End of definition for attribute-class (struct-key). This function is documented on page 103.)

**attribute\_␣(struct-key)**

```

1265 \keys_define:nn { __tag / struct }
1266 {
1267   attribute .code:n = % A property (attribute, value currently a dictionary)
1268   {
1269     \clist_set:Ne          \l__tag_tmpa_clist { #1 }
1270     \clist_if_empty:NF \l__tag_tmpa_clist
1271     {
1272       \seq_set_from_clist:NN \l__tag_tmpb_seq \l__tag_tmpa_clist
we convert the names into pdf names with slash
1273       \seq_set_map_e:NnN \l__tag_tmpa_seq \l__tag_tmpb_seq
1274       {
1275         \pdf_name_from_unicode_e:n {##1}
1276       }
1277       \tl_set:Ne \l__tag_attr_value_tl
1278       {
1279         \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[{}%]
1280       }
1281       \seq_map_inline:Nn \l__tag_tmpa_seq
1282       {
1283         \prop_if_in:NnF \g__tag_attr_entries_prop {##1}
1284         {
1285           \msg_error:nnn { tag } { attr-unknown } { ##1 }
1286         }
1287         \prop_if_in:NnF \g__tag_attr_objref_prop {##1}
1288         {%\prop_show:N \g__tag_attr_entries_prop
1289           \pdf_object_unnamed_write:ne
1290           { dict }
1291           {
1292             \prop_item:Nn\g__tag_attr_entries_prop {##1}
1293           }
1294           \prop_gput:Nne \g__tag_attr_objref_prop {##1} {\pdf_object_ref_last:}
1295         }
1296         \tl_put_right:Ne \l__tag_attr_value_tl
1297         {
1298           \c_space_tl
1299           \prop_item:Nn \g__tag_attr_objref_prop {##1}
1300         }
1301       }
1302       % \tl_show:N \l__tag_attr_value_tl
1303       }
1304       \tl_put_right:Ne \l__tag_attr_value_tl
1305       { %[
1306         \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[{}%]
1307       }
1308       % \tl_show:N \l__tag_attr_value_tl
1309       \__tag_struct_prop_gput:nne
1310       { \int_use:N \c@g__tag_struct_abs_int }
1311       { A }
1312       { \l__tag_attr_value_tl }
1313     }
1314   },

```

```
1314 }  
1315 </package>
```

*(End of definition for attribute (struct-key). This function is documented on page 103.)*



## Part VIII

# The tagpdf-luatex.def Driver for luatex Part of the tagpdf package

```
1 <@@=tag>
2 <*luatex>
3 \ProvidesExplFile {tagpdf-luatex.def} {2024-09-11} {0.99e}
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_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
36 \cs_set_protected:Npn \__tag_seq_gput_right:Nn #1 #2
37 {
38   \seq_gput_right:Nn #1 { #2 }
39   \lua_now:e { table.insert(ltx.__tag.tables.\cs_to_str:N#1, "#2") }
40 }
41
42 %Hm not quite sure about the naming
43
44 \cs_set:Npn \__tag_seq_item:cn #1 #2
45 {
46   \lua_now:e { tex.print(ltx.__tag.tables.#1[#2]) }
47 }
48
49 \cs_set:Npn \__tag_prop_item:cn #1 #2
50 {
51   \lua_now:e { tex.print(ltx.__tag.tables.#1["#2"]) }
52 }
53
54 %for debugging commands that show both the seq/prop and the lua tables
55 \cs_set_protected:Npn \__tag_seq_show:N #1
56 {
57   \seq_show:N #1
58   \lua_now:e { ltx.__tag.trace.log ("lua~sequence~array~\cs_to_str:N#1",1) }
59   \lua_now:e { ltx.__tag.trace.show_seq (ltx.__tag.tables.\cs_to_str:N#1) }
60 }
61
62 \cs_set_protected:Npn \__tag_prop_show:N #1
63 {
64   \prop_show:N #1
65   \lua_now:e {ltx.__tag.trace.log ("lua~property~table~\cs_to_str:N#1",1) }
66   \lua_now:e {ltx.__tag.trace.show_prop (ltx.__tag.tables.\cs_to_str:N#1) }
67 }

```

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

```

68 </luatex>

```

The module declaration

```

69 <*lua>
70 -- tagpdf.lua
71 -- Ulrike Fischer
72
73 local ProvidesLuaModule = {
74   name      = "tagpdf",
75   version   = "0.99e",      --TAGVERSION
76   date      = "2024-09-11", --TAGDATE
77   description = "tagpdf lua code",
78   license   = "The LATEX Project Public License 1.3c"
79 }

```

```

80
81 if luatexbase and luatexbase.provides_module then
82   luatexbase.provides_module (ProvidesLuaModule)
83 end
84
85 --[[
86 The code has quite probably a number of problems
87 - more variables should be local instead of global
88 - the naming is not always consistent due to the development of the code
89 - the traversing of the shipout box must be tested with more complicated setups
90 - it should probably handle more node types
91 -
92 --]]
93

```

Some comments about the lua structure.

```

94 --[[
95 the main table is named ltx.__tag. It contains the functions and also the data
96 collected during the compilation.
97
98 ltx.__tag.mc      will contain mc connected data.
99 ltx.__tag.struct will contain structure related data.
100 ltx.__tag.page   will contain page data
101 ltx.__tag.tables contains also data from mc and struct (from older code). This needs cleaning
102       There are certainly dublettes, but I don't dare yet ...
103 ltx.__tag.func   will contain (public) functions.
104 ltx.__tag.trace  will contain tracing/logging functions.
105 local functions starts with __
106 functions meant for users will be in ltx.tag
107
108 functions
109 ltx.__tag.func.get_num_from (tag):   takes a tag (string) and returns the id number
110 ltx.__tag.func.output_num_from (tag): takes a tag (string) and prints (to tex) the id number
111 ltx.__tag.func.get_tag_from (num):   takes a num and returns the tag
112 ltx.__tag.func.output_tag_from (num): takes a num and prints (to tex) the tag
113 ltx.__tag.func.store_mc_data (num,key,data): stores key=data in ltx.__tag.mc[num]
114 ltx.__tag.func.store_mc_label (label,num): stores label=num in ltx.__tag.mc.labels
115 ltx.__tag.func.store_mc_kid (mcnum,kid,page): stores the mc-kids of mcnum on page page
116 ltx.__tag.func.store_mc_in_page(mcnum,mcpagecnt,page): stores in the page table the number of
117 ltx.__tag.func.store_struct_mcabs (structnum,mcnum): stores relations structnum<->mcnum (abs
118 ltx.__tag.func.mc_insert_kids (mcnum): inserts the /K entries for mcnum by wandering through
119 ltx.__tag.func.mark_page_elements(box,mcpagecnt,mccntprev,mcopen,name,mctypeprev) : the main
120 ltx.__tag.func.mark_shipout (): a wrapper around the core function which inserts the last EN
121 ltx.__tag.func.fill_parent_tree_line (page): outputs the entries of the parenttree for this
122 ltx.__tag.func.output_parenttree(): outputs the content of the parenttree
123 ltx.__tag.func.pdf_object_ref(name,index): outputs the object reference for the object name
124 ltx.__tag.func.markspaceon(), ltx.__tag.func.markspaceoff(): (de)activates the marking of po
125 ltx.__tag.trace.show_mc_data (num,loglevel): shows ltx.__tag.mc[num] is the current log leve
126 ltx.__tag.trace.show_all_mc_data (max,loglevel): shows a maximum about mc's if the current l
127 ltx.__tag.trace.show_seq: shows a sequence (array)
128 ltx.__tag.trace.show_struct_data (num): shows data of structure num
129 ltx.__tag.trace.show_prop: shows a prop
130 ltx.__tag.trace.log
131 ltx.__tag.trace.showspaces : boolean
132 --]]

```

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

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

```

134 local mctypeattributeid = luatexbase.new_attribute ("g__tag_mc_type_attr")
135 local mcntattributeid   = luatexbase.new_attribute ("g__tag_mc_cnt_attr")
136 local iwspaceOffattributeid = luatexbase.new_attribute ("g__tag_interwordspaceOff_attr")
137 local iwspaceattributeid = luatexbase.new_attribute ("g__tag_interwordspace_attr")
138 local iwfontattributeid  = luatexbase.new_attribute ("g__tag_interwordfont_attr")

```

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

```

139 local tagunmarkedbool= token.create("g__tag_tagunmarked_bool")
140 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.

```

141 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.

```

142 local catlatex      = luatexbase.registernumber("catcodetable@latex")
143 local tableinsert   = table.insert
144 local nodeid        = node.id
145 local nodecopy      = node.copy
146 local nodegetattribute = node.get_attribute
147 local nodesetattribute = node.set_attribute
148 local nodehasattribute = node.has_attribute
149 local nodenew       = node.new
150 local nodetail      = node.tail
151 local nodeslide     = node.slide
152 local noderemove    = node.remove
153 local nodetraverseid = node.traverse_id
154 local nodetraverse  = node.traverse
155 local nodeinsertafter = node.insert_after
156 local nodeinsertbefore = node.insert_before
157 local pdfpageref    = pdf.pageref
158
159 local fonthashes    = fonts.hashes
160 local identifiers   = fonthashes.identifiers
161 local fontid       = font.id
162
163 local HLIST        = node.id("hlist")
164 local VLIST        = node.id("vlist")
165 local RULE         = node.id("rule")
166 local DISC         = node.id("disc")
167 local GLUE         = node.id("glue")
168 local GLYPH        = node.id("glyph")
169 local KERN         = node.id("kern")

```

```

170 local PENALTY      = node.id("penalty")
171 local LOCAL_PAR    = node.id("local_par")
172 local MATH         = node.id("math")
173
174 local explicit_disc = 1
175 local regular_disc = 3

```

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

```

176 ltx          = ltx          or { }
177 ltx.__tag    = ltx.__tag    or { }
178 ltx.__tag.mc = ltx.__tag.mc or { } -- mc data
179 ltx.__tag.struct = ltx.__tag.struct or { } -- struct data
180 ltx.__tag.tables = ltx.__tag.tables or { } -- tables created with new prop and new seq.
181                                           -- wasn't a so great idea ...
182                                           -- g__tag_role_tags_seq used by tag<-> is in this table
183                                           -- used for pure lua tables too now!
184 ltx.__tag.page = ltx.__tag.page or { } -- page data, currently only i->{0->mcnum,1->mcnum}
185 ltx.__tag.trace = ltx.__tag.trace or { } -- show commands
186 ltx.__tag.func  = ltx.__tag.func  or { } -- functions
187 ltx.__tag.conf  = ltx.__tag.conf  or { } -- configuration variables

```

## 2 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`.

```

188 local __tag_log =
189   function (message,loglevel)
190     if (loglevel or 3) <= tex.count["l__tag_loglevel_int"] then
191       texio.write_nl("tagpdf: " .. message)
192     end
193   end
194
195 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`.

```

196 function ltx.__tag.trace.show_seq (seq)
197   if (type(seq) == "table") then
198     for i,v in ipairs(seq) do
199       __tag_log ("[" .. i .. "] => " .. tostring(v),1)
200     end
201   else
202     __tag_log ("sequence " .. tostring(seq) .. " not found",1)
203   end
204 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  
`ltx.__tag.trace.show_prop` \@@\_prop\_show:N function.

```

205 local __tag_pairs_prop =
206   function (prop)
207     local a = {}
208     for n in pairs(prop) do tableinsert(a, n) end
209     table.sort(a)
210     local i = 0           -- iterator variable
211     local iter = function () -- iterator function
212       i = i + 1
213       if a[i] == nil then return nil
214       else return a[i], prop[a[i]]
215     end
216   end
217   return iter
218 end
219
220
221 function ltx.__tag.trace.show_prop (prop)
222 if (type(prop) == "table") then
223   for i,v in __tag_pairs_prop (prop) do
224     __tag_log ("[" .. i .. "] => " .. tostring(v),1)
225   end
226 else
227   __tag_log ("prop " .. tostring(prop) .. " not found or not a table",1)
228 end
229 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

```

230 function ltx.__tag.trace.show_mc_data (num,loglevel)
231 if ltx.__tag and ltx.__tag.mc and ltx.__tag.mc[num] then
232   for k,v in pairs(ltx.__tag.mc[num]) do
233     __tag_log ("mc"..num..": " ..tostring(k)..=>" ..tostring(v),loglevel)
234   end
235   if ltx.__tag.mc[num]["kids"] then
236     __tag_log ("mc" .. num .. " has " .. #ltx.__tag.mc[num]["kids"] .. " kids",loglevel)
237     for k,v in ipairs(ltx.__tag.mc[num]["kids"]) do
238       __tag_log ("mc " .. num .. " kid "..k.." =>" .. v.kid.." on page " ..v.page,loglevel)
239     end
240   end
241 else
242   __tag_log ("mc"..num.." not found",loglevel)
243 end
244 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.

```

245 function ltx.__tag.trace.show_all_mc_data (min,max,loglevel)
246   for i = min, max do

```

```

247 ltx.__tag.trace.show_mc_data (i,loglevel)
248 end
249 texio.write_nl("")
250 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.

```

251 function ltx.__tag.trace.show_struct_data (num)
252 if ltx.__tag and ltx.__tag.struct and ltx.__tag.struct[num] then
253 for k,v in ipairs(ltx.__tag.struct[num]) do
254 __tag_log ("struct "..num..": "..tostring(k).."=>"..tostring(v),1)
255 end
256 else
257 __tag_log ("struct "..num.." not found ",1)
258 end
259 end

```

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

## 3 Helper functions

### 3.1 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.

```

260 local __tag_get_mc_cnt_type_tag = function (n)
261 local mcnt = nodegetattribute(n,mcntattributeid) or -1
262 local mctype = nodegetattribute(n,mctypeattributeid) or -1
263 local tag = ltx.__tag.func.get_tag_from(mctype)
264 return mcnt,mctype,tag
265 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.

```

266 local function __tag_get_mathsubtype (mathnode)
267 if mathnode.subtype == 0 then
268 subtype = "beginmath"
269 else
270 subtype = "endmath"
271 end
272 return subtype
273 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.

```

274 ltx.__tag.tables.role_tag_attribute = {}
275 ltx.__tag.tables.role_attribute_tag = {}

```

(End of definition for ltx.\_\_tag.tables.role\_tag\_attribute.)

ltx.\_\_tag.func.alloctag

```
276 local __tag_alloctag =
277   function (tag)
278     if not ltx.__tag.tables.role_tag_attribute[tag] then
279       table.insert(ltx.__tag.tables.role_attribute_tag,tag)
280       ltx.__tag.tables.role_tag_attribute[tag]=#ltx.__tag.tables.role_attribute_tag
281       __tag_log ("Add "..tag.." "..ltx.__tag.tables.role_tag_attribute[tag],3)
282     end
283   end
284 ltx.__tag.func.alloctag = __tag_alloctag
```

(End of definition for ltx.\_\_tag.func.alloctag.)

\_\_tag\_get\_num\_from

ltx.\_\_tag.func.get\_num\_from

ltx.\_\_tag.func.output\_num\_from

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

```
285 local __tag_get_num_from =
286   function (tag)
287     if ltx.__tag.tables.role_tag_attribute[tag] then
288       a = ltx.__tag.tables.role_tag_attribute[tag]
289     else
290       a = -1
291     end
292     return a
293   end
294
295 ltx.__tag.func.get_num_from = __tag_get_num_from
296
297 function ltx.__tag.func.output_num_from (tag)
298   local num = __tag_get_num_from (tag)
299   tex.sprint(catlatex,num)
300   if num == -1 then
301     __tag_log ("Unknown tag "..tag.." used")
302   end
303 end
```

(End of definition for \_\_tag\_get\_num\_from, ltx.\_\_tag.func.get\_num\_from, and ltx.\_\_tag.func.output\_num\_from.)

\_\_tag\_get\_tag\_from

ltx.\_\_tag.func.get\_tag\_from

ltx.\_\_tag.func.output\_tag\_from

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

```
304 local __tag_get_tag_from =
305   function (num)
306     if ltx.__tag.tables.role_attribute_tag[num] then
307       a = ltx.__tag.tables.role_attribute_tag[num]
308     else
309       a = "UNKNOWN"
310     end
311     return a
312   end
313
314 ltx.__tag.func.get_tag_from = __tag_get_tag_from
315
```



```

316 function ltx.__tag.func.output_tag_from (num)
317   tex.sprint(catlatex,__tag_get_tag_from (num))
318 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.

```

319 function ltx.__tag.func.store_mc_data (num,key,data)
320   ltx.__tag.mc[num] = ltx.__tag.mc[num] or { }
321   ltx.__tag.mc[num][key] = data
322   __tag_log ("INFO TEX-STORE-MC-DATA: "..num.." => "..tostring(key).. " => "..tostring(data),3)
323 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.

```

324 function ltx.__tag.func.store_mc_label (label,num)
325   ltx.__tag.mc["labels"] = ltx.__tag.mc["labels"] or { }
326   ltx.__tag.mc.labels[label] = num
327 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.

```

328 function ltx.__tag.func.store_mc_kid (mcnum,kid,page)
329   ltx.__tag.trace.log("INFO TAG-STORE-MC-KID: "..mcnum.." => " .. kid.." on page " .. page,3)
330   ltx.__tag.mc[mcnum]["kids"] = ltx.__tag.mc[mcnum]["kids"] or { }
331   local kidtable = {kid=kid,page=page}
332   tableinsert(ltx.__tag.mc[mcnum]["kids"], kidtable )
333 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.

```

334 function ltx.__tag.func.mc_num_of_kids (mcnum)
335   local num = 0
336   if ltx.__tag.mc[mcnum] and ltx.__tag.mc[mcnum]["kids"] then
337     num = #ltx.__tag.mc[mcnum]["kids"]
338   end
339   ltx.__tag.trace.log ("INFO MC-KID-NUMBERS: " .. mcnum .. "has " .. num .. "KIDS",4)
340   return num
341 end

```

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

## 3.2 Functions to insert the pdf literals

`__tag_backend_create_emc_node` This insert the emc node. We support also dvips and dvi<sub>pdf</sub>mx backend  
`__tag_insert_emc_node`

```
342 local __tag_backend_create_emc_node
343 if tex.outputmode == 0 then
344   if token.get_macro("c_sys_backend_str") == "dvipdfmx" then
345     function __tag_backend_create_emc_node ()
346       local emcnode = nodenew("whatsit","special")
347       emcnode.data = "pdf:code EMC"
348       return emcnode
349     end
350   else -- assume a dvips variant
351     function __tag_backend_create_emc_node ()
352       local emcnode = nodenew("whatsit","special")
353       emcnode.data = "ps:SDict begin mark /EMC pdfmark end"
354       return emcnode
355     end
356   end
357 else -- pdf mode
358   function __tag_backend_create_emc_node ()
359     local emcnode = nodenew("whatsit","pdf_literal")
360     emcnode.data = "EMC"
361     emcnode.mode=1
362     return emcnode
363   end
364 end
365
366 local function __tag_insert_emc_node (head,current)
367   local emcnode= __tag_backend_create_emc_node()
368   head = node.insert_before(head,current,emcnode)
369   return head
370 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

```
371 local __tag_backend_create_bmc_node
372 if tex.outputmode == 0 then
373   if token.get_macro("c_sys_backend_str") == "dvipdfmx" then
374     function __tag_backend_create_bmc_node (tag)
375       local bmcnode = nodenew("whatsit","special")
376       bmcnode.data = "pdf:code /"..tag.." BMC"
377       return bmcnode
378     end
379   else -- assume a dvips variant
380     function __tag_backend_create_bmc_node (tag)
381       local bmcnode = nodenew("whatsit","special")
382       bmcnode.data = "ps:SDict begin mark/"..tag.." /BMC pdfmark end"
383       return bmcnode
384     end
385   end
386 else -- pdf mode
387   function __tag_backend_create_bmc_node (tag)
388     local bmcnode = nodenew("whatsit","pdf_literal")
```

```

389     bmcnode.data = "/"..tag.." BMC"
390     bmcnode.mode=1
391     return bmcnode
392 end
393 end
394
395 local function __tag_insert_bmc_node (head,current,tag)
396 local bmcnode = __tag_backend_create_bmc_node (tag)
397 head = node.insert_before(head,current,bmcnode)
398 return head
399 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.

```

400 local __tag_backend_create_bdc_node
401
402 if tex.outputmode == 0 then
403 if token.get_macro("c_sys_backend_str") == "dvipdfmx" then
404 function __tag_backend_create_bdc_node (tag,dict)
405 local bdcnode = nodenew("whatsit","special")
406 bdcnode.data = "pdf:code "/"..tag.."<<"..dict..">> BDC"
407 return bdcnode
408 end
409 else -- assume a dvips variant
410 function __tag_backend_create_bdc_node (tag,dict)
411 local bdcnode = nodenew("whatsit","special")
412 bdcnode.data = "ps:SDict begin mark/"..tag.."<<"..dict..">> /BDC pdfmark end"
413 return bdcnode
414 end
415 end
416 else -- pdf mode
417 function __tag_backend_create_bdc_node (tag,dict)
418 local bdcnode = nodenew("whatsit","pdf_literal")
419 bdcnode.data = "/"..tag.."<<"..dict..">> BDC"
420 bdcnode.mode=1
421 return bdcnode
422 end
423 end
424
425 local function __tag_insert_bdc_node (head,current,tag,dict)
426 bdcnode= __tag_backend_create_bdc_node (tag,dict)
427 head = node.insert_before(head,current,bdcnode)
428 return head
429 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.

```

430 local function __tag_pdf_object_ref (name,index)
431 local object
432 if ltx.pdf.object_id then

```

```

433     object = ltx.pdf.object_id (name,index) ..' 0 R'
434   else
435     local tokename = 'c__pdf_object_'..name..'/'..index..'_int'
436     object = token.create(tokename).mode ..' 0 R'
437   end
438   return object
439 end
440 ltx.__tag.func.pdf_object_ref = __tag_pdf_object_ref

```

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

## 4 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.

```

441 local function __tag_show_spacemark (head,current,color,height)
442   local markcolor = color or "1 0 0"
443   local markheight = height or 10
444   local pdfstring
445   if tex.outputmode == 0 then
446     -- ignore dvi mode for now
447   else
448     pdfstring = node.new("whatsit","pdf_literal")
449     pdfstring.data =
450       string.format("q " ..markcolor.." RG " ..markcolor.." rg 0.4 w 0 %g m 0 %g l S Q",-
3,markheight)
451     head = node.insert_after(head,current,pdfstring)
452   return head
453 end
454 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`

```

455 local function __tag_fakespace()
456   tex.setattribute(iwspaceattributeid,1)
457   tex.setattribute(iwfontattributeid,font.current())
458 end
459 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.

```

460 --[[ a function to mark up places where real space chars should be inserted
461     it only sets an attribute.
462 --]]
463
464 local function __tag_mark_spaces (head)
465   local inside_math = false
466   for n in nodetraverse(head) do
467     local id = n.id

```

```

468 if id == GLYPH then
469     local glyph = n
470     default_currfontid = glyph.font
471     if glyph.next and (glyph.next.id == GLUE)
472         and not inside_math and (glyph.next.width >0)
473     then
474         nodesetattribute(glyph.next,iwspaceattributeid,1)
475         nodesetattribute(glyph.next,iwfontattributeid,glyph.font)
476     -- for debugging
477     if ltx.__tag.trace.showspace then
478         __tag_show_spacemark (head,glyph)
479     end
480     elseif glyph.next and (glyph.next.id==KERN) and not inside_math then
481         local kern = glyph.next
482         if kern.next and (kern.next.id== GLUE) and (kern.next.width >0)
483     then
484         nodesetattribute(kern.next,iwspaceattributeid,1)
485         nodesetattribute(kern.next,iwfontattributeid,glyph.font)
486     end
487     end
488     -- look also back
489     if glyph.prev and (glyph.prev.id == GLUE)
490         and not inside_math
491         and (glyph.prev.width >0)
492         and not nodehasattribute(glyph.prev,iwspaceattributeid)
493     then
494         nodesetattribute(glyph.prev,iwspaceattributeid,1)
495         nodesetattribute(glyph.prev,iwfontattributeid,glyph.font)
496     -- for debugging
497     if ltx.__tag.trace.showspace then
498         __tag_show_spacemark (head,glyph)
499     end
500     end
501 elseif id == PENALTY then
502     local glyph = n
503     -- ltx.__tag.trace.log ("PENALTY " .. n.subtype.."VALUE"..n.penalty,3)
504     if glyph.next and (glyph.next.id == GLUE)
505         and not inside_math and (glyph.next.width >0) and n.subtype==0
506     then
507         nodesetattribute(glyph.next,iwspaceattributeid,1)
508         -- changed 2024-01-18, issue #72
509         nodesetattribute(glyph.next,iwfontattributeid,default_currfontid)
510     -- for debugging
511     if ltx.__tag.trace.showspace then
512         __tag_show_spacemark (head,glyph)
513     end
514     end
515 elseif id == MATH then
516     inside_math = (n.subtype == 0)
517     end
518 end
519 return head
520 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
521 local function __tag_activate_mark_space ()
522   if not luatexbase.in_callback ("pre_linebreak_filter","markspaces") then
523     luatexbase.add_to_callback("pre_linebreak_filter",__tag_mark_spaces,"markspaces")
524     luatexbase.add_to_callback("hpack_filter",__tag_mark_spaces,"markspaces")
525   end
526 end
527
528 ltx.__tag.func.markspaceon=__tag_activate_mark_space
529
530 local function __tag_deactivate_mark_space ()
531   if luatexbase.in_callback ("pre_linebreak_filter","markspaces") then
532     luatexbase.remove_from_callback("pre_linebreak_filter","markspaces")
533     luatexbase.remove_from_callback("hpack_filter","markspaces")
534   end
535 end
536
537 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.

```

538 local default_space_char = nodenew(GLYPH)
539 local default_fontid      = fontid("TU/lmr/m/n/10")
540 local default_currfontid  = fontid("TU/lmr/m/n/10")
541 default_space_char.char   = 32
542 default_space_char.font   = default_fontid

```

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

```

543 local function __tag_font_has_space (fontid)
544   t= fonts.hashes.identifiers[fontid]
545   if luaotfload.aux.slot_of_name(fontid,"space")
546     or t.characters and t.characters[32] and t.characters[32]["unicode"]==32
547   then
548     return true
549   else
550     return false
551   end
552 end

```

`__tag_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.

```

553 local function __tag_space_chars_shipout (box)
554   local head = box.head
555   if head then
556     for n in node.traverse(head) do
557       local spaceattr = -1
558       if not nodehasattribute(n,iwspaceoffattributeid) then
559         spaceattr = nodegetattribute(n,iwspaceattributeid) or -1
560       end
561       if n.id == HLIST then -- enter the hlist
562         __tag_space_chars_shipout (n)
563       elseif n.id == VLIST then -- enter the vlist

```

```

564     __tag_space_chars_shipout (n)
565 elseif n.id == GLUE then
566     if ltx.__tag.trace.showspace and spaceattr==1 then
567         __tag_show_spacemark (head,n,"0 1 0")
568     end
569     if spaceattr==1 then
570         local space
571         local space_char = node.copy(default_space_char)
572         local curfont = node.getattribute(n,iwfontattributeid)
573         ltx.__tag.trace.log ("INFO SPACE-FUNCTION-FONT: ".. tostring(curfont),3)
574         if curfont and
575             -- luaotfload.aux.slot_of_name(curfont,"space")
576             __tag_font_has_space (curfont)
577         then
578             space_char.font=curfont
579         end
580         head, space = node.insert_before(head, n, space_char) --
581         n.width = n.width - space.width
582         space.attr = n.attr
583     end
584 end
585 end
586 box.head = head
587 end
588 end
589
590 function ltx.__tag.func.space_chars_shipout (box)
591     __tag_space_chars_shipout (box)
592 end

```

(End of definition for `__tag_space_chars_shipout` and `ltx.__tag.func.space_chars_shipout`.)

## 5 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.

```

593 function ltx.__tag.func.mc_insert_kids (mcnum,single)
594     if ltx.__tag.mc[mcnum] then
595         ltx.__tag.trace.log("INFO TEX-MC-INSERT-KID-TEST: " .. mcnum,4)
596         if ltx.__tag.mc[mcnum]["kids"] then
597             if #ltx.__tag.mc[mcnum]["kids"] > 1 and single==1 then
598                 tex.sprint("[")
599             end
600             for i,kidstable in ipairs( ltx.__tag.mc[mcnum]["kids"] ) do
601                 local kidnum = kidstable["kid"]
602                 local kidpage = kidstable["page"]
603                 local kidpageobjnum = pdfpageref(kidpage)
604                 ltx.__tag.trace.log("INFO TEX-MC-INSERT-KID: " .. mcnum ..
605                     " insert KID " .. i ..
606                     " with num " .. kidnum ..
607                     " on page " .. kidpage.."/"..kidpageobjnum,3)

```

```

608     tex.sprint(catlatex,"<</Type /MCR /Pg "..kidpageobjnum .. " 0 R /MCID "..kidnum.. ">> "
609     end
610     if #ltx.__tag.mc[mcnum]["kids"] > 1 and single==1 then
611         tex.sprint("")
612     end
613 else
614     -- this is typically not a problem, e.g. empty hbox in footer/header can
615     -- trigger this warning.
616     ltx.__tag.trace.log("WARN TEX-MC-INSERT-NO-KIDS: "..mcnum.." has no kids",2)
617     if single==1 then
618         tex.sprint("null")
619     end
620 end
621 else
622     ltx.__tag.trace.log("WARN TEX-MC-INSERT-MISSING: "..mcnum.." doesn't exist",0)
623 end
624 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.

```

625 function ltx.__tag.func.store_struct_mcabs (structnum,mcnum)
626     ltx.__tag.struct[structnum]=ltx.__tag.struct[structnum] or { }
627     ltx.__tag.struct[structnum]["mc"]=ltx.__tag.struct[structnum]["mc"] or { }
628     -- a structure can contain more than on mc chunk, the content should be ordered
629     tableinsert(ltx.__tag.struct[structnum]["mc"],mcnum)
630     ltx.__tag.trace.log("INFO TEX-MC-INTO-STRUCT: "..
631         mcnum.." inserted in struct "..structnum,3)
632     -- but every mc can only be in one structure
633     ltx.__tag.mc[mcnum]= ltx.__tag.mc[mcnum] or { }
634     ltx.__tag.mc[mcnum]["parent"] = structnum
635 end
636

```

(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.

```

637 -- pay attention: lua counts arrays from 1, tex pages from one
638 -- mcid and arrays in pdf count from 0.
639 function ltx.__tag.func.store_mc_in_page (mcnum,mcpagecnt,page)
640     ltx.__tag.page[page] = ltx.__tag.page[page] or {}
641     ltx.__tag.page[page][mcpagecnt] = mcnum
642     ltx.__tag.trace.log("INFO TAG-MC-INTO-PAGE: page " .. page ..
643         ": inserting MCID " .. mcpagecnt .. " => " .. mcnum,3)
644 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)

```

645 local function __tag_update_mc_attributes (head,mcnum,type)

```



```

646 for n in node.traverse(head) do
647     node.set_attribute(n,mcntattributeid,mcnum)
648     node.set_attribute(n,mctypeattributeid,type)
649     if n.id == HLIST or n.id == VLIST then
650         __tag_update_mc_attributes (n.list,mcnum,type)
651     end
652 end
653 return head
654 end
655 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.

```

656 --[[
657     Now follows the core function
658     It wades through the shipout box and checks the attributes
659     ARGUMENTS
660     box: is a box,
661     mcpagecnt: num, the current page cnt of mc (should start at -1 in shipout box), needed for
662     mcntprev: num, the attribute cnt of the previous node/whatever - if different we have a
663     mcopen: num, records if some bdc/emc is open
664     These arguments are only needed for log messages, if not present are replaced by fix strings
665     name: string to describe the box
666     mctypeprev: num, the type attribute of the previous node/whatever
667
668     there are lots of logging messages currently. Should be cleaned up in due course.
669     One should also find ways to make the function shorter.
670 --]]
671
672 function ltx.__tag.func.mark_page_elements (box,mcpagecnt,mcntprev,mcopen,name,mctypeprev)
673     local name = name or ("SOMEBOX")
674     local mctypeprev = mctypeprev or -1
675     local abspage = status.total_pages + 1 -- the real counter is increased
676                                           -- inside the box so one off
677                                           -- if the callback is not used. (???)
678     ltx.__tag.trace.log ("INFO TAG-ABSPAGE: " .. abspage,3)
679     ltx.__tag.trace.log ("INFO TAG-ARGS: pagecnt".. mcpagecnt..
680                         " prev "..mcntprev ..
681                         " type prev "..mctypeprev,4)
682     ltx.__tag.trace.log ("INFO TAG-TRAVERSING-BOX: ".. tostring(name)..
683                         " TYPE ".. node.type(node.getid(box)),3)
684     local head = box.head -- ShipoutBox is a vlist?
685     if head then
686         mcntthead, mctypehead,taghead = __tag_get_mc_cnt_type_tag (head)
687         ltx.__tag.trace.log ("INFO TAG-HEAD: " ..
688                             node.type(node.getid(head))..
689                             " MC"..tostring(mcntthead)..
690                             " => TAG " .. tostring(mctypehead)..
691                             " => ".. tostring(taghead),3)
692     else
693         ltx.__tag.trace.log ("INFO TAG-NO-HEAD: head is "..
694                             tostring(head),3)
695     end

```

```

696 for n in node.traverse(head) do
697     local mcnt, mctype, tag = __tag_get_mc_cnt_type_tag (n)
698     local spaceattr = nodegetattribute(n,iwspaceattributeid) or -1
699     ltx.__tag.trace.log ("INFO TAG-NODE: "..
700         node.type(node.getid(n))..
701         " MC".. tostring(mcnt)..
702         " => TAG ".. tostring(mctype)..
703         " => " .. tostring(tag),3)
704     if n.id == HLIST
705     then -- enter the hlist
706         mcopen,mcpagecnt,mcntprev,mctypeprev=
707         ltx.__tag.func.mark_page_elements (n,mcpagecnt,mcntprev,mcopen,"INTERNAL HLIST",mctypeprev)
708     elseif n.id == VLIST then -- enter the vlist
709         mcopen,mcpagecnt,mcntprev,mctypeprev=
710         ltx.__tag.func.mark_page_elements (n,mcpagecnt,mcntprev,mcopen,"INTERNAL VLIST",mctypeprev)
711     elseif n.id == GLUE and not n.leader then -- at glue real space chars are inserted, but t
712         -- been done if the previous shipout wandering, so here it
713     elseif n.id == LOCAL_PAR then -- local_par is ignored
714     elseif n.id == PENALTY then -- penalty is ignored
715     elseif n.id == KERN then -- kern is ignored
716         ltx.__tag.trace.log ("INFO TAG-KERN-SUBTYPE: "..
717             node.type(node.getid(n)).." ".n.subtype,4)
718     else
719         -- math is currently only logged.
720         -- we could mark the whole as math
721         -- for inner processing the mlist_to_hlist callback is probably needed.
722         if n.id == MATH then
723             ltx.__tag.trace.log("INFO TAG-MATH-SUBTYPE: "..
724                 node.type(node.getid(n)).." ".__tag_get_mathsubtype(n),4)
725         end
726         -- endmath
727         ltx.__tag.trace.log("INFO TAG-MC-COMPARE: current "..
728             mcnt.." prev "..mcntprev,4)
729         if mcnt~=mcntprev then -- a new mc chunk
730             ltx.__tag.trace.log ("INFO TAG-NEW-MC-NODE: "..
731                 node.type(node.getid(n))..
732                 " MC"..tostring(mcnt)..
733                 " <=> PREVIOUS "..tostring(mcntprev),4)
734         if mcopen~=0 then -- there is a chunk open, close it (hope there is only one ...
735             box.list=__tag_insert_emc_node (box.list,n)
736             mcopen = mcopen - 1
737             ltx.__tag.trace.log ("INFO TAG-INSERT-EMC: " ..
738                 mcpagecnt .. " MCOPEN = " .. mcopen,3)
739             if mcopen ~=0 then
740                 ltx.__tag.trace.log ("WARN TAG-OPEN-MC: " .. mcopen,1)
741             end
742         end
743         if ltx.__tag.mc[mcnt] then
744             if ltx.__tag.mc[mcnt]["artifact"] then
745                 ltx.__tag.trace.log("INFO TAG-INSERT-ARTIFACT: "..
746                     tostring(ltx.__tag.mc[mcnt]["artifact"]),3)
747             if ltx.__tag.mc[mcnt]["artifact"] == "" then
748                 box.list = __tag_insert_bmc_node (box.list,n,"Artifact")
749             else

```

```

750     box.list = __tag_insert_bdc_node (box.list,n,"Artifact", "/Type /"..ltx.__tag.mc[mc
751     end
752 else
753     ltx.__tag.trace.log("INFO TAG-INSERT-TAG: "..
754         tostring(tag),3)
755     mcpagecnt = mcpagecnt +1
756     ltx.__tag.trace.log ("INFO TAG-INSERT-BDC: "..mcpagecnt,3)
757     local dict= "/MCID "..mcpagecnt
758     if ltx.__tag.mc[mccnt]["raw"] then
759         ltx.__tag.trace.log("INFO TAG-USE-RAW: "..
760             tostring(ltx.__tag.mc[mccnt]["raw"]),3)
761         dict= dict .. " " .. ltx.__tag.mc[mccnt]["raw"]
762     end
763     if ltx.__tag.mc[mccnt]["alt"] then
764         ltx.__tag.trace.log("INFO TAG-USE-ALT: "..
765             tostring(ltx.__tag.mc[mccnt]["alt"]),3)
766         dict= dict .. " " .. ltx.__tag.mc[mccnt]["alt"]
767     end
768     if ltx.__tag.mc[mccnt]["actualtext"] then
769         ltx.__tag.trace.log("INFO TAG-USE-ACTUALTEXT: "..
770             tostring(ltx.__tag.mc[mccnt]["actualtext"]),3)
771         dict= dict .. " " .. ltx.__tag.mc[mccnt]["actualtext"]
772     end
773     box.list = __tag_insert_bdc_node (box.list,n,tag, dict)
774     ltx.__tag.func.store_mc_kid (mccnt,mcpagecnt,abspage)
775     ltx.__tag.func.store_mc_in_page(mccnt,mcpagecnt,abspage)
776     ltx.__tag.trace.show_mc_data (mccnt,3)
777 end
778 mcopen = mcopen + 1
779 else
780     if tagunmarkedbool.mode == truebool.mode then
781         ltx.__tag.trace.log("INFO TAG-NOT-TAGGED: this has not been tagged, using artifact",2)
782         box.list = __tag_insert_bmc_node (box.list,n,"Artifact")
783         mcopen = mcopen + 1
784     else
785         ltx.__tag.trace.log("WARN TAG-NOT-TAGGED: this has not been tagged",1)
786     end
787 end
788 mccntprev = mccnt
789 end
790 end -- end if
791 end -- end for
792 if head then
793     mccnthead, mctypehead,taghead = __tag_get_mc_cnt_type_tag (head)
794     ltx.__tag.trace.log ("INFO TAG-ENDHEAD: " ..
795         node.type(node.getid(head))..
796         " MC"..tostring(mccnthead)..
797         " => TAG "..tostring(mctypehead)..
798         " => "..tostring(taghead),4)
799 else
800     ltx.__tag.trace.log ("INFO TAG-ENDHEAD: ".. tostring(head),4)
801 end
802 ltx.__tag.trace.log ("INFO TAG-QUITTING-BOX "..
803     tostring(name)..

```

```

804             " TYPE ".. node.type(node.getid(box)),4)
805     return mcopen,mcpagecnt,mcntprev,mctypeprev
806 end
807

```

(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.

```

808 function ltx.__tag.func.mark_shipout (box)
809     mcopen = ltx.__tag.func.mark_page_elements (box,-1,-100,0,"Shipout",-1)
810     if mcopen~=0 then -- there is a chunk open, close it (hope there is only one ...
811         local emcnode = __tag_backend_create_emc_node ()
812         local list = box.list
813         if list then
814             list = node.insert_after (list,node.tail(list),emcnode)
815             mcopen = mcopen - 1
816             ltx.__tag.trace.log ("INFO SHIPOUT-INSERT-LAST-EMC: MCOOPEN " .. mcopen,3)
817         else
818             ltx.__tag.trace.log ("WARN SHIPOUT-UPS: this shouldn't happen",0)
819         end
820         if mcopen ~=0 then
821             ltx.__tag.trace.log ("WARN SHIPOUT-MC-OPEN: " .. mcopen,1)
822         end
823     end
824 end

```

(End of definition for ltx.\_\_tag.func.mark\_shipout.)

## 6 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.

```

825 function ltx.__tag.func.fill_parent_tree_line (page)
826     -- we need to get page-> i=kid -> mcnum -> structnum
827     -- pay attention: the kid numbers and the page number in the parent tree start with 0!
828     local numscopy = ""
829     local pdfpage = page-1
830     if ltx.__tag.page[page] and ltx.__tag.page[page][0] then
831         mcchunks=#ltx.__tag.page[page]
832         ltx.__tag.trace.log("INFO PARENTTREE-NUM: page "..
833             page.." has "..mcchunks.." +1 Elements ",4)
834         for i=0,mcchunks do
835             -- what does this log??
836             ltx.__tag.trace.log("INFO PARENTTREE-CHUNKS: "..
837                 ltx.__tag.page[page][i],4)
838         end
839         if mcchunks == 0 then
840             -- only one chunk so no need for an array
841             local mcnum = ltx.__tag.page[page][0]
842             local structnum = ltx.__tag.mc[mcnum]["parent"]
843             local propname = "g__tag_struct_"..structnum.."_prop"

```

```

844     --local objref = ltx.__tag.tables[proprname]["objref"] or "XXXX"
845     local objref = __tag_pdf_object_ref('__tag/struct',structnum)
846     ltx.__tag.trace.log("INFO PARENTTREE-STRUCT-OBJREF: =====>"..
847         tostring(objref),5)
848     numsentry = pdfpage .. " [".. objref .. "]"
849     ltx.__tag.trace.log("INFO PARENTTREE-NUMENTRY: page " ..
850         page.. " num entry = ".. numsentry,3)
851     else
852         numsentry = pdfpage .. " ["
853         for i=0,mcchunks do
854             local mcnum = ltx.__tag.page[page][i]
855             local structnum = ltx.__tag.mc[mcnum]["parent"] or 0
856             local proprname = "g__tag_struct_"..structnum.."__prop"
857             --local objref = ltx.__tag.tables[proprname]["objref"] or "XXXX"
858             local objref = __tag_pdf_object_ref('__tag/struct',structnum)
859             numsentry = numsentry .. " " .. objref
860         end
861         numsentry = numsentry .. "]" "
862         ltx.__tag.trace.log("INFO PARENTTREE-NUMENTRY: page " ..
863             page.. " num entry = ".. numsentry,3)
864     end
865     else
866         ltx.__tag.trace.log ("INFO PARENTTREE-NO-DATA: page "..page,3)
867         numsentry = pdfpage.." []"
868     end
869     return numsentry
870 end
871
872 function ltx.__tag.func.output_parenttree (abspage)
873     for i=1,abspage do
874         line = ltx.__tag.func.fill_parent_tree_line (i) .. "^^J"
875         tex.sprint(catlatex,line)
876     end
877 end

```

(End of definition for ltx.\_\_tag.func.fill\_parent\_tree\_line and ltx.\_\_tag.func.output\_parenttree.)

First some local definitions. Since these are only needed locally everything gets wrapped into a block.

```

878 do
879     local properties = node.get_properties_table()
880     local is_soft_hyphen_prop = 'tagpdf.rewrite-softthyphen.is_soft_hyphen'
881     local hyphen_char = 0x2D
882     local soft_hyphen_char = 0xAD
883     A lookup table to test if the font supports the soft hyphen glyph.
884     local softthyphen_fonts = setmetatable({}, {__index = function(t, fid)
885         local fdir = identifiers[fid]
886         local format = fdir and fdir.format
887         local result = (format == 'opentype' or format == 'truetype')
888         local characters = fdir and fdir.characters
889         result = result and (characters and characters[soft_hyphen_char]) ~= nil
890         t[fid] = result
891         return result
892     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.

```

892 local function process_softthyphen_pre(head, _context, _dir)
893   if softthyphenbool.mode ~= truebool.mode then return true end
894   for disc, sub in node.traverse_id(DISC, head) do
895     if sub == explicit_disc or sub == regular_disc then
896       for n, _ch, _f in node.traverse_char(disc.pre) do
897         local props = properties[n]
898         if not props then
899           props = {}
900           properties[n] = props
901         end
902         props[is_soft_hyphen_prop] = true
903       end
904     end
905   end
906   return true
907 end
908

```

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

```

909 local function process_softthyphen_post(head, _context, _dir)
910   if softthyphenbool.mode ~= truebool.mode then return true end
911   for disc, sub in node.traverse_id(DISC, head) do
912     for n, ch, fid in node.traverse_glyph(disc.pre) do
913       local props = properties[n]
914       if softthyphen_fonts[fid] and ch == hyphen_char and props and props[is_soft_hyphen_prop] then
915         n.char = soft_hyphen_char
916         props.glyph_info = nil
917       end
918     end
919   end
920   return true
921 end
922
923 luatexbase.add_to_callback('pre_shaping_filter', process_softthyphen_pre, 'tagpdf.rewrite-
softthyphen')
924 luatexbase.add_to_callback('post_shaping_filter', process_softthyphen_post, 'tagpdf.rewrite-
softthyphen')
925 end

```

(End of definition for process\_softthyphen\_pre process\_softthyphen\_post. This function is documented on page ??.)

```

926 </lua>

```

## Part IX

# The `tagpdf-roles` module

## Tags, roles and namespace code

### Part of the `tagpdf` package

---

```
add-new-tag_(setup-key)
tag_(rolemap-key)
namespace_(rolemap-key)
role_(rolemap-key)
role-namespace_(rolemap-key)
```

---

The `add-new-tag` key can be used in `\tagpdfsetup` to declare and rolemap new tags. It takes as value a key-value list or a simple `new-tag/old-tag`.

The key-value list knows the following keys:

**tag** This is the name of the new tag as it should then be used in `\tagstructbegin`.

**namespace** This is the namespace of the new tag. The value should be a shorthand of a namespace. The allowed values are currently `pdf`, `pdf2`, `mathml`, `latex`, `latex-book` and `user`. The default value (and recommended value for a new tag) is `user`. The public name of the user namespace is `tag/NS/user`. This can be used to reference the namespace e.g. in attributes.

**role** This is the tag the tag should be mapped too. In a PDF 1.7 or earlier this is normally a tag from the `pdf` set, in PDF 2.0 from the `pdf`, `pdf2` and `mathml` set. It can also be a user tag. The tag must be declared before, as the code retrieves the class of the new tag from it. The PDF format allows mapping to be done transitively. But `tagpdf` can't/won't check such unusual role mapping.

**role-namespace** If the role is a known tag the default value is the default namespace of this tag. With this key a specific namespace can be forced.

Namespaces are mostly a PDF 2.0 property, but it doesn't harm to set them also in a PDF 1.7 or earlier.

---

```
\tag_check_child:nnTF \tag_check_child:nn{<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} {2024-09-11} {0.99e}
4 {part of tagpdf - code related to roles and structure names}
5 </header>
```

## 1 Code related to roles and structure names

<sup>6</sup> `\*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.

`\l__tag_role_debug_prop` This property is used to pass some info around for info messages or debugging.



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

```
7 \prop_new:N \g__tag_role_tags_NS_prop
```

(End of definition for `\g__tag_role_tags_NS_prop`.)

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

```
8 \prop_new:N \g__tag_role_tags_class_prop
```

(End of definition for `\g__tag_role_tags_class_prop`.)

`\g__tag_role_NS_prop` This holds the list of supported name spaces. The keys are the name tagpdf will use, the values the object reference. The urls identifier are stored in related dict object.

**mathml** <http://www.w3.org/1998/Math/MathML>

**pdf2** <http://iso.org/pdf/ssn>

**pdf** <http://iso.org/pdf/ssn> (default)

**user** `\c__tag_role_userNS_id_str` (random id, for user tags)

**latex** <https://www.latex-project.org/ns/dft/2022>

**latex-book** <https://www.latex-project.org/ns/book/2022>

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

```
9 \prop_new:N \g__tag_role_NS_prop
```

(End of definition for `\g__tag_role_NS_prop`.)

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

```
10 \prop_new:N \g__tag_role_index_prop
```

(End of definition for `\g__tag_role_index_prop`.)

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

```
11 \prop_new:N \l__tag_role_debug_prop
```

(End of definition for `\l__tag_role_debug_prop`.)

We need also a bunch of temporary variables.

```
\l__tag_role_tag_tmpa_tl
```

```
\l__tag_role_tag_namespace_tmpa_tl
```

```
12 \tl_new:N \l__tag_role_tag_tmpa_tl
```

```
\l__tag_role_tag_namespace_tmpb_tl
```

```
13 \tl_new:N \l__tag_role_tag_namespace_tmpa_tl
```

```
\l__tag_role_role_tmpa_tl
```

```
14 \tl_new:N \l__tag_role_tag_namespace_tmpb_tl
```

```
\l__tag_role_role_namespace_tmpa_tl
```

```
15 \tl_new:N \l__tag_role_role_tmpa_tl
```

```
\l__tag_role_role_namespace_tmpa_tl
```

```
16 \tl_new:N \l__tag_role_role_namespace_tmpa_tl
```

```
\l__tag_role_tmpa_seq
```

```
17 \seq_new:N \l__tag_role_tmpa_seq
```

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

## 1.2 Namespaces

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

`g__tag_role/RoleMap_dict` This is the object which contains the normal RoleMap. It is probably not needed in pdf  
`\g__tag_role_rolemap_prop` 2.0 but currently kept.

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

```

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

```

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

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

`\c__tag_role_userNS_id_str`

```

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

```

(End of definition for `\c__tag_role_userNS_id_str`.)

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

```

73 \bool_new:N \g__tag_role_add_mathml_bool
74 \__tag_role_NS_new:nnn {pdf} {http://iso.org/pdf/ssn}{}
75 \__tag_role_NS_new:nnn {pdf2} {http://iso.org/pdf2/ssn}{}
76 \__tag_role_NS_new:nnn {mathml} {http://www.w3.org/1998/Math/MathML}{}
77 \__tag_role_NS_new:nnn {latex} {https://www.latex-project.org/ns/dfl/2022}{}
78 \__tag_role_NS_new:nnn {latex-book} {https://www.latex-project.org/ns/book/2022}{}
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   \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   \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   \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   \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 {nnV}

```

(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_if_in:NnF \g__tag_role_tags_NS_prop {#1}
131 {
132   \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
133   {
134     \msg_info:nnn { tag }{new-tag}{#1}
135   }
136 }

```

now the addition

```

137 \prop_get:NnN \g__tag_role_tags_class_prop {#2}\l__tag_tmpa_tl
138 \quark_if_no_value:NT \l__tag_tmpa_tl
139 {
140   \tl_set:Nn\l__tag_tmpa_tl{--UNKNOWN--}
141 }
142 \__tag_role_alloctag:nnV {#1}{user}\l__tag_tmpa_tl

```

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

```

143 \tl_if_empty:nF { #2 }
144 {
145   \prop_get:NnN \g__tag_role_rolemap_prop {#2}\l__tag_tmpa_tl
146   \quark_if_no_value:NTF \l__tag_tmpa_tl
147   {
148     \prop_gput:Nne \g__tag_role_rolemap_prop {#1}{\tl_to_str:n{#2}}
149   }
150   {
151     \prop_gput:NnV \g__tag_role_rolemap_prop {#1}\l__tag_tmpa_tl
152   }
153 }
154 }
155 \cs_generate_variant:Nn \__tag_role_add_tag:nn {VV,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.

\\_\_tag\_role\_get:nnNN

```

156 \pdf_version_compare:NnT < {2.0}
157 {
158   \cs_new:Npn \__tag_role_get:nnNN #1#2#3#4 %#1 tag, #2 NS, #3 tlvar which hold the role tag
159   {
160     \prop_get:NnNF \g__tag_role_rolemap_prop {#1}#3
161     {
162       \tl_set:Nn #3 {#1}
163     }
164     \tl_set:Nn #4 {}
165   }
166   \cs_generate_variant:Nn \__tag_role_get:nnNN {VVNN}
167 }
168

```

(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`

```

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

Do not remap standard tags. TODO add warning?

```

187 \tl_if_in:nnF {-pdf-pdf2-mathml-}{-#2-}
188 {
189   \pdfdict_gput:nne {g__tag_role/RoleMapNS_#2_dict}{#1}
190   {
191     [
192       \pdf_name_from_unicode_e:n{#3}
193       \c_space_tl
194       \pdf_object_ref:n {tag/NS/#4}
195     ]
196   }
197 }
  
```

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

```

198 \tl_if_empty:nF { #2 }
199 {
200   \prop_get:cnN { g__tag_role_NS_#4_prop } {#3}\l__tag_tmpa_tl
201   \quark_if_no_value:NTF \l__tag_tmpa_tl
202   {
203     \prop_gput:cne { g__tag_role_NS_#2_prop } {#1}
204     { {\tl_to_str:n{#3}}{\tl_to_str:n{#4}} }
205   }
206   {
207     \prop_gput:cno { g__tag_role_NS_#2_prop } {#1}{\l__tag_tmpa_tl}
208   }
209 }
  
```

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

```

210 \bool_if:NT \l__tag_role_update_bool
211 {
212   \tl_if_empty:nF { #3 }
  
```

```

213     {
214         \tl_if_eq:nnF{#1}{#3}
215         {
216             \prop_get:NnN \g__tag_role_rolemap_prop {#3}\l__tag_tmpa_tl
217             \quark_if_no_value:NTF \l__tag_tmpa_tl
218             {
219                 \prop_gput:Nne \g__tag_role_rolemap_prop {#1}{\tl_to_str:n{#3}}
220             }
221             {
222                 \prop_gput:NnV \g__tag_role_rolemap_prop {#1}\l__tag_tmpa_tl
223             }
224         }
225     }
226 }
227 }
228 \cs_generate_variant:Nn \__tag_role_add_tag:nnnn {VVVV}

```

(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 (and assume that we don't need a name space)

\\_\_tag\_role\_get:nnNN

```

229 \pdf_version_compare:NnF < {2.0}
230 {
231     \cs_new:Npn \__tag_role_get:nnNN #1#2#3#4
232     % #1 tag, #2 NS,
233     % #3 tlvar which hold the role tag
234     % #4 tlvar which hold the name of the target NS
235     {
236         \prop_if_exist:cTF {g__tag_role_NS_#2_prop}
237         {
238             \prop_get:cnNTF {g__tag_role_NS_#2_prop} {#1}\l__tag_get_tmpc_tl
239             {
240                 \tl_set:Ne #3 {\exp_last_unbraced:NV\use_i:nn \l__tag_get_tmpc_tl}
241                 \tl_set:Ne #4 {\exp_last_unbraced:NV\use_ii:nn \l__tag_get_tmpc_tl}
242             }
243             {
244                 \msg_warning:nnn { tag } {role-unknown-tag} { #1 }
245                 \tl_set:Nn #3 {#1}
246                 \tl_set:Nn #4 {#2}
247             }
248         }
249         {
250             \msg_warning:nnn { tag } {role-unknown-NS} { #2 }
251             \tl_set:Nn #3 {#1}
252             \tl_set:Nn #4 {#2}
253         }
254     }
255     \cs_generate_variant:Nn \__tag_role_get:nnNN {VVNN}
256 }

```

(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.

```

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

```



```

303         {
304         \tl_set:Nn\l__tag_tmpa_tl{--UNKNOWN--}
305         }
306     }
307     {
308     \tl_set:Nn \l__tag_tmpa_tl {#5}
309     }
310     \__tag_role_alloctag:nnV {#2}{#1}\l__tag_tmpa_tl
311     \bool_lazy_and:nnT
312     { ! \tl_if_empty_p:n {#3} }{! \str_if_eq_p:nn {#1}{pdf2}}
313     {
314     \__tag_role_add_tag:nxxx {#2}{#1}{#3}{#4}
315     }
316     \prop_gput:cnn {g__tag_role_NS_#1_prop} {#2}{#3}{#4}
317 }
318 }
319 }

```

(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

```

320 \cs_new_protected:Npn \__tag_role_read_namespace:nn #1 #2 %name of namespace #2 name of file
321 {
322     \prop_if_exist:cF {g__tag_role_NS_#1_prop}
323     { \msg_warning:nnn {tag}{namespace-unknown}{#1} }
324     \file_if_exist:nTF { tagpdf-ns-#2.def }
325     {
326     \ior_open:Nn \g_tmpa_ior {tagpdf-ns-#2.def}
327     \msg_info:nnn {tag}{read-namespace}{#2}
328     \ior_map_inline:Nn \g_tmpa_ior
329     {
330     \__tag_role_read_namespace_line:nw {#1} ##1,,, \q_stop
331     }
332     \ior_close:N\g_tmpa_ior
333     }
334     {
335     \msg_info:nnn{tag}{namespace-missing}{#2}
336     }
337 }
338

```

(End of definition for \\_\_tag\_role\_read\_namespace:nn.)

\\_\_tag\_role\_read\_namespace:n

This command reads the default namespace file.

```

339 \cs_new_protected:Npn \__tag_role_read_namespace:n #1 %name of namespace
340 {
341     \__tag_role_read_namespace:nn {#1}{#1}
342 }

```

(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.

```

343 \__tag_role_read_namespace:n {pdf}
344 \__tag_role_read_namespace:n {pdf2}
345 \__tag_role_read_namespace:n {mathml}

```

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

```

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

```

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

```

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

```

```

390     }
391 }

```

## 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.

```

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

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

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

```

393 \cs_new_protected:Npn \_tag_store_parent_child_rule:nnn #1 #2 #3 % num parent, num child, #3
394 {
395   \intarray_gset:Nnn \g_tag_role_parent_child_intarray
396     { #1#2 }{0\prop_item:Nn\c__tag_role_rules_prop{#3}}
397 }

```

(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

```

398 \int_zero:N \l__tag_tmpa_int

```

Open the file depending on the PDF version

```

399 \pdf_version_compare:NnTF < {2.0}
400 {
401   \ior_open:Nn \g_tmpa_ior {tagpdf-parent-child.csv}
402 }
403 {
404   \ior_open:Nn \g_tmpa_ior {tagpdf-parent-child-2.csv}
405 }

```

Now the main loop over the file

```

406 \ior_map_inline:Nn \g_tmpa_ior
407 {

```

ignore lines containing only comments

```

408   \tl_if_empty:nF{#1}
409   {

```

count the lines ...

```

410     \int_incr:N\l__tag_tmpa_int

```

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

```

411     \seq_set_from_clist:Nn\l__tag_tmpa_seq { #1 }
412     \int_compare:nNnTF {\l__tag_tmpa_int}=1

```

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

```

413     {
414         \seq_map_indexed_inline:Nn \l__tag_tmpa_seq
415         {
416             \prop_gput:Nne\g__tag_role_index_prop
417             {##2}
418             {\int_compare:nNnT{##1}<{10}{0}##1}
419         }
420     }

```

now the data lines.

```

421     {
422         \seq_set_from_clist:Nn\l__tag_tmpa_seq { #1 }

```

get the name of the child tag from the first column

```

423         \seq_pop_left:NN\l__tag_tmpa_seq\l__tag_tmpa_tl

```

get the number of the child, and store it in \l\_\_tag\_tmpb\_tl

```

424         \prop_get:NVN \g__tag_role_index_prop \l__tag_tmpa_tl \l__tag_tmpb_tl

```

remove column 2+3

```

425         \seq_pop_left:NN\l__tag_tmpa_seq\l__tag_tmpa_tl
426         \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.

```

427         \seq_map_indexed_inline:Nn \l__tag_tmpa_seq
428         {
429             \exp_args:Nne
430             \__tag_store_parent_child_rule:nnn {##1}{\l__tag_tmpb_tl}{ ##2 }
431         }
432     }
433 }
434 }

```

close the read handle.

```

435 \ior_close:N\g_tmpa_ior

```

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

```

436 \prop_get:NnN\g__tag_role_index_prop{StructTreeRoot}\l__tag_tmpa_tl
437 \prop_gput:Nne\g__tag_role_index_prop{Root}{\l__tag_tmpa_tl}
438 \prop_get:NnN\g__tag_role_index_prop{Hn}\l__tag_tmpa_tl
439 \pdf_version_compare:NnTF < {2.0}
440 {
441     \int_step_inline:nn{6}
442     {
443         \prop_gput:Nne\g__tag_role_index_prop{H#1}{\l__tag_tmpa_tl}
444     }
445 }
446 {
447     \int_step_inline:nn{10}
448     {
449         \prop_gput:Nne\g__tag_role_index_prop{H#1}{\l__tag_tmpa_tl}
450     }

```

all mathml tags are currently handled identically

```

451     \prop_get:NnN\g__tag_role_index_prop {mathml}\l__tag_tmpa_tl
452     \prop_get:NnN\g__tag_role_index_prop {math}\l__tag_tmpb_tl
453     \prop_map_inline:Nn \g__tag_role_NS_mathml_prop
454     {
455         \prop_gput:NnV\g__tag_role_index_prop{#1}\l__tag_tmpa_tl
456     }
457     \prop_gput:NnV\g__tag_role_index_prop{math}\l__tag_tmpb_tl
458 }

```

### 1.6.2 Retrieving the parent-child rule

`\__tag_role_get_parent_child_rule:nnnN`

This command retrieves the rule (as a number) and stores it in the tl-var. It assumes that the tag in #1 is a standard tag after role mapping for which a rule exist and is *not* one of Part, Div, NonStruct as the real parent has already been identified. #3 can be used to pass along data about the original tags and is only used in messages.

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

```

459 \tl_new:N \l__tag_parent_child_check_tl
460 \cs_new_protected:Npn \__tag_role_get_parent_child_rule:nnnN #1 #2 #3 #4
461 % #1 parent (string) #2 child (string) #3 text for messages (eg. about Div or Rolemapping)
462 % #4 tl for state
463 {
464     \prop_get:NnN \g__tag_role_index_prop{#1}\l__tag_tmpa_tl
465     \prop_get:NnN \g__tag_role_index_prop{#2}\l__tag_tmpb_tl
466     \bool_lazy_and:nnTF
467     { ! \quark_if_no_value_p:N \l__tag_tmpa_tl }
468     { ! \quark_if_no_value_p:N \l__tag_tmpb_tl }
469     {

```

Get the rule from the intarray

```

470         \tl_set:Ne#4
471         {
472             \intarray_item:Nn
473             \g__tag_role_parent_child_intarray
474             {\l__tag_tmpa_tl\l__tag_tmpb_tl}
475         }

```

If the state is something is wrong ...

```

476         \int_compare:nNnT
477         {#4} = {\prop_item:Nn\c__tag_role_rules_prop{}}
478         {
479             %warn ?

```

we must take the current child from the stack if is already there, depending on location the check is called, this could also remove the parent, but that is ok too.

```

480     }

```

This is the message, this can perhaps go into debug mode.

```

481     \group_begin:
482     \int_compare:nNnT {\l__tag_tmpa_int*\l__tag_loglevel_int} > { 0 }
483     {
484         \prop_get:NVNF\c__tag_role_rules_num_prop #4 \l__tag_tmpa_tl
485         {
486             \tl_set:Nn \l__tag_tmpa_tl {unknown}

```

```

487     }
488     \tl_set:Nn \l__tag_tmpb_tl {#1}
489     \msg_note:nneee
490     { tag }
491     { role-parent-child }
492     { #1 }
493     { #2 }
494     {
495       #4~(=\l__tag_tmpa_tl')
496       \iow_newline:
497       #3
498     }
499   }
500   \group_end:
501 }
502 {
503   \tl_set:Nn#4 {0}
504   \msg_warning:nneee
505   { tag }
506   {role-parent-child}
507   { #1 }
508   { #2 }
509   { unknown! }
510 }
511 }
512 \cs_generate_variant:Nn\__tag_role_get_parent_child_rule:nnnN {VVVN,VVN}

```

(End of definition for \\_\_tag\_role\_get\_parent\_child\_rule:nnnN.)

\_\_tag\_check\_parent\_child:nnnnN

This commands translates rolemaps its arguments and then calls \\_\_tag\_role\_get\_parent\_child\_rule:nnnN. It does not try to resolve inheritance of Div etc but instead warns that the rule can not be detected in this case. 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.

```

513 \pdf_version_compare:NnTF < {2.0}
514 {
515   \cs_new_protected:Npn \__tag_check_parent_child:nnnnN #1 #2 #3 #4 #5
516   {%#1 parent tag,#2 NS, #3 child tag, #4 NS, #5 tl var
517     {

```

for debugging messages we store the arguments.

```

518     \prop_put:Nnn \l__tag_role_debug_prop {parent} {#1}
519     \prop_put:Nnn \l__tag_role_debug_prop {child} {#3}

```

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

```

520     \prop_get:NnNTF \g__tag_role_index_prop {#1}\l__tag_tmpa_tl
521     {
522       \tl_set:Nn \l__tag_tmpa_tl {#1}
523     }
524     {
525       \prop_get:NnNF \g__tag_role_rolemap_prop {#1}\l__tag_tmpa_tl
526       {
527         \tl_set:Nn \l__tag_tmpa_tl {\q_no_value}
528       }
529     }

```

now the child

```
530     \prop_get:NnNTF \g__tag_role_index_prop {#3}\l__tag_tmpb_tl
531     {
532       \tl_set:Nn \l__tag_tmpb_tl {#3}
533     }
534     {
535       \prop_get:NnNF \g__tag_role_rolemap_prop {#3}\l__tag_tmpb_tl
536       {
537         \tl_set:Nn \l__tag_tmpb_tl {\q_no_value}
538       }
539     }
```

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

```
540     \bool_lazy_and:nnTF
541     { ! \quark_if_no_value_p:N \l__tag_tmpa_tl }
542     { ! \quark_if_no_value_p:N \l__tag_tmpb_tl }
543     {
544       \__tag_role_get_parent_child_rule:VVnN
545       \l__tag_tmpa_tl \l__tag_tmpb_tl
546       {Rolemapped~from:~'~#1'~-->~'~#3'~}
547       #5
548     }
549     {
550       \tl_set:Nn #5 {0}
551       \msg_warning:nneee
552       { tag }
553       {role-parent-child}
554       { #1 }
555       { #3 }
556       { unknown! }
557     }
558   }
559   \cs_new_protected:Npn \__tag_check_parent_child:nnN #1#2#3
560   {
561     \__tag_check_parent_child:nnnnN {#1}{#2}{#3}
562   }
563 }
```

and now the pdf 2.0 version The version with three arguments retrieves the default names space and then calls the full command. Not sure if this will ever be needed but we leave it for now.

```
564 {
565   \cs_new_protected:Npn \__tag_check_parent_child:nnN #1 #2 #3
566   {
567     \prop_get:NnN\g__tag_role_tags_NS_prop {#1}\l__tag_role_tag_namespace_tmpa_tl
568     \prop_get:NnN\g__tag_role_tags_NS_prop {#2}\l__tag_role_tag_namespace_tmpb_tl
569     \str_if_eq:nnT{#2}{MC}{\tl_clear:N \l__tag_role_tag_namespace_tmpb_tl}
570     \bool_lazy_and:nnTF
571     { ! \quark_if_no_value_p:N \l__tag_role_tag_namespace_tmpa_tl }
572     { ! \quark_if_no_value_p:N \l__tag_role_tag_namespace_tmpb_tl }
573     {
574       \__tag_check_parent_child:nVnVN
575       {#1}\l__tag_role_tag_namespace_tmpa_tl
576       {#2}\l__tag_role_tag_namespace_tmpb_tl

```

```

577         #3
578     }
579     {
580         \tl_set:Nn #3 {0}
581         \msg_warning:nnee
582         { tag }
583         {role-parent-child}
584         { #1 }
585         { #2 }
586         { unknown! }
587     }
588 }

```

and now the real command.

```

589 \cs_new_protected:Npn \__tag_check_parent_child:nnnnN #1 #2 #3 #4 #5 %tag,NS,tag,NS, t1 va
590 {
591     \prop_put:Nnn \l__tag_role_debug_prop {parent} {#1/#2}
592     \prop_put:Nnn \l__tag_role_debug_prop {child} {#3/#4}

```

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

```

593     \tl_if_empty:nTF {#2}
594     {
595         \tl_set:Nn \l__tag_tmpa_tl {#1}
596     }
597     {
598         \prop_if_exist:cTF { g__tag_role_NS_#2_prop }
599         {
600             \prop_get:cnNTF
601             { g__tag_role_NS_#2_prop }
602             {#1}
603             \l__tag_tmpa_tl
604             {
605                 \tl_set:Ne \l__tag_tmpa_tl {\tl_head:N\l__tag_tmpa_tl}
606                 \tl_if_empty:NT\l__tag_tmpa_tl
607                 {
608                     \tl_set:Nn \l__tag_tmpa_tl {#1}
609                 }
610             }
611             {
612                 \tl_set:Nn \l__tag_tmpa_tl {\q_no_value}
613             }
614         }
615         {
616             \msg_warning:nnn { tag } {role-unknown-NS} { #2}
617             \tl_set:Nn \l__tag_tmpa_tl {\q_no_value}
618         }
619     }

```

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

```

620     \tl_if_empty:nTF {#4}
621     {
622         \tl_set:Nn \l__tag_tmpb_tl {#3}
623     }

```



```

624     {
625       \prop_if_exist:cTF { g__tag_role_NS_#4_prop }
626       {
627         \prop_get:cnNTF
628         { g__tag_role_NS_#4_prop }
629         {#3}
630         \l__tag_tmpb_tl
631         {
632           \tl_set:Ne \l__tag_tmpb_tl { \tl_head:N\l__tag_tmpb_tl }
633           \tl_if_empty:NT\l__tag_tmpb_tl
634           {
635             \tl_set:Nn \l__tag_tmpb_tl {#3}
636           }
637         }
638         {
639           \tl_set:Nn \l__tag_tmpb_tl {\q_no_value}
640         }
641       }
642       {
643         \msg_warning:nnn { tag } {role-unknown-NS} { #4}
644         \tl_set:Nn \l__tag_tmpb_tl {\q_no_value}
645       }
646     }

```

and now get the relation

```

647     \bool_lazy_and:nnTF
648     { ! \quark_if_no_value_p:N \l__tag_tmpa_tl }
649     { ! \quark_if_no_value_p:N \l__tag_tmpb_tl }
650     {
651       \__tag_role_get_parent_child_rule:VVnN
652       \l__tag_tmpa_tl \l__tag_tmpb_tl
653       {Rolemapped~from~'#1/#2'--->~'#3\str_if_empty:nF{#4}{/#4}' }
654       #5
655     }
656     {
657       \tl_set:Nn #5 {0}
658       \msg_warning:nneee
659       { tag }
660       {role-parent-child}
661       { #1 }
662       { #3 }
663       { unknown! }
664     }
665   }
666 }
667 \cs_generate_variant:Nn\__tag_check_parent_child:nnN {VVN}
668 \cs_generate_variant:Nn\__tag_check_parent_child:nnnnN {VVVVN,nVnVN,VVnnN}
669 \endpackage

```

(End of definition for \_\_tag\_check\_parent\_child:nnnnN.)

**\tag\_check\_child:nnTF**

```

670 <base>\prg_new_protected_conditional:Npnn \tag_check_child:nn #1 #2 {T,F,TF}{\prg_return_true}
671 *package)
672 \prg_set_protected_conditional:Npnn \tag_check_child:nn #1 #2 {T,F,TF}

```

```

673 {
674   \seq_get:NN\g__tag_struct_stack_seq\l__tag_tpa_tl
675   \__tag_struct_get_parentrole:eNN
676     {\l__tag_tpa_tl}
677     \l__tag_get_parent_tpa_tl
678     \l__tag_get_parent_tpb_tl
679   \__tag_check_parent_child:VVnnN
680     \l__tag_get_parent_tpa_tl
681     \l__tag_get_parent_tpb_tl
682     {#1}{#2}
683     \l__tag_parent_child_check_tl
684   \int_compare:nNnTF { \l__tag_parent_child_check_tl } < {0}
685     {\prg_return_false:}
686     {\prg_return_true:}
687 }

```

(End of definition for `\tag_check_child:nTF`. This function is documented on page 159.)

## 1.7 Remapping of tags

In some context it can be necessary to remap or replace the tags. That means instead of `tag=H1` or `tag=section` one wants the effect of `tag=Span`. Or instead of `tag=P` one wants `tag=Code`.

The following command provide some general interface for this. The core idea is that before a tag is set it is fed through a function that can change it. We want to be able to chain such functions, so all of them manipulate the same variables.

```

\l__tag_role_remap_tag_tl
\l__tag_role_remap_NS_tl
688 \tl_new:N \l__tag_role_remap_tag_tl
689 \tl_new:N \l__tag_role_remap_NS_tl

```

(End of definition for `\l__tag_role_remap_tag_tl` and `\l__tag_role_remap_NS_tl`.)

`\__tag_role_remap:` This function is used in the structure and the mc code before using a tag. By default it does nothing with the `tl` vars. Perhaps this should be a hook?

```

690 \cs_new_protected:Npn \__tag_role_remap: { }

```

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

`\__tag_role_remap_id:` This is copy in case we have to restore the main command.

```

691 \cs_set_eq:NN \__tag_role_remap_id: \__tag_role_remap:

```

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

## 1.8 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) 692 \keys_define:nn { __tag / tag-role }
role_(rolemap-key)          693 {
role-namespace_(rolemap-key) 694   ,tag .tl_set:N = \l__tag_role_tag_tmpa_tl
role/new-tag_(setup-key)     695   ,tag-namespace .tl_set:N = \l__tag_role_tag_namespace_tmpa_tl
add-new-tag_(deprecated)    696   ,role .tl_set:N = \l__tag_role_role_tmpa_tl
                             697   ,role-namespace .tl_set:N = \l__tag_role_role_namespace_tmpa_tl
                             698 }
                             699
700 \keys_define:nn { __tag / setup }
701 {
702   role/mathml-tags .bool_gset:N = \g__tag_role_add_mathml_bool
703   ,role/new-tag .code:n =
704   {
705     \keys_set_known:nnnN
706     {__tag/tag-role}
707     {
708       tag-namespace=user,
709       role-namespace=, %so that we can test for it.
710       #1
711       }{__tag/tag-role}\l_tmpa_tl
712     \tl_if_empty:NF \l_tmpa_tl
713     {
714       \exp_args:NNno \seq_set_split:Nnn \l_tmpa_seq { / } {\l_tmpa_tl/}
715       \tl_set:Ne \l__tag_role_tag_tmpa_tl { \seq_item:Nn \l_tmpa_seq {1} }
716       \tl_set:Ne \l__tag_role_role_tmpa_tl { \seq_item:Nn \l_tmpa_seq {2} }
717     }
718     \tl_if_empty:NT \l__tag_role_role_namespace_tmpa_tl
719     {
720       \prop_get:NVNTF
721       \g__tag_role_tags_NS_prop
722       \l__tag_role_role_tmpa_tl
723       \l__tag_role_role_namespace_tmpa_tl
724       {
725         \prop_if_in:NVF\g__tag_role_NS_prop \l__tag_role_role_namespace_tmpa_tl
726         {
727           \tl_set:Nn \l__tag_role_role_namespace_tmpa_tl {user}
728         }
729       }
730       {
731         \tl_set:Nn \l__tag_role_role_namespace_tmpa_tl {user}
732       }
733     }
734     \pdf_version_compare:NnTF < {2.0}
735     {
736       %TODO add check for emptyness?
737       \__tag_role_add_tag:VV
738       \l__tag_role_tag_tmpa_tl
739       \l__tag_role_role_tmpa_tl
740     }
741     {
742       \__tag_role_add_tag:VVVV
743       \l__tag_role_tag_tmpa_tl
744       \l__tag_role_tag_namespace_tmpa_tl

```

```

745         \l__tag_role_role_tmpa_tl
746         \l__tag_role_role_namespace_tmpa_tl
747     }
748 }
749 ,role/map-tags .choice:
750 ,role/map-tags/false .code:n = { \socket_assign_plug:nn { tag/struct/tag } {latex-
tags} }
751 ,role/map-tags/pdf .code:n = { \socket_assign_plug:nn { tag/struct/tag } {pdf-
tags} }

```

deprecated names

```

752 , mathml-tags .bool_gset:N = \g__tag_role_add_mathml_bool
753 , add-new-tag .meta:n = {role/new-tag={#1}}
754 }
755 </package>

```

*(End of definition for tag (rolemap-key) and others. These functions are documented on page 159.)*

## Part X

# The tagpdf-space module

## Code related to real space chars

### Part of the tagpdf package

---

`activate/space_□(setup-key)`  
`interwordspace_□(deprecated)`

---

This key allows to activate/deactivate the real space chars if the engine supports it. The allowed values are `true`, `on`, `false`, `off`. The old name of the key `interwordspace` is still supported but deprecated.

---

`show-spaces_□(deprecated)`

---

This key is deprecated. Use `debug/show=spaces` instead. This key works only with luatex and shows with small red bars where spaces have been inserted. This is only for debugging and is not completely reliable (and change affect other literals and tagging), so it should be used with care.

```
1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-space-code} {2024-09-11} {0.99e}
4 {part of tagpdf - code related to real space chars}
5 </header>
```

## 1 Code for interword spaces

The code is engine/backend dependent. Basically only pdftex and luatex support real space chars. Most of the code for luatex which uses attributes is in the lua code, here are only the keys.

`activate/spaces_□(setup-key)`  
`interwordspace_□(deprecated)`  
`show-spaces_□(deprecated)`

```
6 <*package>
7 \bool_new:N\l__tag_showspaces_bool
8 \keys_define:nn { __tag / setup }
9 {
10   activate/spaces .choice:,
11   activate/spaces/true .code:n =
12     { \msg_warning:nne {tag}{sys-no-interwordspace}{\c_sys_engine_str} },
13   activate/spaces/false .code:n=
14     { \msg_warning:nne {tag}{sys-no-interwordspace}{\c_sys_engine_str} },
15   activate/spaces .default:n = true,
16   debug/show/spaces .code:n = {\bool_set_true:N \l__tag_showspaces_bool},
17   debug/show/spacesOff .code:n = {\bool_set_false:N \l__tag_showspaces_bool},
```

deprecated versions:

```
18   interwordspace .choices:nn = {true,on}{\keys_set:nn{__tag/setup}{activate/spaces={true}}},
19   interwordspace .choices:nn = {false,off}{\keys_set:nn{__tag/setup}{activate/spaces={false}}},
20   interwordspace .default:n = {true},
```

```

21 show-spaces .choice:,
22 show-spaces/true .meta:n = {debug/show=spaces},
23 show-spaces/false .meta:n = {debug/show=spacesOff},
24 show-spaces .default:n = true
25 }
26 \sys_if_engine_pdftex:T
27 {
28   \sys_if_output_pdf:TF
29   {
30     \pdfglyphtounicode{space}{0020}
31     \keys_define:nn { __tag / setup }
32     {
33       activate/spaces/true .code:n = { \AddToHook{shipout/firstpage}[tagpdf/space]{\p
34       activate/spaces/false .code:n = { \RemoveFromHook{shipout/firstpage}[tagpdf/spac
35       activate/spaces .default:n = true,
36     }
37   }
38   {
39     \keys_define:nn { __tag / setup }
40     {
41       activate/spaces .choices:nn = { true, false }
42       { \msg_warning:nnn {tag}{sys-no-interwordspace}{dvi} },
43       activate/spaces .default:n = true,
44     }
45   }
46 }
47
48
49 \sys_if_engine_luatex:T
50 {
51   \keys_define:nn { __tag / setup }
52   {
53     activate/spaces .choice:,
54     activate/spaces/true .code:n =
55     {
56       \bool_gset_true:N \g__tag_active_space_bool
57       \lua_now:e{!tx.__tag.func.markspaceon()}
58     },
59     activate/spaces/false .code:n =
60     {
61       \bool_gset_false:N \g__tag_active_space_bool
62       \lua_now:e{!tx.__tag.func.markspaceoff()}
63     },
64     activate/spaces .default:n = true,
65     debug/show/spaces .code:n =
66     { \lua_now:e{!tx.__tag.trace.showspace=true} },
67     debug/show/spacesOff .code:n =
68     { \lua_now:e{!tx.__tag.trace.showspace=nil} },
69   }
70 }

```

(End of definition for `activate/spaces` (setup-key), `interwordspace` (deprecated), and `show-spaces` (deprecated). These functions are documented on page 6.)

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

```

71 \sys_if_engine_luatex:T
72 {
73   \cs_new_protected:Nn \__tag_fakespace:
74   {
75     \group_begin:
76     \lua_now:e{!tx.__tag.func.fakespace()}
77     \skip_horizontal:n{\c_zero_skip}
78     \group_end:
79   }
80 }

```

We need also a command to interrupt the insertion of real space chars in places where we want to insert manually special spaces. In pdftex this can be done with `\pdfinterwordspaceoff` and `\pdfinterwordspaceon`. These commands insert what-sits and this mean they act globally. In luatex a attribute is used to this effect, for consistency this is also set globally.

The off command sets the attributes in luatex.

```

\tag_spacechar_on: 81 \cs_new_protected:Npn \tag_spacechar_off: {}
\tag_spacechar_off: 82 \cs_new_protected:Npn \tag_spacechar_on: {}
83
84 \sys_if_engine_luatex:T
85 {
86   \cs_set_protected:Npn \tag_spacechar_off:
87   {
88     \lua_now:e
89     {
90       tex.setattribute
91       (
92         "global",
93         luatexbase.attributes.g__tag_interwordspaceOff_attr,
94         1
95       )
96     }
97   }
98   \cs_set_protected:Npn \tag_spacechar_on:
99   {
100     \lua_now:e
101     {
102       tex.setattribute
103       (
104         "global",
105         luatexbase.attributes.g__tag_interwordspaceOff_attr,
106         -2147483647
107       )
108     }
109   }
110 }
111 \sys_if_engine_pdftex:T
112 {
113   \sys_if_output_pdf:T
114   {
115     \cs_set_protected:Npn \tag_spacechar_off:
116     {

```

```
117         \pdfinterwordspaceoff
118     }
119     \cs_set_protected:Npn \tag_spacechar_on:
120     {
121         \pdfinterwordspaceon
122     }
123 }
124 }
```

```
125 \endpackage
```

*(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> .....	1084, 1088
<code>\\</code> .....	10, 23, 27, 28, 44, 49, 50, 51, 56, 58, 60, 67, 70, 72, 78, 80, 91, 256, 257, 258, 410, 473, 481
<code>\_</code> .....	422, 433
<b>A</b>	
<code>activate_</code> (setup-key) .....	36, <u>255</u>
<code>activate-all_</code> (deprecated) .....	6
<code>activate-mc_</code> (deprecated) .....	6
<code>activate-struct_</code> (deprecated) .....	6
<code>activate-tree_</code> (deprecated) .....	6
<code>activate/all_</code> (setup-key) .....	6, <u>254</u>
<code>activate/mc_</code> (setup-key) .....	6, <u>254</u>
<code>activate/socket_</code> (setup-key) .....	255
<code>activate/softhyphen_</code> (setup-key) .	6, <u>288</u>
<code>activate/space_</code> (setup-key) .....	181
<code>activate/spaces_</code> (setup-key) .....	6, <u>6</u>
<code>activate/struct_</code> (setup-key) ....	6, <u>254</u>
<code>activate/struct-dest_</code> (setup-key) .	6, <u>254</u>
<code>activate/tagunmarked_</code> (setup-key) .	6, <u>285</u>
<code>activate/tree_</code> (setup-key) .....	6, <u>254</u>
<code>actualtext_</code> (mc-key) .....	71, <u>255</u> , <u>453</u>
<code>actualtext_</code> (struct-key) .....	102, <u>499</u>
<code>add-new-tag_</code> (deprecated) .....	692
<code>add-new-tag_</code> (setup-key) .....	159
<code>\AddToHook</code> .....	13, 16, 33, 58, 71, 78, 107, 273, 357, 379, 510, 512, 513, 517, 521, 528, 557, 606
<code>AF_</code> (struct-key) .....	102, <u>659</u>
<code>AFinline_</code> (struct-key) .....	102, <u>659</u>
<code>AFinline-o_</code> (struct-key) .....	102, <u>659</u>
<code>AFref_</code> (struct-key) .....	102, <u>659</u>
<code>alt_</code> (mc-key) .....	71, <u>255</u> , <u>453</u>
<code>alt_</code> (struct-key) .....	102, <u>499</u>
<code>artifact_</code> (mc-key) .....	71, <u>255</u> , <u>453</u>
artifact-bool internal commands:	
<code>__artifact-bool</code> .....	121
artifact-type internal commands:	
<code>__artifact-type</code> .....	121
<code>attr-unknown</code> .....	20, <u>84</u>
<code>attribute_</code> (struct-key) .....	103, <u>1265</u>
<code>attribute-class_</code> (struct-key) .	103, <u>1231</u>
<b>B</b>	
benchmark commands:	
<code>\benchmark_tic:</code> .....	490, 492
<code>\benchmark_toc:</code> .....	493
bool commands:	
<code>\bool_gset_eq:NN</code> ...	626, 641, 653, 671
<code>\bool_gset_false:N</code> .....	50, 51, 61, 238, 441, 627, 654
<code>\bool_gset_true:N</code> .....	47, 49, 56, 131, 177, 369
<code>\bool_if:NTF</code> .	9, 13, 18, 27, 36, 40, 67, 69, 74, 79, 114, 192, 200, 210, 223, 234, 241, 258, 266, 277, 303, 308, 317, 343, 373, 381, 390, 418, 419, 430, 442, 444, 461, 469, 494, 501, 561, 621, 636, 648, 666, 890, 951
<code>\bool_if:nTF</code> .....	6, 360
<code>\bool_if_exist_p:N</code> .....	44
<code>\bool_lazy_all:nTF</code> .....	116
<code>\bool_lazy_and:nnTF</code> .....	43, 150, 160, 275, 311, 357, 367, 375, 466, 512, 540, 570, 647
<code>\bool_lazy_and_p:nn</code> .....	8
<code>\bool_new:N</code> . .	7, 16, 20, 21, 41, 42, 63, 73, 126, 127, 128, 129, 130, 132, 134, 136, 137, 138, 257, 293, 294, 617
<code>\bool_set_false:N</code> .....	17, 178, 203, 204, 205, 227, 228, 229, 239, 346, 391, 594, 620, 647
<code>\bool_set_true:N</code> .....	16, 133, 135, 213, 214, 215, 237, 238, 239, 258, 348, 390, 593
<code>\box</code> .....	376
box commands:	
<code>\box_dp:N</code> .....	176, 180
<code>\box_ht:N</code> .....	166
<code>\box_new:N</code> .....	121, 122
<code>\box_set_dp:Nn</code> .....	174, 176
<code>\box_set_eq:NN</code> .....	189
<code>\box_set_ht:Nn</code> .....	173, 175
<code>\box_use_drop:N</code> .....	178, 182
<code>\boxmaxdepth</code> .....	82, 177
<b>C</b>	
c@g internal commands:	
<code>\c@g__tag_MCID_abs_int</code> .....	11, 15, 24, 33, 46, 53, 60, 64, 70, 100, 134, 174, 180, 239, 242, 269, 274, 303, 344, 351, 416
<code>\c@g__tag_parenttree_obj_int</code> .	154, 450

<code>\c@g__tag_struct_abs_int</code> . . . . .	252, 252, 256, 259, 261, 265, 267, 271, 275, 279, 282, 287, 294, 306, 311, 311, 314, 315, 316, 319, 320, 322, 323, 327, 333, 337, 339, 341, 341, 346, 351, 355, 380, 383, 387, 393, 394, 401, 415, 416, 421, 427, 429, 436, 444, 451, 460, 486, 487, 488, 515, 559, 565, 589, 594, 601, 602, 602, 603, 604, 615, 618, 628, 632, 645, 661, 661, 689, 690, 795, 796, 797, 1011, 1068, 1138, 1151, 1171, 1175, 1179, 1183, 1189, 1213
cctab commands:	
<code>\c_document_cctab</code> . . . . .	73
<code>\chapter</code> . . . . .	170, 359, 377
clist commands:	
<code>\clist_const:Nn</code> . . . . .	123, 124
<code>\clist_if_empty:NTF</code> . . . . .	1270
<code>\clist_map_inline:nn</code> . . . . .	148, 374, 640
<code>\clist_new:N</code> . . . . .	119
<code>\clist_set:Nn</code> . . . . .	1235, 1269
color commands:	
<code>\color_select:n</code> . . . . .	422, 433
cs commands:	
<code>\cs:w</code> . . . . .	1160, 1164
<code>\cs_end:</code> . . . . .	1160, 1164
<code>\cs_generate_variant:Nn</code> . . . . .	40, 78, 94, 103, 105, 126, 140, 141, 142, 143, 144, 145, 146, 147, 148, 155, 156, 157, 158, 166, 170, 175, 190, 191, 191, 192, 192, 193, 194, 195, 224, 228, 240, 255, 264, 264, 274, 321, 332, 512, 660, 667, 668, 688, 708, 1136, 1148, 1188, 1198, 1219
<code>\cs_gset_eq:NN</code> . . . . .	285, 922, 923, 1054, 1055, 1117, 1118
<code>\cs_if_exist:NTF</code> . . . . .	490, 563, 608
<code>\cs_if_exist_p:N</code> . . . . .	9, 359, 362, 377, 380
<code>\cs_if_exist_use:NTF</code> . . . . .	360, 1142
<code>\cs_if_free:NTF</code> . . . . .	47
<code>\cs_new:Nn</code> . . . . .	. . . . . 80, 81, 107, 129, 134, 349, 369, 370
<code>\cs_new:Npn</code> . . . . .	9, 15, 26, 68, 98, 108, 138, 158, 159, 193, 231, 254, 353, 453, 461, 467, 473, 1132, 1199
<code>\cs_new_protected:Nn</code> . . . . .	. . . . . 73, 127, 169, 352, 371
<code>\cs_new_protected:Npn</code> . . . . .	. . . . . 13, 19, 20, 22, 30, 30, 35, 41, 41, 57, 59, 59, 60, 65, 74, 78, 78, 79, 80, 80, 81, 82, 85, 85, 87, 89, 90, 95, 104, 107, 117, 125, 140, 146, 149, 149, 150, 160, 162, 163, 168, 170, 171, 176, 176, 183, 185, 192, 196, 205, 206, 225, 225, 226, 239, 241, 244, 247, 248, 249, 250, 251,
<code>\cs_set:Nn</code> . . . . .	686, 687
<code>\cs_set:Npn</code> . . . . .	44, 49
<code>\cs_set_eq:NN</code> . . . . .	14, 20, 76, 77, 78, 103, 180, 181, 182, 183, 184, 185, 186, 187, 188, 220, 221, 230, 231, 232, 233, 348, 349, 350, 351, 492, 493, 679, 680, 681, 682, 688, 689, 691, 693, 694, 695, 696, 919, 920, 1051, 1052, 1114, 1115
<code>\cs_set_protected:Nn</code> . . . . .	. . . . . 171, 233, 277, 428, 434, 969, 970
<code>\cs_set_protected:Npn</code> . . . . .	9, 15, 16, 22, 29, 36, 37, 55, 62, 62, 62, 63, 66, 67, 70, 72, 73, 77, 80, 82, 86, 88, 98, 101, 115, 119, 141, 199, 206, 208, 213, 222, 226, 232, 237, 244, 245, 329, 333, 337, 341, 355, 359, 361, 799, 800, 1005, 1013, 1070, 1140
<code>\cs_to_str:N</code> . . . . .	. . . . . 12, 18, 25, 32, 39, 58, 59, 65, 66
<b>D</b>	
<code>debug/log<sub>□</sub>(setup-key)</code> . . . . .	6, 272
<code>debug/show<sub>□</sub>(setup-key)</code> . . . . .	271
<code>debug/structures<sub>□</sub>(show-key)</code> . . . . .	37, 224
<code>debug/uncompress<sub>□</sub>(setup-key)</code> . . . . .	272
<code>\DeclareOption</code> . . . . .	49, 50, 51
dim commands:	
<code>\c_max_dim</code> . . . . .	165, 190
<code>\c_zero_dim</code> . . . . .	173, 174, 175
<code>\documentclass</code> . . . . .	22
<code>\DocumentMetadata</code> . . . . .	21
<b>E</b>	
<code>E<sub>□</sub>(struct-key)</code> . . . . .	102, 499
<code>\endinput</code> . . . . .	28
<code>\ERRORusetaggingsocket</code> . . . . .	89
<code>exclude-header-footer<sub>□</sub>(deprecated)</code> . . . . .	674
<code>\ExecuteOptions</code> . . . . .	52
exp commands:	
<code>\exp_args:Ne</code> . . . . .	118, 455
<code>\exp_args:NNe</code> . . . . .	82, 85, 94, 191, 211

<code>\exp_args:Nne</code>	79, 340, 344, 429, 441, 492
<code>\exp_args:NNne</code>	94
<code>\exp_args:NNno</code>	714
<code>\exp_args:NV</code>	196, 202, 347, 376, 387, 392
<code>\exp_last_unbraced:NV</code>	184, 185, 240, 241, 459, 463, 994
<code>\exp_not:n</code>	225, 243, 308
<b>F</b>	
file commands:	
<code>\file_if_exist:nTF</code>	324
<code>\file_input:n</code>	304
flag commands:	
<code>\flag_clear:n</code>	236
<code>\flag_height:n</code>	178, 248
<code>\flag_new:n</code>	176
<code>\flag_raise:n</code>	249
fnote internal commands:	
<code>\__fnote_gput_ref:nn</code>	73
<code>\fontencoding</code>	6
<code>\fontfamily</code>	6
<code>\fontseries</code>	6
<code>\fontshape</code>	6
<code>\fontsize</code>	6
<code>\footins</code>	566
<b>G</b>	
g internal commands:	
<code>\g_tag_struct_ref_by_dest:</code>	80
group commands:	
<code>\group_begin:</code>	66, 75, 175, 367, 481, 666, 756, 764, 805
<code>\group_end:</code>	73, 78, 230, 419, 500, 684, 760, 768, 965
<b>H</b>	
<code>\hangindent</code>	374
<code>\hbox</code>	365
hbox commands:	
<code>\hbox_set:Nn</code>	167, 168
hook commands:	
<code>\hook_gput_code:nnn</code>	7, 11, 33, 57, 65, 79, 155, 236, 258, 259, 355, 373, 379, 383, 702, 715, 725, 738
<code>\hook_new:n</code>	340
<code>\hook_use:n</code>	345
<b>I</b>	
<code>\ignorespaces</code>	36
int commands:	
<code>\int_abs:n</code>	143
<code>\int_case:nnTF</code>	82, 289
<code>\int_compare:nNnTF</code>	22, 57, 68, 97, 112, 118, 132, 137, 169, 172, 185, 211, 215, 234, 261, 264, 289, 295, 382, 389, 396, 401, 403, 412, 418, 423, 431, 438, 446, 453, 476, 482, 534, 543, 684, 837, 903, 1049, 1112
<code>\int_compare:nTF</code>	179, 329, 1251, 1253, 1255, 1279, 1305
<code>\int_compare_p:nNn</code>	517
<code>\int_decr:N</code>	210, 234
<code>\int_eval:n</code>	134, 190, 291, 308, 388, 474, 482, 514, 519, 522, 681, 722, 741, 807, 808, 809, 810, 811, 917, 945, 946, 948, 959, 1261
<code>\int_gincr:N</code>	180, 239, 269, 313, 317, 321, 325, 331, 335, 339, 343, 344, 351, 450, 667, 795, 806
<code>\int_gset:Nn</code>	7, 81, 157, 287
<code>\int_gzero:N</code>	295
<code>\int_if_zero:nTF</code>	210, 211, 234, 235, 470, 478
<code>\int_incr:N</code>	92, 202, 226, 410
<code>\int_new:N</code>	77, 120, 125, 198, 263, 296, 297, 298, 299, 659
<code>\int_rand:n</code>	61, 62, 64, 66, 68, 70, 71
<code>\int_set:Nn</code>	273, 276, 279, 280, 281
<code>\int_step_inline:nn</code>	441, 447
<code>\int_step_inline:nnn</code>	25, 90, 229
<code>\int_step_inline:nnnn</code>	148, 173, 176, 193, 314, 320
<code>\int_to_arabic:n</code>	143, 145
<code>\int_to_Hex:n</code>	61, 62, 64, 66, 68, 70, 71
<code>\int_use:N</code>	11, 15, 18, 24, 33, 39, 46, 53, 58, 60, 64, 70, 73, 83, 99, 100, 101, 135, 137, 165, 172, 174, 201, 218, 225, 234, 242, 243, 251, 266, 274, 303, 355, 416, 422, 433, 475, 504, 539, 539, 540, 548, 549, 551, 565, 581, 589, 643, 654, 670, 673, 676, 715, 717, 734, 736, 815, 821, 826, 833, 836, 853, 871, 880, 925, 930, 1057, 1120, 1199, 1258, 1309
<code>\int_zero:N</code>	89, 104, 398
intarray commands:	
<code>\intarray_gset:Nnn</code>	292, 395
<code>\intarray_item:Nn</code>	294, 297, 472
<code>\intarray_new:Nn</code>	284, 392
<code>interwordspace<sub>□</sub></code> (deprecated)	181, <u>6</u>
ior commands:	
<code>\ior_close:N</code>	332, 435
<code>\ior_map_inline:Nn</code>	328, 406
<code>\ior_open:Nn</code>	326, 401, 404
<code>\g_tmpa_ior</code>	326, 328, 332, 401, 404, 406, 435
iow commands:	
<code>\iow_newline:</code>	201, 296, 496
<code>\iow_now:Nn</code>	94

<code>\iow_term:n</code>	181, 184, 190, 194, 194, 277, 347, 351, 355, 359, 363, 367, 371	<code>ltx.__tag.trace.log</code>	188
<b>K</b>		<code>ltx.__tag.trace.show_all_mc_data</code>	245
kernel internal commands:		<code>ltx.__tag.trace.show_mc_data</code>	230
<code>\__kernel_pdfdict_name:n</code>	44	<code>ltx.__tag.trace.show_prop</code>	205
keys commands:		<code>ltx.__tag.trace.show_seq</code>	196
<code>\keys_define:nn</code>	8, 31, 33, 39, 51, 99, 111, 121, 173, 216, 225, 255, 255, 261, 387, 396, 403, 409, 454, 499, 597, 636, 674, 692, 700, 709, 771, 1220, 1231, 1265	<code>ltx.__tag.trace.show_struct_data</code>	251
<code>\keys_set:nn</code>	10, 18, 18, 19, 96, 187, 266, 341, 345, 372, 493, 831	lua commands:	
<code>\keys_set_known:nnnN</code>	705	<code>\lua_escape:n</code>	32
<b>L</b>		<code>\lua_now:n</code>	8, 12, 15, 18, 25, 26, 32, 35, 39, 42, 46, 50, 51, 57, 58, 59, 59, 62, 65, 66, 66, 68, 71, 76, 84, 85, 87, 88, 94, 100, 105, 109, 109, 118, 122, 130, 131, 136, 142, 156, 183, 230, 247, 261, 269, 285, 306, 320, 330
<code>\label</code>	11	<b>M</b>	
<code>label_(mc-key)</code>	71, 255, 453	<code>\MakeLinkTarget</code>	131
<code>label_(struct-key)</code>	101, 499	<code>mathml</code>	102
<code>lang_(struct-key)</code>	102, 499	<code>\maxdimen</code>	188
legacy commands:		<code>mc-current</code>	19, 16
<code>\legacy_if:nTF</code>	92, 463, 466, 467	<code>mc-current_(show-key)</code>	37, 111
<code>\llap</code>	422	<code>mc-data_(show-key)</code>	37, 99
<code>log_(deprecated)</code>	272	<code>mc-label-unknown</code>	19, 9
ltx. internal commands:		<code>mc-marks_(show-key)</code>	37, 173
<code>ltx.__tag.func.alloctag</code>	276	<code>mc-nested</code>	19, 6
<code>ltx.__tag.func.fakespace</code>	455	<code>mc-not-open</code>	19, 13
<code>ltx.__tag.func.fill_parent_tree_- line</code>	825	<code>mc-popped</code>	19, 14
<code>ltx.__tag.func.get_num_from</code>	285	<code>mc-pushed</code>	19, 14
<code>ltx.__tag.func.get_tag_from</code>	304	<code>mc-tag-missing</code>	19, 8
<code>ltx.__tag.func.mark_page_- elements</code>	656	<code>mc-used-twice</code>	19, 12
<code>ltx.__tag.func.mark_shipout</code>	808	<code>\MessageBreak</code>	15, 19, 20, 21
<code>ltx.__tag.func.markspaceoff</code>	521	msg commands:	
<code>ltx.__tag.func.markspaceon</code>	521	<code>\msg_error:nn</code>	173, 194, 425, 843
<code>ltx.__tag.func.mc_insert_kids</code>	593	<code>\msg_error:nnn</code>	210, 221, 229, 240, 275, 412, 1245, 1285
<code>ltx.__tag.func.mc_num_of_kids</code>	334	<code>\msg_error:nnnnn</code>	536, 545
<code>ltx.__tag.func.output_num_from</code>	285	<code>\msg_info:nnn</code>	134, 174, 187, 263, 267, 327, 335
<code>ltx.__tag.func.output_parenttree</code>	825	<code>\msg_info:nnnn</code>	217, 236
<code>ltx.__tag.func.output_tag_from</code>	304	<code>\msg_line_context:</code>	91, 377, 378, 410, 414, 418, 474, 482
<code>ltx.__tag.func.space_chars_- shipout</code>	553	<code>\g_msg_module_name_prop</code>	30, 34
<code>ltx.__tag.func.store_mc_data</code>	319	<code>\g_msg_module_type_prop</code>	33
<code>ltx.__tag.func.store_mc_in_page</code>	637	<code>\msg_new:nnn</code>	7, 8, 9, 12, 13, 14, 15, 16, 22, 24, 25, 32, 35, 36, 38, 40, 42, 47, 54, 65, 74, 85, 86, 87, 88, 89, 90, 92, 94, 95, 96, 97, 98, 99, 101, 254, 377, 378, 408, 412, 416, 468, 476
<code>ltx.__tag.func.store_mc_kid</code>	328	<code>\msg_new:nnnn</code>	104
<code>ltx.__tag.func.store_mc_label</code>	324	<code>\msg_note:nn</code>	28, 169
<code>ltx.__tag.func.store_struct_- mcabs</code>	625	<code>\msg_note:nnn</code>	201, 218, 398, 405, 440, 448
<code>ltx.__tag.func.update_mc_- attributes</code>	645	<code>\msg_note:nnnn</code>	224, 242, 384, 391, 425, 433
<code>ltx.__tag.tables.role_tag_- attribute</code>	274		

<code>\msg_note:nnnnn</code> .....	489	<code>\l_pdf_current_structure_-</code>	
<code>\msg_redirect_name:nnn</code> .....	532	<code>destination_tl</code> .....	279
<code>\msg_show_item_unbraced:n</code> .....	246	<code>\pdf_emc:</code> .....	231
<code>\msg_show_item_unbraced:nn</code> .....	237	<code>\pdf_name_from_unicode_e:n</code> .....	98, 108, 113,
<code>\msg_term:nnnnnn</code> .....	231, 240	156, 165, 192, 271, 1216, 1239, 1275	
<code>\msg_warning:nn</code> .....	24, 215, 261	<code>\pdf_object_if_exist:n</code> .....	139
<code>\msg_warning:nnn</code> .....		<code>\pdf_object_if_exist:nTF</code> ...	713, 775
..... 12, 14, 42, 44, 53, 180, 203,		<code>\pdf_object_new:n</code> .....	102, 34, 36, 153, 255, 302, 313
244, 248, 250, 256, 279, 302, 323,		<code>\pdf_object_new_indexed:nn</code> ..	30, 812
612, 616, 625, 643, 1063, 1082, 1126		<code>\pdf_object_ref:n</code> ...	102, 56, 130,
<code>\msg_warning:nnnn</code> .....	393, 461, 521	130, 134, 140, 194, 310, 327, 715, 777	
<code>\msg_warning:nnnnn</code> .....		<code>\pdf_object_ref_indexed:nn</code> .....	56, 73, 95, 168, 204,
..... 217, 407, 504, 551, 581, 658, 910		231, 389, 447, 933, 1030, 1093, 1134	
<b>N</b>			
<code>namespace_□(rolemap-key)</code> .....	159	<code>\pdf_object_ref_last:</code> .....	102, 103, 117, 123, 254, 1294
<code>new-tag</code> .....	20, 94	<code>\pdf_object_unnamed_write:nn</code> ...	99, 110, 119, 246, 1289
<code>newattribute_□(deprecated)</code> .....	103, 1213	<code>\pdf_object_write:nnn</code> .....	250, 274, 303, 322, 329, 334
<code>\newcommand</code> .....	590, 591	<code>\pdf_object_write_indexed:nnnn</code> ..	138, 402
<code>\newcounter</code> .....	6, 8, 154	<code>\pdf_pageobject_ref:n</code> .....	199, 438
<code>\NewDocumentCommand</code> .....	6,	<code>\pdf_string_from_unicode:nnN</code> ...	42
23, 29, 34, 40, 46, 51, 56, 94, 286, 595		<code>\pdf_uncompress:</code> .....	282, 284
<code>\newmarks</code> .....	13	<code>\pdf_version_compare:NnTF</code> .....	20, 81, 125, 148, 156,
<code>no-struct-dest_□(deprecated)</code> .....	6	229, 259, 316, 353, 399, 439, 513, 734	
<code>\noindent</code> .....	374	pdfannot commands:	
<code>\nointerlineskip</code> .....	181	<code>\pdfannot_dict_put:nnn</code> .....	141, 709, 732, 750, 755
<b>P</b>			
<code>\PackageError</code> .....	13	<code>\pdfannot_link_ref_last:</code> ...	719, 742
<code>\PackageWarning</code> .....	28, 554	pdfdict commands:	
<code>page/exclude-header-footer_□(setup-</code>		<code>\pdfdict_gput:nnn</code> .....	38, 45, 53, 189, 269, 326
<code>key)</code> .....	39, 674	<code>\pdfdict_if_empty:nTF</code> .....	320
<code>page/tabsorder_□(setup-key)</code> .....	6, 290	<code>\pdfdict_new:n</code> .....	18, 35, 37
<code>para-flattened_□(deprecated)</code> .....	387	<code>\pdfdict_put:nnn</code> ...	757, 758, 765, 766
<code>para-hook-count-wrong</code> .....	20, 104	<code>\pdfdict_use:n</code> .....	276, 324, 331
<code>para/flattened_□(tool-key)</code> .....	387	<code>\pdffakespace</code> .....	38, 284
<code>para/maintag_□(setup-key)</code> .....	387	pdffile commands:	
<code>para/maintag_□(tool-key)</code> .....	387	<code>\pdffile_embed_stream:nnN</code> ..	660, 668
<code>para/tag_□(setup-key)</code> .....	387	<code>\pdffile_embed_stream:nnn</code> .....	142
<code>para/tag_□(tool-key)</code> .....	387	<code>\pdfglyphptounicode</code> .....	30
<code>para/tagging_□(setup-key)</code> .....	38, 387	<code>\pdfinterwordspaceoff</code> .....	183, 117
<code>para/tagging_□(tool-key)</code> .....	387	<code>\pdfinterwordspaceon</code> .....	183, 33, 121
<code>\PARALABEL</code> .....	487	pdfmanagement commands:	
<code>paratag_□(deprecated)</code> .....	387	<code>\pdfmanagement_add:nnn</code> .....	51, 69, 70, 292, 294, 296, 385
<code>paratagging_□(deprecated)</code> .....	38, 387	<code>\pdfmanagement_if_active_p:</code> ..	9, 10
<code>paratagging-show_□(deprecated)</code> ..	38, 387	<code>\pdfmanagement_remove:nn</code> .....	298
<code>parent_□(struct-key)</code> .....	101, 499		
pdf commands:			
<code>\pdf_activate_indexed_structure_-</code>			
<code>destination:</code> .....	281		
<code>\pdf_bdc:nn</code> .....	232		
<code>\pdf_bdc_shipout:nn</code> .....	233		
<code>\pdf_bmc:n</code> .....	230		

pdfmanagement internal commands:	
\l__pdfmanagement_delayed_	
shipout_bool	44, 45
prg commands:	
\prg_do_nothing:	78, 85, 285, 348, 349, 350, 351, 693, 694, 695, 696
\prg_generate_conditional_	
variant:Nnn	139
\prg_new_conditional:Nnn	66, 221
\prg_new_conditional:Npnn	110, 133, 148, 158, 352, 358, 369
\prg_new_eq_conditional:NNn	80, 228
\prg_new_protected_conditional:Npnn	670
\prg_replicate:nn	142
\prg_return_false:	76, 111, 128, 139, 142, 155, 165, 225, 355, 367, 373, 685
\prg_return_true:	77, 125, 138, 152, 162, 224, 356, 366, 372, 670, 686
\prg_set_conditional:Npnn	114
\prg_set_protected_conditional:Npnn	672
process commands:	
process_softhyphen_pre	
softhyphen_post	878
\ProcessOptions	53
prop commands:	
\prop_clear:N	175
\prop_count:N	196
\prop_get:NnN	137, 145, 179, 200, 213, 216, 270, 301, 403, 424, 436, 438, 451, 452, 464, 465, 567, 568, 905
\prop_get:NnNTF	43, 92, 160, 166, 179, 198, 198, 213, 232, 238, 279, 387, 484, 520, 525, 530, 535, 600, 604, 617, 627, 720, 865, 995, 1077, 1154
\prop_gput:Nnn	26, 30, 31, 33, 34, 56, 88, 89, 90, 91, 92, 94, 97, 97, 98, 99, 99, 100, 110, 111, 112, 113, 119, 120, 121, 122, 143, 144, 148, 151, 163, 183, 203, 207, 219, 222, 262, 284, 284, 287, 288, 316, 328, 385, 386, 389, 390, 416, 437, 443, 449, 455, 457, 503, 947, 958, 1032, 1095, 1215, 1247, 1294
\prop_gremove:Nn	136, 136
\prop_gset_eq:NN	135, 944
\prop_if_exist:NTF	176, 202, 236, 322, 385, 598, 625, 1017, 1074
\prop_if_exist_p:N	514
\prop_if_in:NnTF	70, 130, 170, 178, 277, 725, 1243, 1283, 1287
\prop_item:Nn	41, 74, 145, 185, 186, 221, 292, 396, 458, 477, 482, 490, 955, 1292, 1299
\prop_map_function:NN	235
\prop_map_inline:Nn	82, 260, 265, 286, 318, 358, 365, 383, 453
\prop_map_tokens:Nn	336
\prop_new:N	7, 8, 9, 10, 11, 11, 19, 24, 25, 32, 33, 116, 133, 180, 808, 1208, 1211
\prop_new_linked:N	17, 62, 65, 67, 181, 1209
\prop_put:Nnn	144, 182, 518, 519, 591, 592
\prop_show:N	64, 91, 188, 941, 962, 1261, 1288
property commands:	
\property_gset:nnnn	265
\property_new:nnnn	163, 166, 170, 173, 177
\property_record:nn	152
\property_ref:nn	101, 157
\property_ref:nnn	41, 156, 161, 180, 184, 199, 199, 200, 272, 331, 342, 438, 1018, 1024, 1027, 1033, 1040
\providecommand	62, 63, 64, 291, 559, 560
\ProvidesExplFile	3
\ProvidesExplPackage	3, 3, 3, 3, 3, 3, 3, 3, 7, 7, 26, 37, 1204
<b>Q</b>	
\quad	203, 204
quark commands:	
\q_no_value	527, 537, 612, 617, 639, 644
\quark_if_no_value:NTF	138, 146, 180, 201, 217, 271, 302, 909
\quark_if_no_value_p:N	467, 468, 541, 542, 571, 572, 648, 649
\q_stop	261, 294, 330
<b>R</b>	
raw_(mc-key)	71, 255, 453
ref_(struct-key)	102, 499
\RemoveFromHook	34, 515, 516
\renewcommand	593, 594
\RenewDocumentCommand	8
\RequirePackage	20, 54, 310, 313, 319, 322, 555
\rlap	433
role_(rolemap-key)	159, 692
role-missing	20, 86
role-namespace_(rolemap-key)	159, 692
role-parent-child	20, 90
role-remapping	20, 92

role-tag	20, <a href="#">94</a>	show-kids	20, <a href="#">64</a>
role-unknown	20, <a href="#">86</a>	show-spaces <sub>□</sub> (deprecated)	181, <a href="#">6</a>
role-unknown-NS	20, <a href="#">86</a>	show-struct	20, <a href="#">64</a>
role-unknown-tag	20, <a href="#">86</a>	\ShowTagging	17, <a href="#">37</a> , <a href="#">93</a>
role/new-attribute <sub>□</sub> (setup-key)	103, <a href="#">1213</a>	skip commands:	
role/new-tag <sub>□</sub> (setup-key)	692	\skip_horizontal:n	77
root-AF <sub>□</sub> (setup-key)	103, <a href="#">771</a>	\c_zero_skip	77
<b>S</b>			
\selectfont	6	socket commands:	
seq commands:		\socket_assign_plug:nn	498, 508, 509, 525, 750, 751
\seq_clear:N	279, 319	\socket_new:nn	437, 438, 479
\seq_const_from_clist:Nn	21, 34	\socket_new_plug:nnn	440, 459, 480, 488, 492
\seq_count:N	22, 25, 57, 291, 397, 1251, 1253, 1255, 1279, 1305	\socket_use:n	510, 512, 519, 523
\seq_get:NN	674	\socket_use:nn	529
\seq_get:NNTF	421, 455, 839, 984, 991	stash <sub>□</sub> (mc-key)	71, <a href="#">121</a>
\seq_gpop:NN	977	stash <sub>□</sub> (struct-key)	101, <a href="#">499</a>
\seq_gpop:NNTF	105, 978	str commands:	
\seq_gpop_left:NN	267	\str_case:nnTF	60, 858
\seq_gpush:Nn	13, 15, 88, 95, 846, 886	\str_const:Nn	59
\seq_gput_left:Nn	271	\str_if_empty:nTF	653
\seq_gput_right:Nn	38, 144, 150, 184, 213, 233, 256, 339	\str_if_eq:nnTF	123, 371, 457, 569
\seq_gset_eq:NN	155, 217, 286	\str_if_eq_p:nn	312, 362, 364
\seq_if_empty:NTF	196, 391	\str_new:N	115
\seq_item:Nn	58, 112, 114, 121, 125, 132, 136, 185, 308, 315, 328, 362, 364, 371, 483, 484, 491, 492, 715, 716	\str_set_convert:Nnnn	147, 278, 299, 467, 480, 533, 545, 559, 575, 648
\seq_log:N	171, 195, 219, 268, 426, 441	\str_use:N	289, 312
\seq_map_function:NN	244	\c_tilde_str	57, 59
\seq_map_indexed_inline:Nn	414, 427	\string	20, 21, 22, 579
\seq_map_inline:Nn	280, 282, 1241, 1281	struct-faulty-nesting	20, 32
\seq_new:N	12, 14, 14, 15, 16, 17, 17, 18, 18, 24, 117, 118, 134, 182, 811, 1212	struct-label-unknown	20, <a href="#">38</a>
\seq_pop_left:NN	423, 425, 426	struct-missing-tag	20, <a href="#">35</a>
\seq_put_right:Nn	281	struct-no-objnum	20, <a href="#">24</a>
\seq_remove_all:Nn	284	struct-orphan	20, <a href="#">25</a>
\seq_set_eq:NN	203, 204	struct-Ref-unknown	42
\seq_set_from_clist:NN	1236, 1272	struct-show-closing	20, <a href="#">40</a>
\seq_set_from_clist:Nn	83, 86, 192, 212, 411, 422	struct-stack <sub>□</sub> (show-key)	37, <a href="#">216</a>
\seq_set_map_e:NNn	1237, 1273	struct-unknown	20, <a href="#">22</a>
\seq_set_split:Nnn	50, 146, 482, 490, 714	struct-used-twice	20, <a href="#">36</a>
\seq_show:N	57, 186, 187, 187, 220, 282, 283, 285, 349, 889, 942, 963, 973	\SuspendTagging	40
\seq_use:Nn	50, 106, 107, 201, 203, 204, 343, 1252	sys commands:	
\l_tmpa_seq	319, 339, 349, 714, 715, 716	\c_sys_backend_str	60
\setbox	364	\c_sys_engine_str	12, 14
shipout commands:		\sys_if_engine luatex:TF	49, 49, 71, 83, 84, 103, 105, 116, 284, 302
\g_shipout_readonly_int	99, 172, 234, 388	\sys_if_engine pdftex:TF	26, 111
		\sys_if_output_pdf:TF	11, 28, 113
		sys-no-interwordspace	20, <a href="#">101</a>
<b>T</b>			
tabsorder <sub>□</sub> (deprecated)	6, <a href="#">290</a>	tag <sub>□</sub> (mc-key)	71, <a href="#">255</a> , <a href="#">453</a>
tag <sub>□</sub> (rolemap-key)	159, <a href="#">692</a>	tag <sub>□</sub> (struct-key)	101, <a href="#">499</a>



tag commands:

- \tag\_check\_benchmark\_on: . . . . . 488
- \tag\_check\_child:nn . . . 159, 670, 672
- \tag\_check\_child:nnTF . . . . 159, 670
- \tag\_get:n . . . . . 17, 72,  
100, 101, 116, 117, 88, 91, 108, 108, 410
- \tag\_if\_active: . . . . . 110, 114
- \tag\_if\_active:TF . . . 17, 18, 109, 530
- \tag\_if\_active\_p: . . . . . 17, 109, 368
- \tag\_if\_box\_tagged:N . . . . . 17, 133
- \tag\_if\_box\_tagged:NTF . . . . . 17, 132
- \tag\_if\_box\_tagged\_p:N . . . . . 17, 132
- \tag\_mc\_artifact\_group\_begin:n . . . . . 70, 59, 59, 62
- \tag\_mc\_artifact\_group\_end: . . . . . 70, 59, 60, 70
- \tag\_mc\_begin:n . . . 10, 70, 25, 65,  
113, 171, 171, 351, 351, 355, 361,  
421, 432, 456, 488, 629, 657, 708, 731
- \tag\_mc\_begin\_pop:n . . . . . 70,  
75, 79, 80, 101, 638, 668, 722, 745
- \tag\_mc\_end: . . . . . 70,  
31, 74, 92, 233, 233, 351, 352, 423,  
428, 434, 434, 498, 635, 664, 720, 743
- \tag\_mc\_end\_push: . . . . . 70, 64, 79, 79, 82, 623, 650, 706, 729
- \tag\_mc\_if\_in: . . . . . 80, 228
- \tag\_mc\_if\_in:TF . . . . . 70, 42, 66, 221
- \tag\_mc\_if\_in\_p: . . . . . 70, 66, 221
- \tag\_mc\_reset\_box:N 71, 78, 78, 245, 245
- \tag\_mc\_use:n . . . . . 70, 35, 35, 36, 37
- \l\_tag\_para\_attr\_class\_tl . . 381, 383
- \tag\_socket\_use:n . . 39, 40, 62, 66, 67
- \tag\_socket\_use:nn . . 39, 40, 63, 66, 72
- \tag\_spacechar\_off: . . 81, 81, 86, 115
- \tag\_spacechar\_on: . . . 81, 82, 98, 119
- \tag\_start: . . . . . 6, 197, 208, 221, 248
- \tag\_start:n . . . . . 6, 72, 197, 232, 252, 375, 634, 663
- \tag\_stop: . . . 6, 48, 197, 199, 220, 247
- \tag\_stop:n . . . . . 6, 67, 197, 222, 251, 373, 630, 658
- \tag\_struct\_begin:n . . . . . 100, 48, 447, 454, 472,  
482, 656, 707, 730, 795, 795, 799, 800
- \tag\_struct\_end: . . . . . 100, 26, 53, 500, 504,  
665, 721, 744, 795, 796, 969, 970, 1008
- \tag\_struct\_end:n . . . . 100, 797, 1005
- \tag\_struct\_gput:nnn . . . . . 101,  
75, 84, 642, 1138, 1138, 1140, 1148
- \tag\_struct\_gput\_ref:nnn . . . . . 101
- \tag\_struct\_insert\_annot:nn . . . . . 100, 132, 719, 742, 1189, 1189, 1198
- \tag\_struct\_object\_ref:n . . . . . 100, 608, 621, 632, 1131, 1132, 1136
- \tag\_struct\_parent\_int: . . . . 100,  
132, 712, 719, 735, 742, 1189, 1199
- \tag\_struct\_use:n . . . . . 100, 101, 58, 1011, 1011, 1013
- \tag\_struct\_use\_num:n . . . . . 100, 1068, 1068, 1070
- \tag\_tool:n . . . . . 36, 13, 13, 14, 16, 20

tag internal commands:

- \_\_tag\_activate\_mark\_space . . . . . 521
- \g\_\_tag\_active\_mc\_bool . . . . . 40, 119, 126, 150, 257, 264
- \l\_\_tag\_active\_mc\_bool . . . . . 122, 132, 150, 204, 214, 228, 238
- \l\_\_tag\_active\_socket\_bool . . 69,  
74, 79, 132, 205, 215, 229, 239, 263
- \g\_\_tag\_active\_space\_bool . . . . . 13, 56, 61, 126
- \g\_\_tag\_active\_struct\_bool . . . . . 118, 126, 160, 259, 266, 277, 418
- \l\_\_tag\_active\_struct\_bool . . . . . 121, 132, 160, 203, 213, 227, 237
- \g\_\_tag\_active\_struct\_dest\_bool . . . . . 126, 263, 270, 276
- \g\_\_tag\_active\_tree\_bool . . . . . 9, 67, 120, 126, 258, 265, 343, 381
- \_\_tag\_add\_missing\_mcs:Nn . . . . . 83, 163, 163, 215
- \_\_tag\_add\_missing\_mcs\_to\_-  
stream:Nn . . . . . 65,  
65, 185, 185, 566, 570, 575, 582, 584
- \g\_\_tag\_attr\_class\_used\_prop . . . . . 284, 286, 1207, 1247
- \g\_\_tag\_attr\_class\_used\_seq 282, 1212
- \g\_\_tag\_attr\_entries\_prop . . . . . 293, 1207, 1215, 1243, 1283, 1288, 1292
- \_\_tag\_attr\_new\_entry:nn . . . . . 644, 1213, 1213, 1219, 1224, 1228
- \g\_\_tag\_attr\_objref\_prop . . . . . 1207, 1287, 1294, 1299
- \l\_\_tag\_attr\_value\_tl . . . . . 1207,  
1277, 1296, 1301, 1303, 1307, 1311
- \_\_tag\_backend\_create\_bdc\_node . . 400
- \_\_tag\_backend\_create\_bmc\_node . . 371
- \_\_tag\_backend\_create\_emc\_node . . 342
- \_\_tag\_check\_add\_tag\_role:nn . . . . . 129, 206, 206
- \_\_tag\_check\_add\_tag\_role:nnn . . . . . 171, 225
- \_\_tag\_check\_benchmark\_tic: 348,  
352, 356, 360, 364, 368, 372, 486, 492
- \_\_tag\_check\_benchmark\_toc: 350,  
354, 358, 362, 366, 370, 374, 487, 493



\_tag_check_if_active_mc: . . . .	148	\_tag_check_struct_used:n . . . . .	196, 196, 1020
\_tag_check_if_active_mc:TF . . . .		\_tag_check_structure_has_tag:n . . . . .	168, 168, 836
. . . . .	84, 103,	\_tag_check_structure_tag:N . . . . .	176, 176, 485, 496
. . . . .	147, 173, 187, 235, 357, 363, 430, 436	\_tag_check_timeout_v:n <u>103</u> , 103,	106, 107, 110, 145, 153, 160, 198,
\_tag_check_if_active_struct: . . . .	158	. . . . .	207, 277, 465, 481, 497, 569, 574, 579
\_tag_check_if_active_struct:TF . . . . .	39, <u>147</u> , 802,	\_tag_debug_mc_begin_ignore:n . . . . .	387, 423
. . . . .	803, 974, 975, 1007, 1015, 1072, 1192	\_tag_debug_mc_begin_insert:n . . . . .	365, 380
\_tag_check_if_mc_in_galley: . . . .	352	\_tag_debug_mc_end_ignore: 401, 448	
\_tag_check_if_mc_in_galley:TF . . . . .	179, 200	\_tag_debug_mc_end_insert: 394, 438	
\_tag_check_if_mc_tmb_missing: 358		\_tag_debug_struct_begin_ign ignore:n . . . . .	429, 967
\_tag_check_if_mc_tmb_missing:TF . . . . .	108, 188, 205, <u>358</u>	\_tag_debug_struct_begin_ig insert:n . . . . .	421, 964
\_tag_check_if_mc_tmb_missing_- p: . . . . .	<u>358</u>	\_tag_debug_struct_end_check:n . . . . .	451, 1007
\_tag_check_if_mc_tme_missing: 369		\_tag_debug_struct_end_ignore: . . . . .	444, 1002
\_tag_check_if_mc_tme_missing:TF . . . . .	151, 192, 209, <u>369</u>	\_tag_debug_struct_end_insert: . . . . .	436, 1000
\_tag_check_if_mc_tme_missing_- p: . . . . .	<u>369</u>	\g_tag_delayed_shipout_bool . . . . .	42, 47, 51, 234
\_tag_check_info_closing_- struct:n . . . . .	<u>183</u> , 183, 191, 980	\_tag_exclude_headfoot_begin: . . . . .	618, 679, 680
\_tag_check_init_mc_used: . . . . .	<u>282</u> , 282, 285, 291	\_tag_exclude_headfoot_end: . . . . .	632, 681, 682
\_tag_check_mc_if_nested: . . . . .	176, <u>244</u> , 244, 368	\_tag_exclude_struct_headfoot_ begin:n . . . . .	645, 686, 687
\_tag_check_mc_if_open: . . . . .	237, <u>244</u> , 252, 440	\_tag_exclude_struct_headfoot_ end: . . . . .	661, 688, 689
\_tag_check_mc_in_galley:TF . . . .	352	__tag_fakespace . . . . .	<u>455</u>
\_tag_check_mc_in_galley_p: . . . .	<u>352</u>	\_tag_fakespace: . . . . .	<u>71</u> , 73, 288
\_tag_check_mc_pushed_popped:nn . . . . .	89, 96, 109, 112, 117, <u>259</u> , 259	\_tag_finish_structure: . . . . .	13, 16, <u>340</u> , 341
\_tag_check_mc_tag:N . . . . .	189, <u>271</u> , 271, 380	\_tag_get_data_mc_counter: . . . . .	9, 9
\_tag_check_mc_used:n . . . . .	143, <u>287</u> , 287, 324	\_tag_get_data_mc_tag: . . . . .	254, 254, <u>349</u> , 349
\g_tag_check_mc_used_intarray . . . . .	<u>282</u> , 292, 294, 297	\_tag_get_data_struct_counter: . . . . .	472, 473
\_tag_check_no_open_struct: . . . . .	<u>192</u> , 192, 982, 989	\_tag_get_data_struct_id: . . . . .	461, 461
\_tag_check_para_begin_show:nn . . . . .	416, 455, 487	\_tag_get_data_struct_num: <u>466</u> , 467	
\_tag_check_para_end_show:nn . . . . .	427, 499	\_tag_get_data_struct_tag: <u>453</u> , 453	
\_tag_check_parent_child:nnN . . . . .	559, 565, 667	__tag_get_mathsubtype . . . . .	<u>266</u>
__tag_check_parent_child:nnnnN . . . . .	<u>513</u>	\_tag_get_mc_abs_cnt: . . . . .	14, 15, 19, 20,
\_tag_check_parent_child:nnnnN . . . . .	206, 396, 515,	. . . . .	100, 105, 135, 146, 185, 227, 248,
. . . . .	561, 574, 589, 668, 679, 897, 1043, 1106	. . . . .	256, 263, 271, 275, 289, 310, 324, 334
\_tag_check_show_MCID_by_page: . . . . .	<u>306</u> , 306	__tag_get_mc_cnt_type_tag . . . . .	<u>260</u>
		__tag_get_num_from . . . . .	<u>285</u>

\l__tag_get_parent_tmpa_tl . . . . .	\__tag_mc_begin_marks:nn . . . . .
. . . . . 113, 204, 207, 220,	. . . . . 19, 19, 40, 76, 384
394, 397, 410, 677, 680, 895, 898, 913	\__tag_mc_bmc:n . . . . . 229, 230, 335
\l__tag_get_parent_tmpa_tl\l__-	\__tag_mc_bmc_artifact: 333, 333, 346
_tag_get_parent_tmpb_tl\l__-	\__tag_mc_bmc_artifact:n 333, 337, 347
_tag_tmpa_str . . . . . 109	\l__tag_mc_botmarks_seq . . . . .
\l__tag_get_parent_tmpb_tl . . . . .	. . . . . 83, 17, 86, 107,
. . . . . 114, 205, 208, 220,	157, 187, 204, 204, 212, 217, 354, 371
395, 398, 410, 678, 681, 896, 899, 913	\__tag_mc_disable_marks: . . . . 74, 74
__tag_get_tag_from . . . . . 304	\__tag_mc_emc: . . . . 154, 229, 231, 443
\l__tag_get_tmpe_tl . . . . . 109,	\__tag_mc_end_marks: . 19, 59, 78, 444
166, 171, 182, 184, 185, 238, 240,	\l__tag_mc_firstmarks_seq . . . . .
241, 868, 874, 1157, 1159, 1163, 1169	. . . . . 82, 17, 83, 106, 186, 192,
\__tag_gincr_para_begin_int: . . .	195, 196, 203, 203, 204, 354, 362, 364
. . . . 311, 315, 333, 349, 362, 453, 480	\g__tag_mc_footnote_marks_seq . . . 14
\__tag_gincr_para_end_int: . . . . .	\__tag_mc_get_marks: 80, 80, 178, 199
. . . . . 311, 323, 341, 351, 496	\__tag_mc_handle_artifact:N . . . . .
\__tag_gincr_para_main_begin_-	. . . . . 115, 333, 341, 375
int: . . . 311, 311, 329, 348, 446, 471	\__tag_mc_handle_mc_label:n . . . . .
\__tag_gincr_para_main_end_int: . .	. . . . . 26, 26, 197, 388
. . . . . 311, 319, 337, 350, 503	\__tag_mc_handle_mcid:nn . . . . .
\__tag_hook_kernel_after_foot: . .	. . . . . 234, 316, 321, 381
. . . . . 604, 613, 682, 689, 696	\__tag_mc_handle_stash:n 49, 138,
\__tag_hook_kernel_after_head: . .	140, 141, 170, 227, 322, 322, 332, 416
. . . . . 602, 611, 681, 688, 695	\__tag_mc_if_in: . . . . 66, 80, 221, 228
\__tag_hook_kernel_before_foot: . .	\__tag_mc_if_in:TF 66, 86, 221, 246, 254
. . . . . 603, 612, 680, 687, 694	\__tag_mc_if_in:p: . . . . . 66, 221
\__tag_hook_kernel_before_head: . .	\__tag_mc_insert_extra_tmb:n . . . . .
. . . . . 601, 610, 679, 686, 693	. . . . . 104, 104, 167
\g__tag_in_mc_bool . . . . .	\__tag_mc_insert_extra_tme:n . . . . .
. . . . . 16, 18, 177, 223, 238,	. . . . . 104, 149, 168
369, 441, 626, 627, 641, 653, 654, 671	\__tag_mc_insert_mcid_kids:n . . . . .
__tag_insert_bdc_node . . . . . 400	. . . . . 129, 129, 148, 269
__tag_insert_bmc_node . . . . . 371	\__tag_mc_insert_mcid_single_-
__tag_insert_emc_node . . . . . 342	kids:n . . . . . 129, 134, 270
\__tag_lastpagelabel: . . . 89, 90, 108	\l__tag_mc_key_label_tl . . . . .
__tag_log . . . . . 188	. 22, 194, 197, 319, 384, 385, 388, 489
\l__tag_loglevel_int 125, 132, 169,	\l__tag_mc_key_properties_tl . . . . .
172, 185, 215, 234, 262, 265, 273,	. . . . . 22, 179, 268, 283, 284,
276, 279, 280, 281, 289, 382, 389,	304, 305, 383, 463, 472, 473, 485, 486
396, 403, 423, 431, 438, 446, 453, 482	\l__tag_mc_key_stash_bool . . . . .
__tag_mark_spaces . . . . . 460	. . . . . 20, 27, 36, 123, 200, 390
\__tag_mc_artifact_begin_marks:n	\g__tag_mc_key_tag_tl . . . 19, 22,
. . . . . 19, 41, 77, 377	182, 242, 254, 260, 349, 371, 442, 459
\l__tag_mc_artifact_bool . . . . .	\l__tag_mc_key_tag_tl 22, 181, 189,
. . . . . 20, 124, 178, 192, 239, 373	191, 241, 259, 370, 380, 382, 384, 458
\l__tag_mc_artifact_type_tl . . . . .	\__tag_mc_lua_set_mc_type_attr:n
. . . . . 19, 128, 132, 136,	. . . . . 81, 81, 105, 191
140, 144, 148, 152, 156, 347, 375, 377	\__tag_mc_lua_unset_mc_type_-
\__tag_mc_bdc:nn 229, 232, 264, 306, 339	attr: . . . . . 81, 107, 240
\__tag_mc_bdc_mcid:n . . 119, 234, 311	\g__tag_mc_main_marks_seq . . . . . 14
\__tag_mc_bdc_mcid:nn . . . . .	\g__tag_mc_marks . . . . . 13,
. . . . . 234, 237, 267, 313, 318	21, 30, 43, 50, 61, 67, 84, 87, 193, 213
\__tag_mc_bdc_shipout:nn . . . 233, 245	\g__tag_mc_multicol_marks_seq . . . 14

\g\_\_tag\_mc\_parenttree\_prop .....  
     ..... [17](#), [18](#), [99](#), [164](#), [185](#), [328](#)  
 \l\_\_tag\_mc\_ref\_abspage\_tl .....  
     ..... [11](#), [270](#), [282](#), [290](#), [298](#)  
 \\_\_tag\_mc\_set\_label\_used:n [30](#), [30](#), [50](#)  
 \g\_\_tag\_mc\_stack\_seq .....  
     ..... [18](#), [88](#), [95](#), [105](#), [268](#)  
 \\_\_tag\_mc\_store:nnn . [89](#), [89](#), [103](#), [130](#)  
 \l\_\_tag\_mc\_tmpa\_tl .. [12](#), [284](#), [287](#), [291](#)  
 g\_\_tag\_MCID\_abs\_int ..... [7](#)  
 \g\_\_tag\_MCID\_byabspage\_prop .....  
     ..... [262](#), [280](#), [289](#), [297](#)  
 \g\_\_tag\_MCID\_tmp\_bypage\_int .....  
     ..... [263](#), [266](#), [287](#), [295](#), [308](#)  
 \g\_\_tag\_mode\_lua\_bool .....  
     ... [41](#), [49](#), [50](#), [114](#), [241](#), [277](#), [303](#),  
     [308](#), [317](#), [369](#), [561](#), [621](#), [636](#), [648](#), [666](#)  
 \\_\_tag\_new\_output\_prop\_handler:n  
     ..... [68](#), [78](#), [102](#), [809](#)  
 \_\_tag\_pairs\_prop ..... [205](#)  
 \l\_\_tag\_para\_attr\_class\_tl .....  
     ..... [292](#), [383](#), [485](#)  
 \g\_\_tag\_para\_begin\_int .....  
     ..... [292](#), [317](#), [335](#), [422](#), [543](#), [548](#)  
 \l\_\_tag\_para\_bool [292](#), [389](#), [398](#), [405](#),  
     [411](#), [442](#), [461](#), [494](#), [593](#), [594](#), [620](#), [647](#)  
 \g\_\_tag\_para\_end\_int .....  
     ..... [292](#), [325](#), [343](#), [433](#), [543](#), [549](#)  
 \l\_\_tag\_para\_flattened\_bool .....  
     ..... [292](#), [394](#), [401](#), [414](#), [444](#), [469](#), [501](#)  
 \l\_\_tag\_para\_main\_attr\_class\_tl .  
     ..... [292](#), [475](#)  
 \g\_\_tag\_para\_main\_begin\_int .....  
     ..... [292](#), [313](#), [331](#), [534](#), [539](#)  
 \g\_\_tag\_para\_main\_end\_int .....  
     ..... [292](#), [321](#), [339](#), [534](#), [540](#)  
 \\_\_tag\_para\_main\_store\_struct: . .  
     ..... [353](#), [353](#), [451](#), [477](#)  
 \g\_\_tag\_para\_main\_struct\_tl [292](#), [355](#)  
 \l\_\_tag\_para\_main\_tag\_tl .....  
     ..... [292](#), [393](#), [400](#), [413](#), [449](#), [474](#)  
 \l\_\_tag\_para\_show\_bool .....  
     ..... [292](#), [390](#), [391](#), [406](#), [419](#), [430](#)  
 \l\_\_tag\_para\_tag\_default\_tl ... [292](#)  
 \l\_\_tag\_para\_tag\_tl .....  
     ..... [292](#), [361](#), [392](#), [399](#), [407](#), [412](#), [454](#), [484](#)  
 \l\_\_tag\_parent\_child\_check\_tl ...  
     ..... [210](#), [211](#), [400](#), [401](#), [459](#), [683](#),  
     [684](#), [902](#), [903](#), [1048](#), [1049](#), [1111](#), [1112](#)  
 \\_\_tag\_parenttree\_add\_objr:mn ...  
     ..... [162](#), [162](#), [442](#)  
 \l\_\_tag\_parenttree\_content\_tl ...  
     ..... [169](#), [188](#), [200](#), [220](#), [228](#), [249](#), [252](#)  
 \g\_\_tag\_parenttree\_objr\_tl .....  
     ..... [161](#), [164](#), [249](#)  
 \\_\_tag\_pdf\_name\_e:n ..... [98](#), [98](#)  
 \_\_tag\_pdf\_object\_ref ..... [430](#)  
 \\_\_tag\_prop\_gput:Nnn .....  
     ..... [9](#), [29](#), [90](#), [120](#), [127](#),  
     [131](#), [180](#), [183](#), [190](#), [288](#), [296](#), [307](#), [1026](#)  
 \\_\_tag\_prop\_item:Nn .. [9](#), [49](#), [180](#), [186](#)  
 \\_\_tag\_prop\_new:N ..... [9](#),  
     [9](#), [11](#), [101](#), [180](#), [180](#), [192](#), [262](#), [807](#)  
 \\_\_tag\_prop\_new\_linked:N .....  
     ..... [15](#), [17](#), [180](#), [181](#)  
 \\_\_tag\_prop\_show:N [9](#), [62](#), [180](#), [188](#), [195](#)  
 \c\_\_tag\_property\_mc\_clist .....  
     ..... [123](#), [244](#), [305](#)  
 \\_\_tag\_property\_record:nn .....  
     . [28](#), [149](#), [149](#), [158](#), [240](#), [301](#), [429](#), [505](#)  
 \\_\_tag\_property\_ref\_lastpage:nn .  
     . [82](#), [159](#), [159](#), [159](#), [173](#), [176](#), [310](#), [324](#)  
 \c\_\_tag\_property\_struct\_clist ...  
     ..... [123](#), [507](#)  
 \l\_\_tag\_Ref\_tmpa\_tl ..... [109](#)  
 g\_\_tag\_role/RoleMap\_dict ..... [18](#)  
 \g\_\_tag\_role\_add\_mathml\_bool ...  
     ..... [73](#), [258](#), [702](#), [752](#)  
 \\_\_tag\_role\_add\_tag:nn .....  
     ..... [127](#), [127](#), [155](#), [282](#), [367](#), [737](#)  
 \\_\_tag\_role\_add\_tag:nnnn .....  
     ..... [169](#), [169](#), [228](#), [314](#), [742](#)  
 \\_\_tag\_role\_alloctag:nnn .... [81](#),  
     [85](#), [95](#), [107](#), [117](#), [126](#), [142](#), [186](#), [279](#), [310](#)  
 \l\_\_tag\_role\_debug\_prop .....  
     ..... [160](#), [11](#), [518](#), [519](#), [591](#), [592](#)  
 \\_\_tag\_role\_get:nnNN .....  
     ..... [156](#), [158](#), [166](#), [229](#), [231](#), [255](#), [493](#), [847](#)  
 \\_\_tag\_role\_get\_parent\_child\_  
     rule:nnnN [174](#), [459](#), [460](#), [512](#), [544](#), [651](#)  
 \g\_\_tag\_role\_index\_prop . [160](#), [10](#),  
     [416](#), [424](#), [436](#), [437](#), [438](#), [443](#), [449](#),  
     [451](#), [452](#), [455](#), [457](#), [464](#), [465](#), [520](#), [530](#)  
 \g\_\_tag\_role\_NS\_<ns>\_class\_prop [160](#)  
 \g\_\_tag\_role\_NS\_<ns>\_prop ..... [160](#)  
 \g\_\_tag\_role\_NS\_mathml\_prop [260](#), [453](#)  
 \\_\_tag\_role\_NS\_new:nnn .....  
     . [162](#), [20](#), [22](#), [30](#), [74](#), [75](#), [76](#), [77](#), [78](#), [80](#)  
 \g\_\_tag\_role\_NS\_prop .....  
     ... [160](#), [9](#), [26](#), [56](#), [166](#), [318](#), [336](#), [725](#)  
 \g\_\_tag\_role\_parent\_child\_  
     intarray ..... [392](#), [395](#), [473](#)  
 \\_\_tag\_role\_read\_namespace:n [339](#),  
     [339](#), [343](#), [344](#), [345](#), [347](#), [349](#), [351](#), [352](#)  
 \\_\_tag\_role\_read\_namespace:nn ...  
     ..... [320](#), [320](#), [341](#), [350](#)

\__tag_role_read_namespace_-	\__tag_store_parent_child_-
line:nw . . . . . <a href="#">257</a> , <a href="#">261</a> , <a href="#">294</a> , <a href="#">330</a>	rule:nnn . . . . . <a href="#">393</a> , <a href="#">393</a> , <a href="#">430</a>
\__tag_role_remap: . . . . .	g__tag_struct_1_prop . . . . . <a href="#">100</a>
. . . . . <a href="#">690</a> , <a href="#">690</a> , <a href="#">691</a> , <a href="#">921</a> , <a href="#">1053</a> , <a href="#">1116</a>	\__tag_struct_add_AF:nn . . . . .
\__tag_role_remap_id: . . . . . <a href="#">691</a> , <a href="#">691</a>	. . . . . <a href="#">672</a> , <a href="#">689</a> , <a href="#">708</a> , <a href="#">715</a> , <a href="#">734</a> , <a href="#">777</a>
\l__tag_role_remap_NS_tl . . . . .	\__tag_struct_add_inline_AF:nn . . . . .
<a href="#">688</a> , <a href="#">920</a> , <a href="#">923</a> , <a href="#">1052</a> , <a href="#">1055</a> , <a href="#">1115</a> , <a href="#">1118</a>	. . . . . <a href="#">661</a> , <a href="#">688</a> , <a href="#">748</a> , <a href="#">752</a> , <a href="#">759</a> , <a href="#">767</a>
\l__tag_role_remap_tag_tl . . . . .	\g__tag_struct_AFobj_int <a href="#">659</a> , <a href="#">667</a> , <a href="#">670</a>
<a href="#">688</a> , <a href="#">919</a> , <a href="#">922</a> , <a href="#">1051</a> , <a href="#">1054</a> , <a href="#">1114</a> , <a href="#">1117</a>	\g__tag_struct_cont_mc_prop . . . . .
\l__tag_role_role_namespace_-	. . . . . <a href="#">11</a> , <a href="#">91</a> , <a href="#">92</a> , <a href="#">94</a> , <a href="#">97</a> , <a href="#">221</a>
tmpa_tl . . . . . <a href="#">12</a> ,	\g__tag_struct_dest_num_prop <a href="#">64</a> , <a href="#">617</a>
<a href="#">697</a> , <a href="#">718</a> , <a href="#">723</a> , <a href="#">725</a> , <a href="#">727</a> , <a href="#">731</a> , <a href="#">746</a>	\l__tag_struct_elem_stash_bool . . . . .
\l__tag_role_role_tmpa_tl . . . . .	. . . . . <a href="#">63</a> , <a href="#">509</a> , <a href="#">891</a> , <a href="#">951</a>
. . . . . <a href="#">12</a> , <a href="#">696</a> , <a href="#">716</a> , <a href="#">722</a> , <a href="#">739</a> , <a href="#">745</a>	\__tag_struct_exchange_kid_-
\g__tag_role_rolemap_prop . . . . .	command:N . . . . . <a href="#">265</a> , <a href="#">265</a> , <a href="#">274</a> , <a href="#">305</a>
. . . . . <a href="#">160</a> , <a href="#">18</a> , <a href="#">145</a> , <a href="#">148</a> , <a href="#">151</a> , <a href="#">160</a> ,	\__tag_struct_fill_kid_key:n . . . . .
<a href="#">216</a> , <a href="#">219</a> , <a href="#">222</a> , <a href="#">262</a> , <a href="#">265</a> , <a href="#">387</a> , <a href="#">525</a> , <a href="#">535</a>	. . . . . <a href="#">135</a> , <a href="#">275</a> , <a href="#">275</a> , <a href="#">400</a>
\c__tag_role_rules_num_prop <a href="#">393</a> , <a href="#">484</a>	\__tag_struct_format_parentrole:nnN
\c__tag_role_rules_prop <a href="#">393</a> , <a href="#">396</a> , <a href="#">477</a>	. . . . . <a href="#">369</a> , <a href="#">370</a>
\l__tag_role_tag_namespace_tmpa_-	\__tag_struct_format_Ref . . . . . <a href="#">119</a>
tl . . . . . <a href="#">12</a> , <a href="#">567</a> , <a href="#">571</a> , <a href="#">575</a> , <a href="#">695</a> , <a href="#">744</a>	\__tag_struct_format_Ref:nnN <a href="#">371</a> , <a href="#">371</a>
\l__tag_role_tag_namespace_tmpb_-	\__tag_struct_format_rolemap:nnN
tl . . . . . <a href="#">14</a> , <a href="#">568</a> , <a href="#">569</a> , <a href="#">572</a> , <a href="#">576</a>	. . . . . <a href="#">369</a> , <a href="#">369</a>
\l__tag_role_tag_namespace_tmpb_-	\__tag_struct_get_dict_content:nN
tluuuuuu% . . . . . <a href="#">12</a>	. . . . . <a href="#">137</a> , <a href="#">355</a> , <a href="#">355</a> , <a href="#">401</a>
\l__tag_role_tag_tmpa_tl . . . . .	\__tag_struct_get_id:n . . . . .
. . . . . <a href="#">12</a> , <a href="#">694</a> , <a href="#">715</a> , <a href="#">738</a> , <a href="#">743</a>	. . . . . <a href="#">95</a> , <a href="#">100</a> , <a href="#">113</a> , <a href="#">114</a> , <a href="#">137</a> , <a href="#">138</a> , <a href="#">407</a> , <a href="#">463</a>
\g__tag_role_tags_class_prop . . . . .	\__tag_struct_get_parentrole:nnN
. . . . . <a href="#">160</a> , <a href="#">8</a> , <a href="#">90</a> , <a href="#">99</a> , <a href="#">112</a> , <a href="#">121</a> , <a href="#">137</a> , <a href="#">270</a>	. . . . . <a href="#">176</a> ,
\g__tag_role_tags_NS_prop . . . . . <a href="#">160</a> ,	<a href="#">176</a> , <a href="#">192</a> , <a href="#">202</a> , <a href="#">392</a> , <a href="#">675</a> , <a href="#">893</a> , <a href="#">1039</a> , <a href="#">1102</a>
<a href="#">7</a> , <a href="#">88</a> , <a href="#">97</a> , <a href="#">110</a> , <a href="#">119</a> , <a href="#">130</a> , <a href="#">178</a> , <a href="#">213</a> ,	\__tag_struct_gput_data_ref:nn . . . . .
<a href="#">277</a> , <a href="#">385</a> , <a href="#">482</a> , <a href="#">490</a> , <a href="#">567</a> , <a href="#">568</a> , <a href="#">721</a> , <a href="#">995</a>	. . . . . <a href="#">1171</a> , <a href="#">1188</a>
\l__tag_role_tmpa_seq . . . . . <a href="#">12</a>	\__tag_struct_gput_data_ref_-
\l__tag_role_update_bool . . . . .	aux:nnn . . . . .
. . . . . <a href="#">210</a> , <a href="#">257</a> , <a href="#">258</a> , <a href="#">266</a> , <a href="#">346</a> , <a href="#">348</a>	. . . . . <a href="#">1150</a> , <a href="#">1151</a> , <a href="#">1173</a> , <a href="#">1177</a> , <a href="#">1181</a> , <a href="#">1185</a>
\c__tag_role_userNS_id_str . . . . .	\__tag_struct_gput_data_ref_-
. . . . . <a href="#">161</a> , <a href="#">59</a> , <a href="#">80</a>	dest:nn . . . . . <a href="#">1179</a>
\g__tag_root_default_tl . . . . . <a href="#">255</a>	\__tag_struct_gput_data_ref_-
\g__tag_saved_in_mc_bool . . . . .	label:nn . . . . . <a href="#">1175</a>
. . . . . <a href="#">617</a> , <a href="#">626</a> , <a href="#">641</a> , <a href="#">653</a> , <a href="#">671</a>	\__tag_struct_gput_data_ref_-
\__tag_seq_gput_right:Nn . . . . . <a href="#">9</a> ,	num:nn . . . . . <a href="#">1183</a>
<a href="#">36</a> , <a href="#">180</a> , <a href="#">184</a> , <a href="#">191</a> , <a href="#">208</a> , <a href="#">218</a> , <a href="#">228</a> , <a href="#">251</a>	\__tag_struct_insert_annot:nn . . . . .
\__tag_seq_item:Nn . . . . . <a href="#">9</a> , <a href="#">44</a> , <a href="#">180</a> , <a href="#">185</a>	. . . . . <a href="#">415</a> , <a href="#">415</a> , <a href="#">1194</a>
\__tag_seq_new:N . . . . .	\__tag_struct_kid_mc_gput_-
. . . . . <a href="#">9</a> , <a href="#">9</a> , <a href="#">22</a> , <a href="#">103</a> , <a href="#">180</a> , <a href="#">182</a> , <a href="#">193</a> , <a href="#">810</a>	right:nn . . . . . <a href="#">193</a> , <a href="#">205</a> , <a href="#">206</a> , <a href="#">224</a> , <a href="#">325</a>
\__tag_seq_show:N . . . . . <a href="#">9</a> , <a href="#">55</a> , <a href="#">180</a> , <a href="#">187</a> , <a href="#">194</a>	\__tag_struct_kid_OBJR_gput_-
__tag_show_spacemark . . . . . <a href="#">441</a>	right:nnn . . . . . <a href="#">241</a> , <a href="#">241</a> , <a href="#">244</a> , <a href="#">264</a> , <a href="#">430</a>
\l__tag_showspaces_bool . . . . . <a href="#">7</a> , <a href="#">16</a> , <a href="#">17</a>	\__tag_struct_kid_struct_gput_-
\g__tag_softyphen_bool . . . . . <a href="#">138</a> , <a href="#">288</a>	right:nn . . . . .
__tag_space_chars_shipout . . . . . <a href="#">553</a>	. . . . . <a href="#">225</a> , <a href="#">225</a> , <a href="#">226</a> , <a href="#">240</a> , <a href="#">938</a> , <a href="#">1022</a> , <a href="#">1085</a>
\__tag_start_para_ints: . . . . .	g__tag_struct_kids_1_seq . . . . . <a href="#">100</a>
. . . . . <a href="#">216</a> , <a href="#">240</a> , <a href="#">327</a> , <a href="#">327</a>	\g__tag_struct_label_num_prop . . . . .
\__tag_stop_para_ints: . . . . .	. . . . . <a href="#">62</a> , <a href="#">503</a> , <a href="#">604</a>
. . . . . <a href="#">206</a> , <a href="#">230</a> , <a href="#">327</a> , <a href="#">346</a>	

\l__tag_struct_lang_tl . . . . .	993, 995, 1037, 1044, 1051, 1054, 1058, 1100, 1107, 1114, 1117, 1121
. . . . . 599, 793, 818, 823	
\__tag_struct_mcid_dict:n . . . . .	\__tag_struct_write_obj . . . . . 119
. . . . . 94, 97, 193, 211	\__tag_struct_write_obj:n . . . . .
\c__tag_struct_null_tl . . . . . 10, 309	. . . . . 150, 383, 383
\g__tag_struct_objR_seq . . . . . 8	\l__tag_tag_stop_int 197, 201, 202, 210, 211, 218, 225, 226, 234, 235, 243
\__tag_struct_output_prop_aux:nn	\g__tag_tagunmarked_bool 137, 285, 287
. . . . . 68, 68, 82	\l__tag_tmpa_box . . . . .
\__tag_struct_prop_gput:nnn . . . . .	. . . . . 109, 167, 173, 174, 178, 189, 190
86, 87, 88, 94, 105, 110, 115, 120, 127, 153, 162, 168, 311, 324, 338, 538, 550, 564, 580, 588, 653, 675, 716, 735, 778, 814, 820, 825, 852, 870, 879, 929, 1089, 1166, 1257, 1308	\l__tag_tmpa_clist . . . . .
\g__tag_struct_ref_by_dest_prop .	. . . . . 109, 1235, 1236, 1269, 1270, 1272
. . . . . 67, 82	\l__tag_tmpa_int . . . . . 89, 92, 97, 100, 104, 109, 113, 398, 410, 412, 482
\__tag_struct_Ref_dest:nN . . 594, 615	\l__tag_tmpa_prop . . . . .
\__tag_struct_Ref_label:nN . 594, 602	. . . . . 109, 175, 183, 196, 198
\__tag_struct_Ref_num:nN . . . 594, 628	\l__tag_tmpa_seq . . . . .
\__tag_struct_Ref_obj:nN . . . 594, 594	. . . 50, 57, 58, 109, 279, 281, 283, 284, 285, 286, 411, 414, 422, 423, 425, 426, 427, 482, 483, 484, 490, 491, 492, 1237, 1241, 1251, 1252, 1253, 1255, 1273, 1279, 1281, 1305
\g__tag_struct_roletag_NS_tl . . . 58	\l__tag_tmpa_str . . . . . 42, 43, 48, 115, 279, 284, 289, 300, 305, 312, 468, 473, 481, 486, 534, 541, 546, 553, 560, 567, 576, 583, 649, 656
\l__tag_struct_roletag_NS_tl . . .	\l__tag_tmpa_tl . . . . .
. . . . . 61, 851, 856, 883	. . . 41, 42, 46, 48, 49, 50, 55, 84, 87, 91, 92, 93, 94, 101, 105, 105, 107, 108, 109, 112, 113, 114, 115, 115, 137, 137, 138, 140, 142, 142, 145, 146, 151, 179, 180, 182, 185, 186, 198, 199, 200, 201, 201, 202, 204, 207, 216, 216, 217, 222, 224, 267, 268, 270, 271, 271, 273, 277, 279, 281, 288, 300, 301, 302, 304, 306, 307, 308, 308, 309, 310, 316, 401, 406, 406, 413, 423, 424, 425, 426, 436, 437, 438, 443, 449, 451, 455, 455, 459, 463, 464, 467, 474, 484, 486, 493, 494, 495, 520, 522, 525, 527, 541, 545, 595, 603, 605, 606, 608, 612, 617, 648, 652, 671, 674, 674, 676, 908, 909, 916, 977, 978, 984, 986, 991, 994, 995, 997, 1041, 1046, 1080, 1104, 1109, 1249, 1260
\l__tag_struct_roletag_tl . . . . .	\l__tag_tmpb_box . . . . .
. . . . . 58, 850, 856, 858, 883, 887	. . . . . 109, 168, 175, 176, 180, 182
\__tag_struct_set_tag_info:nnn . .	\l__tag_tmpb_seq . . . . .
148, 150, 160, 175, 832, 924, 1056, 1119	. . . . . 109, 1236, 1237, 1272, 1273
\g__tag_struct_stack_current_tl .	\l__tag_tmpb_tl . . . . .
. . . . . 16, 25, 34, 65, 71, 97, 146, 152, 160, 166, 203, 214, 224, 280, 326, 330, 393, 404, 413, 458, 463, 469, 888, 936, 940, 941, 962, 980, 986, 1023, 1030, 1036, 1086, 1093, 1099	. . . . . 172, 88, 103, 109, 117, 119, 387, 424, 430, 452, 457, 465, 468, 474, 488, 493, 495, 530, 532,
\l__tag_struct_stack_parent_-	
tmpa_tl . . . . . 16, 423, 432, 447, 519, 830, 837, 841, 866, 894, 906, 916, 933, 937, 939, 942, 954, 955, 963	
\g__tag_struct_stack_seq 12, 22, 25, 422, 674, 840, 846, 889, 973, 978, 984	
\c__tag_struct_StructElem_-	
entries_seq . . . . . 21	
\c__tag_struct_StructTreeRoot_-	
entries_seq . . . . . 21	
\g__tag_struct_tag_NS_tl 58, 484, 492, 493, 495, 835, 849, 901, 914, 920, 923, 927, 961, 997, 1045, 1052, 1055, 1059, 1108, 1115, 1118, 1122	
\g__tag_struct_tag_stack_seq . . .	
. . . . . 14, 50, 219, 220, 426, 441, 455, 886, 977, 991	
\g__tag_struct_tag_tl . . . . .	
. . . . . 58, 181, 182, 185, 370, 371, 483, 485, 491, 493, 494, 496, 834, 848, 887, 900, 914, 919, 922, 926,	

535, 537, 542, 545, 604, 608, 617, 621, 622, 630, 632, 633, 635, 639, 644, 649, 652, 1042, 1047, 1105, 1110	
<code>\__tag_tree_fill_parenttree:</code> . . .	
. . . . .	<a href="#">170, 171, 246</a>
<code>\__tag_tree_final_checks:</code> <a href="#">20, 20, 346</a>	
<code>\g__tag_tree_id_pad_int</code> . . . <a href="#">77, 81, 143</a>	
<code>\__tag_tree_lua_fill_parenttree:</code> . . . . .	<a href="#">226, 226, 243</a>
<code>\g__tag_tree_openaction_struct_- t1</code> . . . . .	<a href="#">31, 37, 56</a>
<code>\__tag_tree_parenttree_rerun_- msg:</code> . . . . .	<a href="#">170, 213, 248</a>
<code>\__tag_tree_update_openaction:</code> . . . . . . .	<a href="#">41, 74</a>
<code>\__tag_tree_write_classmap:</code> . . . . . . . . .	<a href="#">279, 279, 361</a>
<code>\__tag_tree_write_idtree:</code> . . . <a href="#">85, 353</a>	
<code>\__tag_tree_write_namespaces:</code> . . . . . . . .	<a href="#">314, 314, 365</a>
<code>\__tag_tree_write_parenttree:</code> . . . . . . . .	<a href="#">239, 239, 349</a>
<code>\__tag_tree_write_rolemap:</code> . . . . . . . . . .	<a href="#">256, 256, 357</a>
<code>\__tag_tree_write_structelements:</code> . . . . .	<a href="#">146, 146, 369</a>
<code>\__tag_tree_write_structtreeroot:</code> . . . . .	<a href="#">125, 125, 373</a>
<code>\__tag_whatsits:</code> . . . . . . . . . .	<a href="#">35, 57, 62, 63, 66, 351, 352</a>
<code>tag-namespace<sub>␣</sub>(rolemap-key)</code> . . . . .	<a href="#">692</a>
tag/struct/1 internal commands:	
<code>__tag/struct/1</code> . . . . .	<a href="#">30</a>
tag/tree/namespaces internal commands:	
<code>__tag/tree/namespaces</code> . . . . .	<a href="#">313</a>
tag/tree/parenttree internal commands:	
<code>__tag/tree/parenttree</code> . . . . .	<a href="#">153</a>
tag/tree/rolemap internal commands:	
<code>__tag/tree/rolemap</code> . . . . .	<a href="#">255</a>
<code>tagabspage</code> . . . . .	<a href="#">7, 163</a>
<code>tagmcabs</code> . . . . .	<a href="#">7, 163</a>
<code>\tagmcbegin</code> . . . . .	<a href="#">36, 160, 22, 370, 376</a>
<code>\tagmccend</code> . . . . .	<a href="#">36, 22, 376</a>
<code>tagmcid</code> . . . . .	<a href="#">7, 163</a>
<code>\tagmccifin</code> . . . . .	<a href="#">36</a>
<code>\tagmccifinTF</code> . . . . .	<a href="#">36, 39</a>
<code>\tagmccuse</code> . . . . .	<a href="#">36, 22</a>
<code>\tagpdfparaOff</code> . . . . .	<a href="#">38, 590</a>
<code>\tagpdfparaOn</code> . . . . .	<a href="#">38, 590</a>
<code>\tagpdfsetup</code> . . . . .	<a href="#">36, 103, 159, 6</a>
<code>\tagpdfsuppressmarks</code> . . . . .	<a href="#">38, 595</a>
<code>\tagstart</code> . . . . .	<a href="#">6, 221, 250</a>
<code>\tagstop</code> . . . . .	<a href="#">6, 220, 249</a>
<code>tagstruct</code> . . . . .	<a href="#">7, 163</a>
<code>\tagstructbegin</code> . . . . .	
. . . . .	<a href="#">37, 131, 159, 160, 45, 258, 361, 363</a>
<code>\tagstructend</code> . . . . .	<a href="#">37, 45, 259, 376</a>
<code>tagstructobj</code> . . . . .	<a href="#">7, 163</a>
<code>\tagstructuse</code> . . . . .	<a href="#">37, 45</a>
<code>\tagtool</code> . . . . .	<a href="#">36, 13</a>
<code>tagunmarked<sub>␣</sub>(deprecated)</code> . . . . .	<a href="#">6, 285</a>
<code>test/lang<sub>␣</sub>(setup-key)</code> . . . . .	<a href="#">597</a>
TeX and L <sup>A</sup> T <sub>E</sub> X 2 <sub>ε</sub> commands:	
<code>\M</code> . . . . .	<a href="#">164</a>
<code>\@auxout</code> . . . . .	<a href="#">94</a>
<code>\@bsphack</code> . . . . .	<a href="#">151</a>
<code>\@cclv</code> . . . . .	<a href="#">570</a>
<code>\@esphack</code> . . . . .	<a href="#">153</a>
<code>\@gobble</code> . . . . .	<a href="#">31, 55</a>
<code>\@ifpackageloaded</code> . . . . .	<a href="#">28, 553</a>
<code>\@kernel@after@foot</code> . . . . .	<a href="#">613</a>
<code>\@kernel@after@head</code> . . . . .	<a href="#">611</a>
<code>\@kernel@before@cclv</code> . . . . .	<a href="#">560, 567</a>
<code>\@kernel@before@foot</code> . . . . .	<a href="#">612</a>
<code>\@kernel@before@footins</code> . . . . .	<a href="#">563, 565</a>
<code>\@kernel@before@head</code> . . . . .	<a href="#">608, 610</a>
<code>\@kernel@tag@hangfrom</code> . . . . .	<a href="#">359</a>
<code>\@kernel@tagsupport@makecol</code> <a href="#">559, 572</a>	
<code>\@makecol</code> . . . . .	<a href="#">569, 574</a>
<code>\@maxdepth</code> . . . . .	<a href="#">177</a>
<code>\@mult@ptagging@hook</code> . . . . .	<a href="#">577</a>
<code>\@outputbox</code> . . . . .	<a href="#">575</a>
<code>\@secondoftwo</code> . . . . .	<a href="#">31, 55</a>
<code>\@tempboxa</code> . . . . .	<a href="#">364, 374, 376</a>
<code>\c@chapter</code> . . . . .	<a href="#">362, 380</a>
<code>\c@page</code> . . . . .	<a href="#">569, 574</a>
<code>\count@</code> . . . . .	<a href="#">582</a>
<code>\mult@firstbox</code> . . . . .	<a href="#">580</a>
<code>\mult@rightbox</code> . . . . .	<a href="#">584</a>
<code>\new@label@record</code> . . . . .	<a href="#">96</a>
<code>\on@line</code> . . . . .	<a href="#">466, 481, 497</a>
<code>\page@sofar</code> . . . . .	<a href="#">579</a>
<code>\process@cols</code> . . . . .	<a href="#">580</a>
tex commands:	
<code>\tex_botmarks:D</code> . . . . .	<a href="#">87</a>
<code>\tex_firstmarks:D</code> . . . . .	<a href="#">84</a>
<code>\tex_kern:D</code> . . . . .	<a href="#">180</a>
<code>\tex_marks:D</code> . . . . .	<a href="#">21, 30, 43, 50, 61, 67</a>
<code>\tex_special:D</code> . . . . .	<a href="#">66</a>
<code>\tex_splitbotmarks:D</code> . . . . .	<a href="#">213</a>
<code>\tex_splitfirstmarks:D</code> . . . . .	<a href="#">193</a>
<code>texsource</code> . . . . .	<a href="#">102</a>
<code>\the</code> . . . . .	<a href="#">569, 574</a>
<code>\tiny</code> . . . . .	<a href="#">422, 433</a>
<code>title<sub>␣</sub>(struct-key)</code> . . . . .	<a href="#">101, 499</a>
<code>title-o<sub>␣</sub>(struct-key)</code> . . . . .	<a href="#">101, 499</a>
tl commands:	
<code>\c_empty_tl</code> . . . . .	<a href="#">367, 387</a>

