

tagpdf – A package to experiment with pdf tagging*

Ulrike Fischer[†]

Released 2022-01-09

Contents

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

*This file describes v0.93, last revised 2022-01-09.

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

2	Description of log messages	12
2.1	\ShowTagging command	12
2.2	Messages in checks and commands	12
2.3	Messages from the ptagging code	13
2.4	Warning messages from the lua-code	13
2.5	Info messages from the lua-code	13
2.6	Debug mode messages and code	14
3	Messages	14
3.1	Messages related to mc-chunks	14
3.2	Messages related to mc-chunks	15
3.3	Attributes	16
3.4	Roles	16
3.5	Miscellaneous	16
4	Retrieving data	17
5	User conditionals	17
6	Internal checks	17
6.1	checks for active tagging	18
6.2	Checks related to stuctures	18
6.3	Checks related to roles	19
6.4	Check related to mc-chunks	20
6.5	Checks related to the state of MC on a page or in a split stream	22
	Index	26

<code>\ref_value:nnn</code>	<code>\ref_value:nnn{<label>}{<attribute>}{<fallback default>}</code>
-----------------------------	---

This is a temporary definition which will have to move to l3ref. It allows to locally set a default value if the label or the attribute doesn't exist. See issue #4 in Accessible-xref.

<code>\tag_stop_group_begin:</code>	We need a command to stop tagging in some places. This simply switches the two local booleans.
<code>\tag_stop_group_end:</code>	

<code>activate-space_<setup-key></code>	<code>activate-space</code> activates the additional parsing needed for interword spaces. It is not documented, the parsing is currently implicitly activated by the known key <code>interwordspace</code> , as the code will perhaps move to some other place, now that it is better separated.
---	--

<code>activate-mc_<setup-key></code>
<code>activate-tree_<setup-key></code>
<code>activate-struct_<setup-key></code>
<code>activate-all_<setup-key></code>

Keys to activate the various tagging steps

<code>no-struct-dest_<setup-key></code>	The key allows to suppress the creation of structure destinations
---	---

<code>log_<setup-key></code>	The <code>log</code> takes currently the values <code>none</code> , <code>v</code> , <code>vv</code> , <code>vvv</code> , <code>all</code> . More details are in <code>tagpdf-checks</code> .
------------------------------------	---

<code>tagunmarked_<setup-key></code>	This key allows to set if (in <code>luamode</code>) unmarked text should be marked up as artifact. The initial value is <code>true</code> .
--	--

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

<code>tagstruct</code>	These are attributes used by the label/ref system.
<code>tagstructobj</code>	
<code>tagabspace</code>	
<code>tagmcabs</code>	
<code>tagmcid</code>	

1 Initialization and test if pdfmanagement is active.

```

1 <@@=tag>
2 <*package>
3 \ProvidesExplPackage {tagpdf} {2022-01-09} {0.93}
4 { A package to experiment with 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\DeclareDocumentMetadata{<options>}\MessageBreak
22         before~\string\documentclass
23     }
24 }
25 </package>
<*debug>
26 \ProvidesExplPackage {tagpdf-debug} {2022-01-09} {0.93}
27 { debug code for tagpdf }
28 \@ifpackageloaded{tagpdf}{\PackageWarning{tagpdf-debug}{tagpdf~not~loaded,~quitting}\ending
29     \end{macrocode}
30 </debug>
31 % We map the internal module name \enquote{tag} to \enquote{tagpdf} in messages.
32 % \begin{macrocode}
33 <*package>
34 \prop_gput:Nnn \g_msg_module_name_prop { tag }{ tagpdf }
35 </package>

```

Debug mode has its special mapping:

```

36 <*debug>
37 \prop_gput:Nnn \g_msg_module_type_prop { tag / debug } {}
38 \prop_gput:Nnn \g_msg_module_name_prop { tag / debug }{tagpdf~DEBUG}
39 </debug>

```

2 Package options

There are only two options to switch for luatex between generic and luamode, TODO try to get rid of them.

```

40 <*package>
41 \bool_new:N\g__tag_mode_lua_bool
42 \DeclareOption {luamode} { \sys_if_engine luatex:T { \bool_gset_true:N \g__tag_mode_lua_bo
43 \DeclareOption {genericmode}{ \bool_gset_false:N\g__tag_mode_lua_bool }
44 \ExecuteOptions{luamode}

```

```
45 \ProcessOptions
```

3 Packages

We need the temporary version of l3ref until this is in the kernel.

```
46 \RequirePackage{l3ref-tmp}
```

4 Temporary code

This is code which will be removed when proper support exists in LaTeX

4.1 a LastPage label

See also issue #2 in Accessible-xref

```
\__tag_lastpagelabel:
```

```
47 \cs_new_protected:Npn \__tag_lastpagelabel:
48 {
49   \legacy_if:nT { @filesw }
50   {
51     \exp_args:NNnx \exp_args:NNx\iow_now:Nn \@auxout
52     {
53       \token_to_str:N \newlabeldata
54       {__tag_LastPage}
55       {
56         {abspage} { \int_use:N \g_shipout_readonly_int}
57         {tagmcabs}{ \int_use:N \c@g__tag_MCID_abs_int }
58       }
59     }
60   }
61 }
62
63 \AddToHook{enddocument/afterlastpage}
64 { \__tag_lastpagelabel: }
```

(End definition for __tag_lastpagelabel:.)

\ref_value:nnn This allows to locally set a default value if the label or the attribute doesn't exist.

```
65 \cs_if_exist:NF \ref_value:nnn
66 {
67   \cs_new:Npn \ref_value:nnn #1#2#3
68   {
69     \exp_args:Nee
70     \__ref_value:nnn
71     { \tl_to_str:n {#1} } { \tl_to_str:n {#2} } {#3}
72   }
73   \cs_new:Npn \__ref_value:nnn #1#2#3
74   {
75     \tl_if_exist:cTF { g__ref_label_ #1 _ #2 _tl }
76     { \tl_use:c { g__ref_label_ #1 _ #2 _tl } }
77     {
78       #3
```

```

79         }
80     }
81 }

```

(End definition for `\ref_value:nnn`. This function is documented on page 3.)

5 Variables

```

\l__tag_tmpa_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

```

A few temporary variables

```

82 \tl_new:N \l__tag_tmpa_tl
83 \str_new:N \l__tag_tmpa_str
84 \prop_new:N \l__tag_tmpa_prop
85 \seq_new:N \l__tag_tmpa_seq
86 \seq_new:N \l__tag_tmpb_seq
87 \clist_new:N \l__tag_tmpa_clist
88 \int_new:N \l__tag_tmpa_int
89 \box_new:N \l__tag_tmpa_box
90 \box_new:N \l__tag_tmpb_box

```

(End 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_refmc_clist
\c__tag_refstruct_clist
91 \clist_const:Nn \c__tag_refmc_clist {tagabspage,tagmcabs,tagmcid}
92 \clist_const:Nn \c__tag_refstruct_clist {tagstruct,tagstructobj}

```

(End definition for `\c__tag_refmc_clist` and `\c__tag_refstruct_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.

```

93 \int_new:N \l__tag_loglevel_int

```

(End definition for `\l__tag_loglevel_int`.)

`\g__tag_active_space_bool` These booleans should help to control the global behaviour of tagpdf. Ideally it should more or less do nothing if all are false. The space-boolean controles 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 if pdf version 2.0 is detected, 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.

```

94 \bool_new:N \g__tag_active_space_bool
95 \bool_new:N \g__tag_active_mc_bool
96 \bool_new:N \g__tag_active_tree_bool
97 \bool_new:N \g__tag_active_struct_bool
98 \bool_new:N \g__tag_active_struct_dest_bool
99 \bool_gset_true:N \g__tag_active_struct_dest_bool

```

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

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

```

100 \bool_new:N \l__tag_active_mc_bool
101 \bool_set_true:N \l__tag_active_mc_bool
102 \bool_new:N \l__tag_active_struct_bool
103 \bool_set_true:N \l__tag_active_struct_bool

```

(End definition for `\l__tag_active_mc_bool` and `\l__tag_active_struct_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.

```

104 \bool_new:N \g__tag_tagunmarked_bool

```

(End definition for `\g__tag_tagunmarked_bool`.)

6 Variants of l3 commands

```

105 \prg_generate_conditional_variant:Nnn \pdf_object_if_exist:n {e}{T,F}
106 \cs_generate_variant:Nn \pdf_object_ref:n {e}
107 \cs_generate_variant:Nn \pdfannot_dict_put:nnn {nnx}
108 \cs_generate_variant:Nn \pdffile_embed_stream:nnn {nxx,oxx}
109 \cs_generate_variant:Nn \prop_gput:Nnn {Nxx}
110 \cs_generate_variant:Nn \prop_put:Nnn {Nxx}
111 \cs_generate_variant:Nn \ref_label:nn { nv }
112 \cs_generate_variant:Nn \seq_set_split:Nnn{Nne}
113 \cs_generate_variant:Nn \str_set_convert:Nnnn {Nonn, Noon, Nnon }

```

7 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`.

```

114 \ref_attribute_gset:nnnn { tagstruct } {0} { now }
115 { \int_use:N \c@g__tag_struct_abs_int }
116 \ref_attribute_gset:nnnn { tagstructobj } {} { now }
117 {
118   \pdf_object_if_exist:eT {__tag/struct/\int_use:N \c@g__tag_struct_abs_int}
119   {
120     \pdf_object_ref:ef{__tag/struct/\int_use:N \c@g__tag_struct_abs_int}
121   }
122 }
123 \ref_attribute_gset:nnnn { tagabspage } {0} { shipout }
124 { \int_use:N \g_shipout_readonly_int }
125 \ref_attribute_gset:nnnn { tagmcabs } {0} { now }
126 { \int_use:N \c@g__tag_MCID_abs_int }
127 \ref_attribute_gset:nnnn {tagmcid } {0} { now }
128 { \int_use:N \g__tag_MCID_tmp_bypage_int }

```

(End definition for `tagstruct` and others. These functions are documented on page 3.)

8 Label commands

```

\__tag_ref_label:nn A version of \ref_label:nn to set a label which takes a keyword mc or struct to call
the relevant lists. TODO: check if \@bsphack and \@esphack make sense here.
129 \cs_new_protected:Npn \__tag_ref_label:nn #1 #2 {%#1 label, #2 name of list mc or struct
130   {
131     \@bsphack
132     \ref_label:nv {#1}{c__tag_ref#2_clist}
133     \@esphack
134   }
135 \cs_generate_variant:Nn \__tag_ref_label:nn {en}

```

(End definition for `__tag_ref_label:nn`.)

```

\__tag_ref_value:nnn A local version to retrieve the value. It is a direct wrapper, but to keep naming consistent
.... It uses the variant defined temporarily above.
136 \cs_new:Npn \__tag_ref_value:nnn #1 #2 #3 {%#1 label, #2 attribute, #3 default
137   {
138     \ref_value:nnn {#1}{#2}{#3}
139   }
140 \cs_generate_variant:Nn \__tag_ref_value:nnn {enn}

```

(End definition for `__tag_ref_value:nnn`.)

```

\__tag_ref_value_lastpage:nn A command to retrieve the lastpage label, this will be adapted when there is a proper,
kernel lastpage label.
141 \cs_new:Npn \__tag_ref_value_lastpage:nn #1 #2
142   {
143     \ref_value:nnn {\__tag_LastPage}{#1}{#2}
144   }

```

(End definition for `__tag_ref_value_lastpage:nn`.)

9 Commands to fill seq and prop

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

```

\__tag_prop_new:N
\__tag_seq_new:N
145 \cs_set_eq:NN \__tag_prop_new:N \prop_new:N
\__tag_prop_gput:Nnn
146 \cs_set_eq:NN \__tag_seq_new:N \seq_new:N
\__tag_seq_gput_right:Nn
147 \cs_set_eq:NN \__tag_prop_gput:Nnn \prop_gput:Nnn
\__tag_seq_item:cn
148 \cs_set_eq:NN \__tag_seq_gput_right:Nn \seq_gput_right:Nn
\__tag_prop_item:cn
149 \cs_set_eq:NN \__tag_seq_item:cn \seq_item:cn
\__tag_seq_show:N
150 \cs_set_eq:NN \__tag_prop_item:cn \prop_item:cn
\__tag_prop_show:N
151 \cs_set_eq:NN \__tag_seq_show:N \seq_show:N
152 \cs_set_eq:NN \__tag_prop_show:N \prop_show:N
153
154 \cs_generate_variant:Nn \__tag_prop_gput:Nnn { Nxn , Nxx , Nnx , cnn , cxn , cnx , cno }
155 \cs_generate_variant:Nn \__tag_seq_gput_right:Nn { Nx , No , cn , cx }

```



```

156 \cs_generate_variant:Nn \__tag_prop_new:N { c }
157 \cs_generate_variant:Nn \__tag_seq_new:N { c }
158 \cs_generate_variant:Nn \__tag_seq_show:N { c }
159 \cs_generate_variant:Nn \__tag_prop_show:N { c }

```

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

10 General tagging commands

`\tag_stop_group_begin:` We need a command to stop tagging in some places. This simply switches the two local
`\tag_stop_group_end:` booleans.

```

160 \cs_new_protected:Npn \tag_stop_group_begin:
161 {
162   \group_begin:
163   \bool_set_false:N \l__tag_active_struct_bool
164   \bool_set_false:N \l__tag_active_mc_bool
165 }
166 \cs_set_eq:NN \tag_stop_group_end: \group_end:

```

(End definition for `\tag_stop_group_begin:` and `\tag_stop_group_end:`. These functions are documented on page 3.)

11 Keys for tagpdfsetup

TODO: the log-levels must be sorted

`activate-space_␣(setup-key)` Keys to (globally) activate tagging. `activate-space` activates the additional parsing
`activate-mc_␣(setup-key)` needed for interword spaces. It is not documented, the parsing is currently implicitly
`activate-tree_␣(setup-key)` activated by the known key `interwordspace`, as the code will perhaps move to some
`activate-struct_␣(setup-key)` other place, now that it is better separated. `no-struct-dest` allows to suppress structure
`activate-all_␣(setup-key)` destinations.
`no-struct-dest_␣(setup-key)`

```

167 \keys_define:nn { __tag / setup }
168 {
169   activate-space .bool_gset:N = \g__tag_active_space_bool,
170   activate-mc .bool_gset:N = \g__tag_active_mc_bool,
171   activate-tree .bool_gset:N = \g__tag_active_tree_bool,
172   activate-struct .bool_gset:N = \g__tag_active_struct_bool,
173   activate-all .meta:n =
174     {activate-mc={#1},activate-tree={#1},activate-struct={#1}},
175   activate-all .default:n = true,
176   no-struct-dest .bool_gset_inverse:N = \g__tag_active_struct_dest_bool,
177

```

(End definition for `activate-space (setup-key)` and others. These functions are documented on page 3.)

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

```

178   log .choice:,
179   log / none .code:n = {\int_set:Nn \l__tag_loglevel_int { 0 }},
180   log / v .code:n =
181   {

```

```

182     \int_set:Nn \l__tag_loglevel_int { 1 }
183     \cs_set_protected:Nn \__tag_check_typeout_v:n { \iow_term:x {##1} }
184   },
185   log / vv      .code:n = {\int_set:Nn \l__tag_loglevel_int { 2 }},
186   log / vvv     .code:n = {\int_set:Nn \l__tag_loglevel_int { 3 }},
187   log / all     .code:n = {\int_set:Nn \l__tag_loglevel_int { 10 }},

```

(End definition for `log` (setup-key). This function is documented on page 3.)

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

```

188   tagunmarked      .bool_gset:N = \g__tag_tagunmarked_bool,
189   tagunmarked      .initial:n = true,

```

(End definition for `tagunmarked` (setup-key). This function is documented on page 3.)

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

```

190   tabsorder      .choice:,
191   tabsorder / row      .code:n =
192     \pdfmanagement_add:nnn { Page } {Tabs}{/R},
193   tabsorder / column  .code:n =
194     \pdfmanagement_add:nnn { Page } {Tabs}{/C},
195   tabsorder / structure .code:n =
196     \pdfmanagement_add:nnn { Page } {Tabs}{/S},
197   tabsorder / none    .code:n =
198     \pdfmanagement_remove:nn {Page} {Tabs},
199   tabsorder      .initial:n = structure,
200   uncompress     .code:n = { \pdf_uncompress: },
201   }

```

(End definition for `tabsorder` (setup-key). This function is documented on page 3.)

12 loading of engine/more dependent code

```

202 \sys_if_engine luatex:T
203 {
204   \file_input:n {tagpdf-luatex.def}
205 }
206 </package>
207 <*mcloding>
208 \bool_if:NTF \g__tag_mode_lua_bool
209 {
210   \RequirePackage {tagpdf-mc-code-lua}
211 }
212 {
213   \RequirePackage {tagpdf-mc-code-generic} %
214 }
215 </mcloding>
216 <*debug>
217 \bool_if:NTF \g__tag_mode_lua_bool
218 {

```

```
219 \RequirePackage {tagpdf-debug-lua}  
220 }  
221 {  
222 \RequirePackage {tagpdf-debug-generic} %  
223 }  
224 </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. Currently the only sensible values for the argument *<keyword>* are `mc_tag` and `struct_tag`.

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	

2.2 Messages in checks and commands

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

2.3 Messages from the ptagging code

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

2.4 Warning messages from the lua-code

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

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

2.5 Info messages from the lua-code

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

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

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

2.6 Debug mode messages and code

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

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

```

1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-checks-code} {2022-01-09} {0.93}
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 issue is 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 definition for mc-nested. This function is documented on page ??.)

mc-tag-missing If the tag is missing

```

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

```

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

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 definition for mc-label-unknown. This function is documented on page ??.)

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 definition for mc-used-twice. This function is documented on page ??.)

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 definition for mc-not-open. This function is documented on page ??.)

mc-pushed Informational messages about mc-pushing.

```
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 definition for mc-pushed and mc-popped. These functions are documented on page ??.)

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 definition for mc-current. This function is documented on page ??.)

3.2 Messages related to mc-chunks

struct-no-objnum Should not happen ...

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

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

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

```
23 \msg_new:nnn { tag }
24 {struct-faulty-nesting}
25 { there~is~no~open~structure~on~the~stack }
```

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

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

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

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

struct-used-twice

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

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

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

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

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

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

```
31 \msg_new:nnn { tag } {struct-show-closing}
32   { closing-structure~#1~tagged~\prop_item:cn{g__tag_struct_#1_prop}{S} }
```

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

3.3 Attributes

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

`attr-unknown`

```
33 \msg_new:nnn { tag } {attr-unknown} { attribute~#1~is~unknown }
```

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

3.4 Roles

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

```
role-unknown 34 \msg_new:nnn { tag } {role-missing} { tag~#1~has~no~role~assigned }
role-unknown-tag 35 \msg_new:nnn { tag } {role-unknown} { role~#1~is~not~known }
36 \msg_new:nnn { tag } {role-unknown-tag} { tag~#1~is~not~known }
```

(End definition for `role-missing`, `role-unknown`, and `role-unknown-tag`. These functions are documented on page ??.)

`role-tag` Info messages.

```
new-tag 37 \msg_new:nnn { tag } {role-tag} { mapping-tag~#1~to~role~#2 }
38 \msg_new:nnn { tag } {new-tag} { adding-new-tag~#1 }
```

(End definition for `role-tag` and `new-tag`. These functions are documented on page ??.)

3.5 Miscellaneous

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

```
39 \msg_new:nnn { tag } {tree-mcid-index-wrong}
40   {something~is~wrong~with~the~mcid--rerun}
```

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

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

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

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

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

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

(End definition for `__tag_check_typeout_v:n`.)

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

```

44 \msg_new:nnnn { tag } {para-hook-count-wrong}
45   {The~number~of~automatic~begin~(#1)~and~end~(#2)~para-hooks~differ!}
46   {This~quite~probably~a~coding~error~and~the~structure~will~be~wrong!}

```

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

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` and `struct_tag`.

```

47 \cs_new:Npn \tag_get:n #1 { \use:c {__tag_get_data_#1: } }

```

(End definition for \tag_get:n. This function is documented on page 12.)

5 User conditionals

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

```

48 \prg_new_conditional:Npnn \tag_if_active: { p , T , TF , F }
49 {
50   \bool_lazy_all:nTF
51   {
52     {\g__tag_active_struct_bool}
53     {\g__tag_active_mc_bool}
54     {\g__tag_active_tree_bool}
55     {\l__tag_active_struct_bool}
56     {\l__tag_active_mc_bool}
57   }
58   {
59     \prg_return_true:
60   }
61   {
62     \prg_return_false:
63   }
64 }

```

(End definition for \tag_if_active:TF. This function is documented on page 12.)

6 Internal checks

These are checks used in various places in the code.

6.1 checks for active tagging

`__tag_check_if_active_mc:TF` 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.

`__tag_check_if_active_struct:TF`

```

65 \prg_new_conditional:Npnn \__tag_check_if_active_mc: {T,F,TF}
66 {
67   \bool_lazy_and:nnTF { \g__tag_active_mc_bool } { \l__tag_active_mc_bool }
68   {
69     \prg_return_true:
70   }
71   {
72     \prg_return_false:
73   }
74 }
75 \prg_new_conditional:Npnn \__tag_check_if_active_struct: {T,F,TF}
76 {
77   \bool_lazy_and:nnTF { \g__tag_active_struct_bool } { \l__tag_active_struct_bool }
78   {
79     \prg_return_true:
80   }
81   {
82     \prg_return_false:
83   }
84 }

```

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

6.2 Checks related to stuctures

`__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.

```

85 \cs_new_protected:Npn \__tag_check_structure_has_tag:n #1 %#1 struct num
86 {
87   \prop_if_in:cnF { \g__tag_struct_#1_prop }
88   {S}
89   {
90     \msg_error:nn { tag } {struct-missing-tag}
91   }
92 }

```

(End 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.

```

93 \cs_new_protected:Npn \__tag_check_structure_tag:N #1
94 {
95   \prop_if_in:Nof \g__tag_role_tags_prop {#1}
96   {
97     \msg_warning:nnx { tag } {role-unknown-tag} {#1}
98   }
99 }

```

(End 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.

```

100 \cs_new_protected:Npn __tag_check_info_closing_struct:n #1 %#1 struct num
101 {
102   \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
103   {
104     \msg_info:nnn { tag } {struct-show-closing} {#1}
105   }
106 }
107
108 \cs_generate_variant:Nn __tag_check_info_closing_struct:n {o,x}

```

(End 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.

```

109 \cs_new_protected:Npn __tag_check_no_open_struct:
110 {
111   \msg_error:nn { tag } {struct-faulty-nesting}
112 }

```

(End definition for `__tag_check_no_open_struct:`.)

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

```

113 \cs_new_protected:Npn __tag_check_struct_used:n #1 %#1 label
114 {
115   \prop_get:cnNT
116     {g__tag_struct_}__tag_ref_value:enn{tagpdfstruct-#1}{tagstruct}{unknown}_prop}
117     {P}
118     \l_tmpa_tl
119     {
120       \msg_warning:nnn { tag } {struct-used-twice} {#1}
121     }
122 }

```

(End 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.

```

123 \cs_new_protected:Npn __tag_check_add_tag_role:nn #1 #2 %#1 tag, #2 role
124 {
125   \tl_if_empty:nTF {#2}
126   {
127     \msg_warning:nnn { tag } {role-missing} {#1}
128   }
129   {
130     \prop_get:NnNTF \g__tag_role_tags_prop {#2} \l_tmpa_tl
131     {
132       \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
133       {
134         \msg_info:nnnn { tag } {role-tag} {#1} {#2}
135       }
136     }
137   }

```

```

138         \msg_warning:nnn { tag } {role-unknown} {#2}
139     }
140 }
141 }

```

(End 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).

```

142 \cs_new_protected:Npn \__tag_check_mc_if_nested:
143 {
144     \__tag_mc_if_in:T
145     {
146         \msg_warning:nnx { tag } {mc-nested} { \__tag_get_mc_abs_cnt: }
147     }
148 }
149
150 \cs_new_protected:Npn \__tag_check_mc_if_open:
151 {
152     \__tag_mc_if_in:F
153     {
154         \msg_warning:nnx { tag } {mc-not-open} { \__tag_get_mc_abs_cnt: }
155     }
156 }

```

(End 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.

```

157 \cs_new_protected:Npn \__tag_check_mc_pushed_popped:nn #1 #2
158 {
159     \int_compare:nNnT
160     { \l__tag_loglevel_int } = { 2 }
161     { \msg_info:nnx {tag}{mc-#1}{#2} }
162     \int_compare:nNnT
163     { \l__tag_loglevel_int } > { 2 }
164     {
165         \msg_info:nnx {tag}{mc-#1}{#2}
166         \seq_log:N \g__tag_mc_stack_seq
167     }
168 }

```

(End definition for `__tag_check_mc_pushed_popped:nn`.)

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

```

169 \cs_new_protected:Npn \__tag_check_mc_tag:N #1 % #1 is var with a tag name in it
170 {
171     \tl_if_empty:NT #1
172     {
173         \msg_error:nnx { tag } {mc-tag-missing} { \__tag_get_mc_abs_cnt: }
174     }

```

```

175 \prop_if_in:NoF \g__tag_role_tags_NS_prop {#1}
176 {
177   \msg_warning:nxx { tag } {role-unknown-tag} {#1}
178 }
179 }

```

(End definition for __tag_check_mc_tag:N.)

\g__tag_check_mc_used_intarray This variable holds the list of used mc numbers. Everytime we store a mc-number we
__tag_check_init_mc_used: 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??

```

180 \cs_new_protected:Npn \__tag_check_init_mc_used:
181 {
182   \intarray_new:Nn \g__tag_check_mc_used_intarray { 65536 }
183   \cs_gset_eq:NN \__tag_check_init_mc_used: \prg_do_nothing:
184 }

```

(End 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.

```

185 \cs_new_protected:Npn \__tag_check_mc_used:n #1 %#1 mcid absent
186 {
187   \int_compare:nNnT {\l__tag_loglevel_int} > { 2 }
188   {
189     \__tag_check_init_mc_used:
190     \intarray_gset:Nnn \g__tag_check_mc_used_intarray
191       {#1}
192       { \intarray_item:Nn \g__tag_check_mc_used_intarray {#1} + 1 }
193     \int_compare:nNnT
194       {
195         \intarray_item:Nn \g__tag_check_mc_used_intarray {#1}
196       }
197       >
198       { 1 }
199     {
200       \msg_warning:nnn { tag } {mc-used-twice} {#1}
201     }
202   }
203 }

```

(End definition for __tag_check_mc_used:n.)

__tag_check_show_MCID_by_page: This allows to show the mc on a page. Currently unused.

```

204 \cs_new_protected:Npn \__tag_check_show_MCID_by_page:
205 {
206   \tl_set:Nx \l__tag_tmpa_tl
207   {
208     \__tag_ref_value_lastpage:nn
209     {abspage}
210     {-1}

```

```

211     }
212     \int_step_inline:nnnn {1}{1}
213     {
214         \l__tag_tmpa_tl
215     }
216     {
217         \seq_clear:N \l_tmpa_seq
218         \int_step_inline:nnnn
219             {1}
220             {1}
221             {
222                 \__tag_ref_value_lastpage:nn
223                 {tagmcabs}
224                 {-1}
225             }
226             {
227                 \int_compare:nT
228                     {
229                         \__tag_ref_value:enn
230                         {mcid-####1}
231                         {tagabspage}
232                         {-1}
233                     }
234                     =
235                     ##1
236                     {
237                         \seq_gput_right:Nx \l_tmpa_seq
238                         {
239                             Page##1-####1-
240                             \__tag_ref_value:enn
241                             {mcid-####1}
242                             {tagmcid}
243                             {-1}
244                         }
245                     }
246             }
247         \seq_show:N \l_tmpa_seq
248     }
249 }

```

(End 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

`__tag_check_mc_in_galley:` **TF**

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`.

```

250 \prg_new_conditional:Npnn \__tag_check_if_mc_in_galley: { T,F,TF }
251 {
252   \tl_if_eq:NNTF \l__tag_mc_firstmarks_seq \l__tag_mc_botmarks_seq
253   { \prg_return_false: }
254   { \prg_return_true: }
255 }

```

(End definition for __tag_check_mc_in_galley:TF.)

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

`__tag_check_if_mc_tmb_missing:TF`

```

256 \prg_new_conditional:Npnn \__tag_check_if_mc_tmb_missing: { T,F,TF }
257 {
258   \bool_if:nTF
259   {
260     \str_if_eq_p:ee {\seq_item:Nn \l__tag_mc_firstmarks_seq {1}}{e-}
261     ||
262     \str_if_eq_p:ee {\seq_item:Nn \l__tag_mc_firstmarks_seq {1}}{b+}
263   }
264   { \prg_return_true: }
265   { \prg_return_false: }
266 }

```

(End definition for __tag_check_if_mc_tmb_missing:TF.)

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

`__tag_check_if_mc_tme_missing:TF`

```

267 \prg_new_conditional:Npnn \__tag_check_if_mc_tme_missing: { T,F,TF }
268 {
269   \str_if_eq:eeTF {\seq_item:Nn \l__tag_mc_botmarks_seq {1}}{b+}
270   { \prg_return_true: }
271   { \prg_return_false: }
272 }

```

(End definition for __tag_check_if_mc_tme_missing:TF.)

```

273 </package>

```

```

274 <*debug>

```

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

```

275 \msg_new:nnn { tag / debug } {mc-begin} { MC~begin~#1~with~options:~\tl_to_str:n{#2}~[\msg_line_info:]}
276 \msg_new:nnn { tag / debug } {mc-end} { MC~end~#1~[\msg_line_context:] }
277
278 \cs_new_protected:Npn \__tag_debug_mc_begin_insert:n #1
279 {
280   \int_compare:nNtT { \l__tag_loglevel_int } > {0}
281   {
282     \msg_note:nnnn { tag / debug } {mc-begin} {inserted} { #1 }
283   }

```

```

284 }
285 \cs_new_protected:Npn \__tag_debug_mc_begin_ignore:n #1
286 {
287   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
288   {
289     \msg_note:nnnn { tag / debug } {mc-begin } {ignored} { #1 }
290   }
291 }
292 \cs_new_protected:Npn \__tag_debug_mc_end_insert:
293 {
294   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
295   {
296     \msg_note:nnn { tag / debug } {mc-end} {inserted}
297   }
298 }
299 \cs_new_protected:Npn \__tag_debug_mc_end_ignore:
300 {
301   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
302   {
303     \msg_note:nnn { tag / debug } {mc-end } {ignored}
304   }
305 }

```

And now something for the structures

```

306 \msg_new:nnn { tag / debug } {struct-begin}
307 {
308   Struct-begin~#1~with-options:~\tl_to_str:n{#2}~[\msg_line_context:]
309 }
310 \msg_new:nnn { tag / debug } {struct-end}
311 {
312   Struct-end~#1~[\msg_line_context:]
313 }
314
315 \cs_new_protected:Npn \__tag_debug_struct_begin_insert:n #1
316 {
317   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
318   {
319     \msg_note:nnnn { tag / debug } {struct-begin} {inserted} { #1 }
320     \seq_log:N \g__tag_struct_tag_stack_seq
321   }
322 }
323 \cs_new_protected:Npn \__tag_debug_struct_begin_ignore:n #1
324 {
325   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
326   {
327     \msg_note:nnnn { tag / debug } {struct-begin } {ignored} { #1 }
328   }
329 }
330 \cs_new_protected:Npn \__tag_debug_struct_end_insert:
331 {
332   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
333   {
334     \msg_note:nnn { tag / debug } {struct-end} {inserted}
335     \seq_log:N \g__tag_struct_tag_stack_seq
336   }

```



```
337 }
338 \cs_new_protected:Npn \__tag_debug_struct_end_ignore:
339 {
340   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
341   {
342     \msg_note:nnn { tag / debug } {struct-end } {ignored}
343   }
344 }
345 </debug>
```

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

Symbols	
<code>\</code>	10
A	
<code>activate-all_□(setup-key)</code>	3, <u>167</u>
<code>activate-mc_□(setup-key)</code>	3, <u>167</u>
<code>activate-space_□(setup-key)</code>	3, <u>167</u>
<code>activate-struct_□(setup-key)</code>	3, <u>167</u>
<code>activate-tree_□(setup-key)</code>	3, <u>167</u>
<code>\AddToHook</code>	63
<code>attr-unknown</code>	<u>33</u>
B	
<code>\begin</code>	32
bool commands:	
<code>\bool_gset_false:N</code>	43
<code>\bool_gset_true:N</code>	42, 99
<code>\bool_if:NTF</code>	18, 208, <u>217</u>
<code>\bool_if:nTF</code>	6, <u>258</u>
<code>\bool_lazy_all:nTF</code>	50
<code>\bool_lazy_and:nnTF</code>	67, <u>77</u>
<code>\bool_lazy_and_p:nn</code>	8
<code>\bool_new:N</code>	
..	41, 94, 95, 96, 97, 98, 100, 102, 104
<code>\bool_set_false:N</code>	163, 164
<code>\bool_set_true:N</code>	101, <u>103</u>
box commands:	
<code>\box_new:N</code>	89, 90
C	
c@g internal commands:	
<code>\c@g__tag_MCID_abs_int</code>	57, <u>126</u>
<code>\c@g__tag_struct_abs_int</code>	115, <u>118</u> , <u>120</u>
clist commands:	
<code>\clist_const:Nn</code>	91, <u>92</u>
<code>\clist_new:N</code>	<u>87</u>
cs commands:	
<code>\cs_generate_variant:Nn</code> ..	106, 107, 108, 108, 109, 110, 111, 112, 113, 135, 140, 154, 155, 156, 157, 158, 159
<code>\cs_gset_eq:NN</code>	183
<code>\cs_if_exist:NTF</code>	65
<code>\cs_if_exist_p:N</code>	9
<code>\cs_new:Npn</code>	47, 67, 73, 136, <u>141</u>
<code>\cs_new_protected:Npn</code>	47, 85, 93, 100, 109, 113, 123, 129, 142, 150, 157, 160, 169, 180, 185, 204, 278, 285, 292, 299, 315, 323, 330, 338
<code>\cs_set_eq:NN</code>	43, 145, 146, 147, 148, 149, 150, 151, 152, 166
<code>\cs_set_protected:Nn</code>	183
D	
<code>\DeclareDocumentMetadata</code>	21
<code>\DeclareOption</code>	42, <u>43</u>
<code>\documentclass</code>	22
E	
<code>\end</code>	29
<code>\endinput</code>	28
<code>\enquote</code>	31
<code>\ExecuteOptions</code>	44
exp commands:	
<code>\exp_args:Nee</code>	69
<code>\exp_args:NNnx</code>	51
<code>\exp_args:NNx</code>	51
F	
file commands:	
<code>\file_input:n</code>	204
G	
group commands:	
<code>\group_begin:</code>	162
<code>\group_end:</code>	166
I	
int commands:	
<code>\int_compare:nNnTF</code>	
..	102, 132, 159, 162, 187, 193, 280, 287, 294, 301, 317, 325, 332, 340
<code>\int_compare:nTF</code>	227
<code>\int_new:N</code>	88, 93
<code>\int_set:Nn</code> ...	179, 182, 185, 186, 187
<code>\int_step_inline:nnnn</code>	212, 218
<code>\int_use:N</code>	
..	56, 57, 115, 118, 120, 124, 126, 128
intarray commands:	
<code>\intarray_gset:Nnn</code>	190
<code>\intarray_item:Nn</code>	192, 195
<code>\intarray_new:Nn</code>	182
iow commands:	
<code>\iow_now:Nn</code>	51
<code>\iow_term:n</code>	183
K	
keys commands:	
<code>\keys_define:nn</code>	167

L		
legacy commands:		
\legacy_if:nTF	49	
log _␣ (setup-key)	3, 178	
M		
mc-current	16	
mc-label-unknown	9	
mc-nested	6	
mc-not-open	13	
mc-popped	14	
mc-pushed	14	
mc-tag-missing	8	
mc-used-twice	12	
\MessageBreak	15, 19, 20, 21	
msg commands:		
\msg_error:nn	90, 111	
\msg_error:nnn	173	
\msg_info:nnn	104, 161, 165	
\msg_info:nnnn	134	
\msg_line_context:	275, 276, 308, 312	
\g_msg_module_name_prop	34, 38	
\g_msg_module_type_prop	37	
\msg_new:nnn	7, 8, 9, 12, 13, 14,	
15, 16, 22, 23, 26, 27, 29, 31, 33, 34,		
35, 36, 37, 38, 39, 41, 275, 276, 306, 310		
\msg_new:nnnn	44	
\msg_note:nnn	296, 303, 334, 342	
\msg_note:nnnn	282, 289, 319, 327	
\msg_warning:nnn		
97, 120, 127, 138, 146, 154, 177, 200		
N		
new-tag	37	
\newlabeldata	53	
no-struct-dest _␣ (setup-key)	3, 167	
P		
\PackageError	13	
\PackageWarning	28	
para-hook-count-wrong	44	
pdf commands:		
\pdf_object_if_exist:n	105	
\pdf_object_if_exist:nTF	118	
\pdf_object_ref:n	106, 120	
\pdf_uncompress:	200	
pdfannot commands:		
\pdfannot_dict_put:nnn	107	
pdffile commands:		
\pdffile_embed_stream:nnn	108	
pdfmanagement commands:		
\pdfmanagement_add:nnn	192, 194, 196	
\pdfmanagement_if_active_p:	9, 10	
\pdfmanagement_remove:nn	198	
prg commands:		
\prg_do_nothing:	183	
\prg_generate_conditional_-		
variant:Nnn	105	
\prg_new_conditional:Npnn		
48, 65, 75, 250, 256, 267		
\prg_return_false:		
62, 72, 82, 253, 265, 271		
\prg_return_true:		
59, 69, 79, 254, 264, 270		
\ProcessOptions	45	
prop commands:		
\prop_get:NnNTF	115, 130	
\prop_gput:Nnn	34, 37, 38, 109, 147	
\prop_if_in:NnTF	87, 95, 175	
\prop_item:Nn	32, 150	
\prop_new:N	84, 145	
\prop_put:Nnn	110	
\prop_show:N	152	
\ProvidesExplPackage	3, 3, 26	
R		
ref commands:		
\ref_attribute_gset:nnnn		
114, 116, 123, 125, 127		
\ref_label:nn	111, 132	
\ref_value:nnn	3, 65, 65, 67, 138, 143	
ref internal commands:		
_ref_value:nnn	70, 73	
\RequirePackage	20, 46, 210, 213, 219, 222	
role-missing	34	
role-tag	37	
role-unknown	34	
role-unknown-tag	34	
S		
seq commands:		
\seq_clear:N	217	
\seq_gput_right:Nn	148, 237	
\seq_item:Nn	149, 260, 262, 269	
\seq_log:N	166, 320, 335	
\seq_new:N	85, 86, 146	
\seq_set_split:Nnn	112	
\seq_show:N	151, 247	
\l_tmpa_seq	217, 237, 247	
shipout commands:		
\g_shipout_readonly_int	56, 124	
\ShowTagging	12	
str commands:		
\str_if_eq:nnTF	269	
\str_if_eq_p:nn	260, 262	
\str_new:N	83	
\str_set_convert:Nnnn	113	
\string	20, 21, 22	

struct-faulty-nesting	23	_tag_check_mc_in_galley_p:	250
struct-label-unknown	29	_tag_check_mc_pushed_popped:nn	
struct-missing-tag	26	157, 157
struct-no-objnum	22	_tag_check_mc_tag:N	169, 169
struct-show-closing	31	_tag_check_mc_used:n	185, 185
struct-used-twice	27	\\g_tag_check_mc_used_intarray	..
sys commands:		180, 190, 192, 195
\\sys_if_engine_luatex:TF	42, 202	_tag_check_no_open_struct:	109, 109
sys-no-interwordspace	41	_tag_check_show_MCID_by_page:	..
		204, 204
		_tag_check_struct_used:n	113, 113
		_tag_check_structure_has_tag:n	
		85, 85
		_tag_check_structure_tag:N	93, 93
		_tag_check_typeof_v:n	43, 43, 183
		_tag_debug_mc_begin_ignore:n	285
		_tag_debug_mc_begin_insert:n	278
		_tag_debug_mc_end_ignore:	299
		_tag_debug_mc_end_insert:	292
		_tag_debug_struct_begin_-	
		ignore:n	323
		_tag_debug_struct_begin_-	
		insert:n	315
		_tag_debug_struct_end_ignore:	338
		_tag_debug_struct_end_insert:	330
		_tag_get_mc_abs_cnt:
		19, 20, 146, 154, 173
		\\g_tag_in_mc_bool
		_tag_lastpagelabel:	47, 47, 64
		\\l_tag_loglevel_int
		93, 102, 132, 160,
			163, 179, 182, 185, 186, 187, 187,
			280, 287, 294, 301, 317, 325, 332, 340
		\\l_tag_mc_botmarks_seq	252, 269
		\\l_tag_mc_firstmarks_seq
		252, 260, 262
		_tag_mc_if_in:TF	144, 152
		\\g_tag_mc_key_tag_tl	19
		\\g_tag_mc_stack_seq	166
		\\g_tag_MCID_tmp_bypage_int	128
		\\g_tag_mode_lua_bool
		41, 42, 43, 208, 217
		_tag_prop_gput:Nnn	145, 147, 154
		_tag_prop_item:Nn	145, 150
		_tag_prop_new:N	145, 145, 156
		_tag_prop_show:N	145, 152, 159
		_tag_ref_label:nn	129, 129, 135
		_tag_ref_value:nnn
		116, 136, 136, 140, 229, 240
		_tag_ref_value_lastpage:nn	..
		141, 141, 208, 222
		\\c_tag_refmc_clist	91
		\\c_tag_refstruct_clist	91
		\\g_tag_role_tags_NS_prop	175

<code>\g__tag_role_tags_prop</code>	95, 130	TeX and L ^A T _E X 2 _ε commands:	
<code>__tag_seq_gput_right:Nn</code> 145 , 148 , 155		<code>\@auxout</code>	51
<code>__tag_seq_item:Nn</code>	145 , 149	<code>\@bsphack</code>	131
<code>__tag_seq_new:N</code>	145 , 146 , 157	<code>\@esphack</code>	133
<code>__tag_seq_show:N</code>	145 , 151 , 158	<code>\@ifpackageloaded</code>	28
<code>\g__tag_struct_tag_stack_seq</code> 320 , 335		tl commands:	
<code>\g__tag_tagunmarked_bool</code> . . . 104 , 188		<code>\tl_if_empty:N</code>	171
<code>\l__tag_tmpa_box</code>	82	<code>\tl_if_empty:n</code>	125
<code>\l__tag_tmpa_clist</code>	82	<code>\tl_if_eq:NNTF</code>	252
<code>\l__tag_tmpa_int</code>	82	<code>\tl_if_exist:N</code>	75
<code>\l__tag_tmpa_prop</code>	82	<code>\tl_new:N</code>	82
<code>\l__tag_tmpa_seq</code>	82	<code>\tl_set:Nn</code>	206
<code>\l__tag_tmpa_str</code>	82	<code>\tl_to_str:n</code>	71 , 275 , 308
<code>\l__tag_tmpa_tl</code>	82 , 206 , 214	<code>\tl_use:N</code>	76
<code>\l__tag_tmpb_box</code>	82	<code>\l_tmpa_tl</code>	118 , 130
<code>\l__tag_tmpb_seq</code>	82	token commands:	
<code>tagabspage</code>	3 , 114	<code>\token_to_str:N</code>	53
<code>tagmcabs</code>	3 , 114	<code>tree-mcid-index-wrong</code>	39
<code>tagmcid</code>	3 , 114		
<code>tagstruct</code>	3 , 114		
<code>tagstructobj</code>	3 , 114		
<code>tagunmarked_␣(setup-key)</code>	3 , 188		

U

use commands:	
<code>\use:N</code>	47
<code>\use_none:n</code>	43