

tagpdf – A package to experiment with pdf tagging*

Ulrike Fischer[†]

Released 2022-01-09

Contents

1	Initialization and test if pdfmanagement is active.	7
2	Package options	7
3	Packages	8
4	Temporary code	8
4.1	a LastPage label	8
5	Variables	9
6	Variants of l3 commands	10
7	Setup label attributes	10
8	Label commands	11
9	Commands to fill seq and prop	11
10	General tagging commands	12
11	Keys for tagpdfsetup	12
12	loading of engine/more dependent code	13
I	The tagpdf-checks module	
	Messages and check code	
	Part of the tagpdf package	15
1	Commands	15

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

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

2	Description of log messages	15
2.1	\ShowTagging command	15
2.2	Messages in checks and commands	15
2.3	Messages from the ptagging code	16
2.4	Warning messages from the lua-code	16
2.5	Info messages from the lua-code	16
2.6	Debug mode messages and code	17
2.7	Messages	17
3	Messages	18
3.1	Messages related to mc-chunks	18
3.2	Messages related to mc-chunks	19
3.3	Attributes	20
3.4	Roles	20
3.5	Miscellaneous	20
4	Retrieving data	21
5	User conditionals	21
6	Internal checks	21
6.1	checks for active tagging	21
6.2	Checks related to stuctures	22
6.3	Checks related to roles	23
6.4	Check related to mc-chunks	23
6.5	Checks related to the state of MC on a page or in a split stream	26
 II The tagpdf-user module		
Code related to L^AT_EX2e user commands and document commands		
Part of the tagpdf package		29
1	Setup commands	29
2	Commands related to mc-chunks	29
3	Commands related to structures	30
4	Debugging	30
5	Extension commands	30
5.1	Fake space	31
5.2	Paratagging	31
5.3	Header and footer	31
5.4	Link tagging	32
6	User commands and extensions of document commands	32
7	Setup and preamble commands	32
8	Commands for the mc-chunks	32

9	Commands for the structure	33
10	Debugging	33
11	Commands to extend document commands	36
11.1	Document structure	36
11.2	Structure destinations	37
11.3	Fake space	37
11.4	Paratagging	38
11.5	Header and footer	40
11.6	Links	42
III The tagpdf-tree module		
Commands trees and main dictionaries		
Part of the tagpdf package		44
1	Trees, pdfmanagement and finalization code	44
1.1	Catalog: MarkInfo and StructTreeRoot	44
1.2	Writing structure elements	45
1.3	ParentTree	45
1.4	Rolemap dictionary	48
1.5	Classmap dictionary	48
1.6	Namespaces	49
1.7	Finishing the structure	50
1.8	StructParents entry for Page	50
IV The tagpdf-mc-shared module		
Code related to Marked Content (mc-chunks), code shared by all modes		
Part of the tagpdf package		51
1	Public Commands	51
2	Public keys	52
3	Marked content code – shared	52
3.1	Variables and counters	53
3.2	Functions	54
3.3	Keys	56
V The tagpdf-mc-generic module		
Code related to Marked Content (mc-chunks), generic mode		
Part of the tagpdf package		58

1	Marked content code – generic mode	58
1.1	Variables	58
1.2	Functions	59
1.3	Looking at MC marks in boxes	62
1.4	Keys	69
VI	The <code>tagpdf-mc-luacode</code> module	
	Code related to Marked Content (mc-chunks), luamode-specific	
	Part of the tagpdf package	71
1	Marked content code – luamode code	71
1.1	Commands	72
1.2	Key definitions	76
VII	The <code>tagpdf-struct</code> module	
	Commands to create the structure	
	Part of the tagpdf package	79
1	Public Commands	79
2	Public keys	79
2.1	Keys for the structure commands	79
2.2	Setup keys	81
3	Variables	81
3.1	Variables used by the keys	83
4	Commands	84
4.1	Initialization of the StructTreeRoot	84
4.2	Handlings kids	85
5	Keys	90
6	User commands	94
7	Attributes and attribute classes	97
7.1	Variables	97
7.2	Commands and keys	98
VIII	The <code>tagpdf-luatex.def</code>	
	Driver for luatex	
	Part of the tagpdf package	101
1	Loading the lua	101
2	Logging functions	105

3	Helper functions	107
3.1	Retrieve data functions	107
3.2	Functions to insert the pdf literals	109
4	Function for the real space chars	110
5	Function for the tagging	113
6	Parenttree	117
IX	The tagpdf-roles module	
	Tags, roles and namespace code	
	Part of the tagpdf package	119
1	Code related to roles and structure names	119
1.1	Variables	119
1.2	Namespaces	120
1.3	Data	122
1.4	Adding new tags and rolemapping	128
1.4.1	pdf 1.7 and earlier	128
1.4.2	The pdf 2.0 version	129
1.5	Key-val user interface	129
X	The tagpdf-space module	
	Code related to real space chars	
	Part of the tagpdf package	132
1	Code for interword spaces	132
	Index	135

<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>tagabspage</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>
26
27 <*debug>
28 \ProvidesExplPackage {tagpdf-debug} {2022-01-09} {0.93}
29 { debug code for tagpdf }
30 \ifpackageloaded{tagpdf}{\PackageWarning{tagpdf-debug}{tagpdf~not~loaded,~quitting}}\endinput
31 \end{macrocode}
32 </debug>
33
34 % We map the internal module name \enquote{tag} to \enquote{tagpdf} in messages.
35 % \begin{macrocode}
36 <*package>
37 \prop_gput:Nnn \g_msg_module_name_prop { tag }{ tagpdf }
38 </package>
39
40 Debug mode has its special mapping:
41 <*debug>
42 \prop_gput:Nnn \g_msg_module_type_prop { tag / debug} {}
43 \prop_gput:Nnn \g_msg_module_name_prop { tag / debug }{tagpdf~DEBUG}
44 </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_bool } }
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 { @files }
50   {
51     \exp_args:NNx \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 6.)

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 {tagabspace,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:e{__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 6.)

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 overwrite 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 6.)

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 6.)

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_typeof_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 6.)

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 6.)

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 6.)

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
\tag_mc_begin:n	mc-begin-insert	msg	
	mc-begin-ignore	msg	if inactive

2.7 Messages

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

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

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

Various messages related to structure. TODO document their meaning.

attr-unknown

Message if an attribute is unknown.

role-missing
role-unknown
role-unknown-tag
role-tag
new-tag

Messages related to role mapping.

<hr/> <hr/> tree-mcid-index-wrong	Used in the tree code, typically indicates the document must be rerun.
<hr/> <hr/> sys-no-interwordspace	Message if an engine doesn't support inter word spaces
<hr/> <hr/> para-hook-count-wrong	Message if the number of begin paragraph and end paragraph differ. This normally means faulty structure.
	<pre> 1 <@@=tag> 2 <*header> 3 \ProvidesExplPackage {tagpdf-checks-code} {2022-01-09} {0.93} 4 {part of tagpdf - code related to checks, conditionals, debugging and messages} 5 </header> </pre>

3 Messages

3.1 Messages related to mc-chunks

mc-nested This message is 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 17.)

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 17.)

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 17.)

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 17.)

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 17.)

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 definition for mc-pushed and mc-popped. These functions are documented on page 17.)

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 17.)

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 17.)

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 17.)

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 17.)

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 17.)

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 17.)

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 17.)

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 17.)

3.4 Roles

role-missing

Warning message if either the tag or the role is missing

role-unknown

role-unknown-tag

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

```
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 17.)

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 17.)

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 18.)

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 18.)

_tag_check_typeout_v:n

A simple logging function. By default it 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 18.)

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 15.)

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 15.)

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
__tag_check_if_active_struct:TF this is missing. The argument is a number.

```
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 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.

```

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:nxx { 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:nxx { 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:nxx {tag}{mc-#1}{#2} }
162     \int_compare:nNnT
163     { \l__tag_loglevel_int } > { 2 }
164     {
165         \msg_info:nxx {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:nxx { 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 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 abscent
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             ##1
235         }
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 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.

```

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
 __tag_check_if_mc_tme_missing:TF this the case if the botmarks starts with b+. Like above we assume that the marks content
 is already in the seq’s.

```

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_no] }
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:nNnT { \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>

```

Part II

The **tagpdf-user** module

Code related to L^AT_EX2e user commands and document commands

Part of the tagpdf package

1 Setup commands

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

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

<code>activate_<setup-key></code>

And additional setup key which combine the other activate keys `activate-mc`, `activate-tree`, `activate-struct` and additionally add a document structure.

<code>\tagpdfifluatexTF</code>
<code>\tagpdfifluatexT</code>
<code>\tagpdfifpdfTeXT</code>

small wrappers around engine tests. This functions should not be used and will be removed in one of the next versions.

2 Commands related to mc-chunks

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

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

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

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

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

3 Commands related to structures

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

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

4 Debugging

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

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

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

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

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

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

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

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

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

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

5 Extension commands

The following commands and code parts are not core command 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 Paratagging

This is a first try to make use of the new paragraph hooks in a current 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>paratagging_␣(setup-key)</code>	<code>paratagging = true false</code>
<code>paratagging-show_␣(setup-key)</code>	<code>paratagging-show = true false</code>

This keys can be used in `\tagpdfsetup` and enable/disable paratagging. `paratagging-show` puts small red numbers at the begin and end of a paragraph. This is meant as a debugging help. The number are boxes and have a (tiny) height, so they can affect typesetting.

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

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

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

5.3 Header and footer

Header and footer are automatically excluded from tagging. This can be disabled with the following key. If some real content is in the header and footer, tagging must be restarted there explicitly. The key accepts the values `true` which surrounds the header with an artifact mc-chunk, `false` which disables the automatic tagging, and `pagination` which additionally adds an artifact structure with an pagination attribute.

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

5.4 Link tagging

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

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

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

6 User commands and extensions of document commands

```
1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-user} {2022-01-09} {0.93}
4 {tagpdf - user commands}
5 </header>
```

7 Setup and preamble commands

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

(End definition for \tagpdfsetup. This function is documented on page 29.)

8 Commands for the mc-chunks

```
\tagmcbegin
\tagmcend
\tagmcuse
11 \NewDocumentCommand \tagmcbegin { m }
12 {
13   \tag_mc_begin:n {#1}%\ignorespaces
14 }
15
16
17 \NewDocumentCommand \tagmcend { }
18 {
19   %\if_mode_horizontal: \unskip \fi: %
20   \tag_mc_end:
21 }
22
23 \NewDocumentCommand \tagmcuse { m }
24 {
25   \tag_mc_use:n {#1}
```



```

26   }
27

```

(End definition for `\tagmcbegin`, `\tagmcentd`, and `\tagmcuse`. These functions are documented on page 29.)

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

```

28 \NewDocumentCommand \tagmcifinTF { m m }
29 {
30   \tag_mc_if_in:TF { #1 } { #2 }
31 }

```

(End definition for `\tagmcifinTF`. This function is documented on page 29.)

9 Commands for the structure

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

`\tagstructend`
`\tagstructuse`

```

32 \NewDocumentCommand \tagstructbegin { m }
33 {
34   \tag_struct_begin:n {#1}
35 }
36
37 \NewDocumentCommand \tagstructend { }
38 {
39   \tag_struct_end:
40 }
41
42 \NewDocumentCommand \tagstructuse { m }
43 {
44   \tag_struct_use:n {#1}
45 }

```

(End definition for `\tagstructbegin`, `\tagstructend`, and `\tagstructuse`. These functions are documented on page 30.)

`\tagpdfifluatexTF` I should deprecate them ...

`\tagpdfifluatexT`
`\tagpdfifpdfTeXT`

```

46 \cs_set_eq:NN\tagpdfifluatexTF \sys_if_engine luatex:TF
47 \cs_set_eq:NN\tagpdfifluatexT  \sys_if_engine luatex:T
48 \cs_set_eq:NN\tagpdfifpdfTeXT  \sys_if_engine pdftex:T

```

(End definition for `\tagpdfifluatexTF`, `\tagpdfifluatexT`, and `\tagpdfifpdfTeXT`. These functions are documented on page 29.)

10 Debugging

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

```

49 \NewDocumentCommand\ShowTagging { m }
50 {
51   \keys_set:nn { __tag / show }{ #1}

```

```

52
53 }

```

(End definition for \ShowTagging. This function is documented on page 30.)

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

```

54 \keys_define:nn { __tag / show }
55 {
56   mc-data .code:n =
57   {
58     \sys_if_engine luatex:T
59     {
60       \lua_now:e{!tx.__tag.trace.show_all_mc_data(#1,\__tag_get_mc_abs_cnt:,0)}
61     }
62   }
63   ,mc-data .default:n = 1
64 }
65

```

(End definition for mc-data (show-key). This function is documented on page 30.)

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

```

66 \keys_define:nn { __tag / show }
67 { mc-current .code:n =
68   {
69     \bool_if:NTF \g__tag_mode_lua_bool
70     {
71       \sys_if_engine luatex:T
72       {
73         \int_compare:nNnTF
74         { -2147483647 }
75         =
76         {
77           \lua_now:e
78           {
79             tex.print
80             (tex.getattribute
81              (luatexbase.attributes.g__tag_mc_cnt_attr))
82           }
83         }
84         {
85           \lua_now:e
86           {
87             !tx.__tag.trace.log
88             (
89               "mc-current:~no~MC~open,~current~absent
90               =\__tag_get_mc_abs_cnt:"
91               ,0
92             )
93             texio.write_nl("")
94           }
95         }
96       }
97     }
98   }
99 }

```

```

96         {
97             \lua_now:e
98             {
99                 ltx.__tag.trace.log
100                 (
101                     "mc-current:~absent=~\__tag_get_mc_abs_cnt:=="
102                     ..
103                     tex.getattribute(luatexbase.attributes.g__tag_mc_cnt_attr)
104                     ..
105                     "~=>tag="
106                     ..
107                     tostring
108                     (ltx.__tag.func.get_tag_from
109                     (tex.getattribute
110                     (luatexbase.attributes.g__tag_mc_type_attr)))
111                     ..
112                     "="
113                     ..
114                     tex.getattribute
115                     (luatexbase.attributes.g__tag_mc_type_attr)
116                     ,0
117                 )
118                 texio.write_nl("")
119             }
120         }
121     }
122 }
123 {
124     \msg_note:nn{ tag }{ mc-current }
125 }
126 }
127 }

```

(End definition for `mc-current` (`show-key`). This function is documented on page 30.)

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

```

128 \keys_define:nn { __tag / show }
129 {
130     mc-marks .choice: ,
131     mc-marks / show .code:n =
132     {
133         \__tag_mc_get_marks:
134         \__tag_check_if_mc_in_galley:TF
135         {
136             \iow_term:n {Marks~from~this~page:~}
137         }
138         {
139             \iow_term:n {Marks~from~a~previous~page:~}
140         }
141         \seq_show:N \l__tag_mc_firstmarks_seq
142         \seq_show:N \l__tag_mc_botmarks_seq
143         \__tag_check_if_mc_tmb_missing:T
144         {

```

```

145         \iow_term:n {BDC~missing~on~this~page!}
146     }
147     \__tag_check_if_mc_tme_missing:T
148     {
149         \iow_term:n {EMC~missing~on~this~page!}
150     }
151 },
152 mc-marks / use .code:n =
153 {
154     \__tag_mc_get_marks:
155     \__tag_check_if_mc_in_galley:TF
156     { Marks~from~this~page::~}
157     { Marks~from~a~previous~page::~}
158     \seq_use:Nn \l__tag_mc_firstmarks_seq {,~}\quad
159     \seq_use:Nn \l__tag_mc_botmarks_seq {,~}\quad
160     \__tag_check_if_mc_tmb_missing:T
161     {
162         BDC~missing~
163     }
164     \__tag_check_if_mc_tme_missing:T
165     {
166         EMC~missing
167     }
168 },
169 mc-marks .default:n = show
170 }

```

(End definition for mc-marks (show-key). This function is documented on page 30.)

struct-stack_␣(show-key)

```

171 \keys_define:nn { __tag / show }
172 {
173     struct-stack .choice:
174     ,struct-stack / log .code:n = \seq_log:N \g__tag_struct_tag_stack_seq
175     ,struct-stack / show .code:n = \seq_show:N \g__tag_struct_tag_stack_seq
176     ,struct-stack .default:n = show
177 }

```

(End definition for struct-stack (show-key). This function is documented on page 30.)

11 Commands to extend document commands

The following commands and code parts are not core command of tagpdf. The 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.

11.1 Document structure

```

\__tag_add_document_structure:n
activate␣(setup-key) 178 \cs_new_protected:Npn \__tag_add_document_structure:n #1
179 {
180     \hook_gput_code:nnn{begindocument}{tagpdf}{\tagstructbegin{tag=#1}}

```

```

181 \hook_gput_code:nnn{tagpdf/finish/before}{tagpdf}{\tagstructend}
182 }
183 \keys_define:nn { __tag / setup}
184 {
185   activate .code:n =
186   {
187     \keys_set:nn { __tag / setup }
188     { activate-mc,activate-tree,activate-struct }
189     \__tag_add_document_structure:n {#1}
190   },
191   activate .default:n = Document
192 }

```

(End definition for `__tag_add_document_structure:n` and `activate (setup-key)`. This function is documented on page 29.)

11.2 Structure destinations

In TeXlive 2022 pdfTeX and LuaTeX will offer support for structure destinations. The pdfmanagement has already backend support. We activate them if the prerequisites are there: The pdf version should be 2.0, structures should be activated, the code in the pdfmanagement must be there.

```

193 \AddToHook{begindocument/before}
194 {
195   \bool_lazy_all:nT
196   {
197     { \g__tag_active_struct_dest_bool }
198     { \g__tag_active_struct_bool }
199     { \cs_if_exist_p:N \pdf_activate_structure_destination: }
200     { ! \pdf_version_compare_p:Nn < {2.0} }
201   }
202   {
203     \tl_set:Nn \l_pdf_current_structure_destination_tl { __tag/struct/\g__tag_struct_stack
204     \pdf_activate_structure_destination:
205   }
206 }

```

11.3 Fake space

`\pdffakespace` We need a LuaTeX variant for `\pdffakespace`. This should probably go into the kernel at some time.

```

207 \sys_if_engine_luaTeX:T
208 {
209   \NewDocumentCommand\pdffakespace { }
210   {
211     \__tag_fakespace:
212   }
213 }

```

(End definition for `\pdffakespace`. This function is documented on page 31.)

11.4 Paratagging

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

```
\l__tag_para_bool   At first some variables.
\l__tag_para_show_bool 214 \bool_new:N \l__tag_para_bool
\g__tag_para_int      215 \bool_new:N \l__tag_para_show_bool
                      216 \int_new:N \g__tag_para_begin_int
                      217 \int_new:N \g__tag_para_end_int
```

(End definition for `\l__tag_para_bool`, `\l__tag_para_show_bool`, and `\g__tag_para_int`.)

`paratagging_␣(setup-key)` These keys enable/disable locally paratagging, and the debug modus. It can affect the typesetting if `paratagging-show` is used. The small numbers are boxes and they have a (small) height.

```
218 \keys_define:nn { __tag / setup }
219 {
220   paratagging      .bool_set:N = \l__tag_para_bool,
221   paratagging-show .bool_set:N = \l__tag_para_show_bool,
222 }
223
```

(End definition for `paratagging (setup-key)` and `paratagging-show (setup-key)`. These functions are documented on page 31.)

This fills the para hooks with the needed code.

```
224 \AddToHook{para/begin}
225 {
226   \bool_if:NT \l__tag_para_bool
227   {
228     \int_gincr:N \g__tag_para_begin_int
229     \tag_struct_begin:n {tag=P}
230     \bool_if:NT \l__tag_para_show_bool
231     { \tag_mc_begin:n{artifact}
232       \llap{\color_select:n{red}\tiny\int_use:N\g__tag_para_begin_int\ }
233       \tag_mc_end:
234     }
235     \tag_mc_begin:n {tag=P}
236   }
237 }
238 \AddToHook{para/end}
239 {
240   \bool_if:NT \l__tag_para_bool
241   {
242     \int_gincr:N \g__tag_para_end_int
243     \tag_mc_end:
244     \bool_if:NT \l__tag_para_show_bool
245     { \tag_mc_begin:n{artifact}
246       \rlap{\color_select:n{red}\tiny\int_use:N\g__tag_para_end_int}
247       \tag_mc_end:
248     }
249     \tag_struct_end:
250   }
251 }
```

```

252 \AddToHook{enddocument/info}
253 {
254   \int_compare:nNnF {\g__tag_para_begin_int}={\g__tag_para_end_int}
255   {
256     \msg_error:nnxx
257       {tag}
258       {para-hook-count-wrong}
259       {\int_use:N\g__tag_para_begin_int}
260       {\int_use:N\g__tag_para_end_int}
261   }
262 }

```

In generic mode we need the additional code from the ptagging tests.

```

263 \AddToHook{begindocument/before}
264 {
265   \bool_if:NF \g__tag_mode_lua_bool
266   {
267     \cs_if_exist:NT \@kernel@before@footins
268     {
269       \tl_put_right:Nn \@kernel@before@footins
270       { \__tag_add_missing_mcs_to_stream:Nn \footins {footnote} }
271       \tl_put_right:Nn \@kernel@before@cclv
272       {
273         \__tag_check_typeout_v:n {====>~In~\token_to_str:N \@makecol\c_space_tl\the\c@p
274         \__tag_add_missing_mcs_to_stream:Nn \@cclv {main}
275       }
276       \tl_put_right:Nn \@mult@ptagging@hook
277       {
278         \__tag_check_typeout_v:n {====>~In~\string\page@sofar}
279         \process@cols\mult@firstbox
280         {
281           \__tag_add_missing_mcs_to_stream:Nn \count@ {multicol}
282         }
283         \__tag_add_missing_mcs_to_stream:Nn \mult@rightbox {multicol}
284       }
285     }
286   }
287 }

```

\tagpdfparaOn This two command switch para mode on and off. **\tagpdfsetup** could be used too but is longer.

```

288 \newcommand\tagpdfparaOn {\bool_set_true:N \l__tag_para_bool}
289 \newcommand\tagpdfparaOff{\bool_set_false:N \l__tag_para_bool}

```

(End definition for \tagpdfparaOn and \tagpdfparaOff. These functions are documented on page 31.)

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

```

\@hangfrom
{
  \tagstructbegin{tag=H1}%

```

```

\tagmcbegin      {tag=H1}%
#2
}
{#3\tagpdfsuppressmarks{\tagmcend}\tagstructend}%
290 \NewDocumentCommand\tagpdfsuppressmarks{m}
291   {{\use:c{__tag_mc_disable_marks:} #1}}

```

(End definition for \tagpdfsuppressmarks. This function is documented on page 31.)

11.5 Header and footer

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

```

292 \cs_new_protected:Npn\__tag_hook_kernel_before_head:{}
293 \cs_new_protected:Npn\__tag_hook_kernel_after_head:{}
294 \cs_new_protected:Npn\__tag_hook_kernel_before_foot:{}
295 \cs_new_protected:Npn\__tag_hook_kernel_after_foot:{}
296
297 \AddToHook{begindocument}
298 {
299   \cs_if_exist:NT \@kernel@before@head
300   {
301     \tl_put_right:Nn \@kernel@before@head {\__tag_hook_kernel_before_head:}
302     \tl_put_left:Nn \@kernel@after@head {\__tag_hook_kernel_after_head:}
303     \tl_put_right:Nn \@kernel@before@foot {\__tag_hook_kernel_before_foot:}
304     \tl_put_left:Nn \@kernel@after@foot {\__tag_hook_kernel_after_foot:}
305   }
306 }
307
308 \bool_new:N \g__tag_saved_in_mc_bool
309 \cs_new_protected:Npn \__tag_exclude_headfoot_begin:
310 {
311   \bool_set_false:N \l__tag_para_bool
312   \bool_if:NTF \g__tag_mode_lua_bool
313   {
314     \tag_mc_end_push:
315   }
316   {
317     \bool_gset_eq:NN \g__tag_saved_in_mc_bool \g__tag_in_mc_bool
318     \bool_gset_false:N \g__tag_in_mc_bool
319   }
320   \tag_mc_begin:n {artifact}
321 }
322 \cs_new_protected:Npn \__tag_exclude_headfoot_end:
323 {
324   \tag_mc_end:
325   \bool_if:NTF \g__tag_mode_lua_bool
326   {
327     \tag_mc_begin_pop:n{ }
328   }
329   {
330     \bool_gset_eq:NN \g__tag_in_mc_bool\g__tag_saved_in_mc_bool

```



```

331     }
332 }

This version allows to use an Artifact structure
333 \__tag_attr_new_entry:nn {__tag/attr/pagination}{/0/Artifact/Type/Pagination}
334 \cs_new_protected:Npn \__tag_exclude_struct_headfoot_begin:n #1
335 {
336     \bool_set_false:N \l__tag_para_bool
337     \bool_if:NTF \g__tag_mode_lua_bool
338     {
339         \tag_mc_end_push:
340     }
341     {
342         \bool_gset_eq:NN \g__tag_saved_in_mc_bool \g__tag_in_mc_bool
343         \bool_gset_false:N \g__tag_in_mc_bool
344     }
345     \tag_struct_begin:n{tag=Artifact,attribute-class=__tag/attr/#1}
346     \tag_mc_begin:n {artifact=#1}
347 }
348
349 \cs_new_protected:Npn \__tag_exclude_struct_headfoot_end:
350 {
351     \tag_mc_end:
352     \tag_struct_end:
353     \bool_if:NTF \g__tag_mode_lua_bool
354     {
355         \tag_mc_begin_pop:n{ }
356     }
357     {
358         \bool_gset_eq:NN \g__tag_in_mc_bool \g__tag_saved_in_mc_bool
359     }
360 }

```

And now the keys

`exclude-header-footerU(setup-key)`

```

361 \keys_define:nn { __tag / setup }
362 {
363     exclude-header-footer .choice:,
364     exclude-header-footer / true .code:n =
365     {
366         \cs_set_eq:NN \__tag_hook_kernel_before_head: \__tag_exclude_headfoot_begin:
367         \cs_set_eq:NN \__tag_hook_kernel_before_foot: \__tag_exclude_headfoot_begin:
368         \cs_set_eq:NN \__tag_hook_kernel_after_head: \__tag_exclude_headfoot_end:
369         \cs_set_eq:NN \__tag_hook_kernel_after_foot: \__tag_exclude_headfoot_end:
370     },
371     exclude-header-footer / pagination .code:n =
372     {
373         \cs_set:Nn \__tag_hook_kernel_before_head: { \__tag_exclude_struct_headfoot_begin:n {p
374         \cs_set:Nn \__tag_hook_kernel_before_foot: { \__tag_exclude_struct_headfoot_begin:n {p
375         \cs_set_eq:NN \__tag_hook_kernel_after_head: \__tag_exclude_struct_headfoot_end:
376         \cs_set_eq:NN \__tag_hook_kernel_after_foot: \__tag_exclude_struct_headfoot_end:
377     },
378     exclude-header-footer / false .code:n =
379     {

```

```

380     \cs_set_eq:NN \__tag_hook_kernel_before_head: \prg_do_nothing:
381     \cs_set_eq:NN \__tag_hook_kernel_before_foot: \prg_do_nothing:
382     \cs_set_eq:NN \__tag_hook_kernel_after_head: \prg_do_nothing:
383     \cs_set_eq:NN \__tag_hook_kernel_after_foot: \prg_do_nothing:
384   },
385   exclude-header-footer .default:n = true,
386   exclude-header-footer .initial:n = true
387 }

```

(End definition for `exclude-header-footer` (setup-key). This function is documented on page 31.)

11.6 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.

```

388 \hook_gput_code:nnn
389 {pdfannot/link/URI/before}
390 {tagpdf}
391 {
392   \tag_mc_end_push:
393   \tag_struct_begin:n { tag=Link }
394   \tag_mc_begin:n { tag=Link }
395   \pdfannot_dict_put:nnx
396     { link/URI }
397     { StructParent }
398     { \tag_struct_parent_int: }
399 }
400
401 \hook_gput_code:nnn
402 {pdfannot/link/URI/after}
403 {tagpdf}
404 {
405   \tag_struct_insert_annot:xx {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
406   \tag_mc_end:
407   \tag_struct_end:
408   \tag_mc_begin_pop:n{ }
409 }
410
411 \hook_gput_code:nnn
412 {pdfannot/link/GoTo/before}
413 {tagpdf}
414 {
415   \tag_mc_end_push:
416   \tag_struct_begin:n{tag=Link}
417   \tag_mc_begin:n{tag=Link}
418   \pdfannot_dict_put:nnx
419     { link/GoTo }
420     { StructParent }
421     { \tag_struct_parent_int: }
422 }
423
424 \hook_gput_code:nnn
425 {pdfannot/link/GoTo/after}

```

```

426 {tagpdf}
427 {
428   \tag_struct_insert_annot:xx {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
429   \tag_mc_end:
430   \tag_struct_end:
431   \tag_mc_begin_pop:n{ }
432
433 }
434
435 % "alternative descriptions " for PAX3. How to get better text here??
436 \pdfannot_dict_put:nnn
437 { link/URI }
438 { Contents }
439 { (url) }
440
441 \pdfannot_dict_put:nnn
442 { link/GoTo }
443 { Contents }
444 { (ref) }
445
</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} {2022-01-09} {0.93}
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 Catalog: MarkInfo and StructTreeRoot

The StructTreeRoot and the MarkInfo entry must be added to the catalog. We do it late so that we can win, but before the pdfmanagement hook.

```
--tag/struct/0 This is the object for the root object, the StructTreeRoot
20 \pdf_object_new:nn { __tag/struct/0 }{ dict }
(End definition for __tag/struct/0.)
21 \hook_gput_code:nnn{shipout/lastpage}{tagpdf}
22 {
23   \bool_if:NT \g__tag_active_tree_bool
24   {
25     \pdfmanagement_add:nnn { Catalog / MarkInfo } { Marked } { true }
26     \pdfmanagement_add:nnx
```

```

27         { Catalog }
28         { StructTreeRoot }
29         { \pdf_object_ref:n { __tag/struct/0 } }
30     }
31 }

```

1.2 Writing structure elements

The following commands are needed to write out the structure.

`__tag_tree_write_structtreeroot:` This writes out the root object.

```

32 \cs_new_protected:Npn \__tag_tree_write_structtreeroot:
33 {
34     \__tag_prop_gput:cnx
35     { g__tag_struct_0_prop }
36     { ParentTree }
37     { \pdf_object_ref:n { __tag/tree/parenttree } }
38     \__tag_prop_gput:cnx
39     { g__tag_struct_0_prop }
40     { RoleMap }
41     { \pdf_object_ref:n { __tag/tree/rolemap } }
42     \__tag_struct_write_obj:n { 0 }
43 }

```

(End definition for __tag_tree_write_structtreeroot:.)

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

```

44 \cs_new_protected:Npn \__tag_tree_write_structelements:
45 {
46     \int_step_inline:nnnn {1}{1}{\c@g__tag_struct_abs_int}
47     {
48         \__tag_struct_write_obj:n { ##1 }
49     }
50 }

```

(End definition for __tag_tree_write_structelements:.)

1.3 ParentTree

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

```

51 \pdf_object_new:nn { __tag/tree/parenttree }{ dict }

```

(End 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`.

```

52 \newcounter { g__tag_parenttree_obj_int }
53 \hook_gput_code:nnn{begindocument}{tagpdf}
54 {

```

```

55 \int_gset:Nn
56 \c@g__tag_parenttree_obj_int
57 { \__tag_ref_value_lastpage:nn{abspage}{100} }
58 }

```

(End 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

```

59 \tl_new:N \g__tag_parenttree_objr_tl

```

(End 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.

```

60 \cs_new_protected:Npn \__tag_parenttree_add_objr:nn #1 #2 %1 StructParent number, #2 objref
61 {
62   \tl_gput_right:Nx \g__tag_parenttree_objr_tl
63   {
64     #1 \c_space_tl #2 ^^J
65   }
66 }

```

(End definition for __tag_parenttree_add_objr:nn.)

\l__tag_parenttree_content_tl

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

```

67 \tl_new:N \l__tag_parenttree_content_tl

```

(End 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.

```

68
69 \cs_new_protected:Npn \__tag_tree_fill_parenttree:
70 {
71   \int_step_inline:nnnn{1}{1}{\__tag_ref_value_lastpage:nn{abspage}{-1}} %not quite clear if
72   { %page ##1
73     \prop_clear:N \l__tag_tmpa_prop
74     \int_step_inline:nnnn{1}{1}{\__tag_ref_value_lastpage:nn{tagmcabs}{-1}}
75     {
76       %mcid###1
77       \int_compare:nT
78       {\__tag_ref_value:enn{mcid-###1}{tagabspage}{-1}=##1} %mcid is on current page
79       {% yes
80         \prop_put:Nxx
81         \l__tag_tmpa_prop
82         {\__tag_ref_value:enn{mcid-###1}{tagmcid}{-1}}
83         {\prop_item:Nn \g__tag_mc_parenttree_prop {###1}}
84       }
85     }
86     \tl_put_right:Nx\l__tag_parenttree_content_tl
87     {
88       \int_eval:n {##1-1}\c_space_tl
89       [\c_space_tl %]

```

```

90     }
91     \int_step_inline:nnnn
92     {0}
93     {1}
94     { \prop_count:N \l__tag_tmpa_prop -1 }
95     {
96         \prop_get:NnNTF \l__tag_tmpa_prop {###1} \l__tag_tmpa_tl
97         {% page#1:mcid##1:\l__tag_tmpa_tl :content
98         \tl_put_right:Nx \l__tag_parenttree_content_tl
99         {
100             \pdf_object_if_exist:eT { __tag/struct/\l__tag_tmpa_tl }
101             {
102                 \pdf_object_ref:e { __tag/struct/\l__tag_tmpa_tl }
103             }
104             \c_space_tl
105         }
106     }
107     {
108         \msg_warning:nn { tag } {tree-mcid-index-wrong}
109     }
110 }
111 \tl_put_right:Nn
112 \l__tag_parenttree_content_tl
113 {%[
114 ]^^J
115 }
116 }
117 }

```

(End 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.

```

118 \cs_new_protected:Npn \__tag_tree_lua_fill_parenttree:
119 {
120     \tl_set:Nn \l__tag_parenttree_content_tl
121     {
122         \lua_now:e
123         {
124             ltx.__tag.func.output_parenttree
125             (
126                 \int_use:N\g_shipout_readonly_int
127             )
128         }
129     }
130 }

```

(End 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?

```

131 \cs_new_protected:Npn \__tag_tree_write_parenttree:
132 {
133     \bool_if:NTF \g__tag_mode_lua_bool
134     {

```

```

135     \__tag_tree_lua_fill_parenttree:
136   }
137   {
138     \__tag_tree_fill_parenttree:
139   }
140   \tl_put_right:NV \l__tag_parenttree_content_tl\g__tag_parenttree_objr_tl
141   \pdf_object_write:nx { __tag/tree/parenttree }
142   {
143     /Nums\c_space_tl [\l__tag_parenttree_content_tl]
144   }
145 }

```

(End definition for __tag_tree_write_parenttree:.)

1.4 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.
146 \pdf_object_new:nn { __tag/tree/rolemap }{ dict }

```

(End 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.
147 \cs_new_protected:Npn \__tag_tree_write_rolemap:
148   {
149     \pdf_object_write:nx { __tag/tree/rolemap }
150     {
151       \pdfdict_use:n{g__tag_role/RoleMap_dict}
152     }
153   }

```

(End definition for __tag_tree_write_rolemap:.)

1.5 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:
154 \cs_new_protected:Npn \__tag_tree_write_classmap:
155   {
156     \tl_clear:N \l__tag_tmpa_tl
157     \seq_gremove_duplicates:N \g__tag_attr_class_used_seq
158     \seq_set_map:NNn \l__tag_tmpa_seq \g__tag_attr_class_used_seq
159     {
160       /##1\c_space_tl
161       <<
162       \prop_item:Nn
163         \g__tag_attr_entries_prop
164         {##1}

```



```

165     >>
166   }
167   \tl_set:Nx \l__tag_tmpa_tl
168   {
169     \seq_use:Nn
170       \l__tag_tmpa_seq
171       { \iow_newline: }
172   }
173   \tl_if_empty:NF
174     \l__tag_tmpa_tl
175   {
176     \pdf_object_new:nn { __tag/tree/classmap }{ dict }
177     \pdf_object_write:nx
178       { __tag/tree/classmap }
179       { \l__tag_tmpa_tl }
180     \__tag_prop_gput:cnx
181       { g__tag_struct_0_prop }
182       { ClassMap }
183       { \pdf_object_ref:n { __tag/tree/classmap } }
184   }
185 }

```

(End definition for __tag_tree_write_classmap:.)

1.6 Namespaces

Namespaces are handle in the role module, here is the code to write them out. Namespaces are only relevant for pdf2.0 but we don't care, it doesn't harm.

__tag/tree/namespaces

```

186 \pdf_object_new:nn{ __tag/tree/namespaces }{array}

```

(End definition for __tag/tree/namespaces.)

__tag_tree_write_namespaces:

```

187 \cs_new_protected:Npn \__tag_tree_write_namespaces:
188 {
189   \prop_map_inline:Nn \g__tag_role_NS_prop
190   {
191     \pdfdict_if_empty:nF {g__tag_role/RoleMapNS_##1_dict}
192     {
193       \pdf_object_write:nx {__tag/RoleMapNS/##1}
194       {
195         \pdfdict_use:n {g__tag_role/RoleMapNS_##1_dict}
196       }
197       \pdfdict_gput:nnx{g__tag_role/RoleMapNS_##1_dict}
198       {RoleMapNS}{\pdf_object_ref:n {__tag/RoleMapNS/##1}}
199     }
200     \pdf_object_write:nx{tag/NS/##1}
201     {
202       \pdfdict_use:n {g__tag_role/RoleMapNS_##1_dict}
203     }
204   }
205   \pdf_object_write:nx {__tag/tree/namespaces}
206   {

```

```

207     \prop_map_tokens:Nn \g__tag_role_NS_prop{\use_i:nn}
208   }
209 }

```

(End definition for __tag_tree_write_namespaces:.)

1.7 Finishing the structure

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

__tag_finish_structure:

```

210 \cs_new_protected:Npn \__tag_finish_structure:
211 {
212   \bool_if:NT\g__tag_active_tree_bool
213   {
214     \hook_use:n {tagpdf/finish/before}
215     \__tag_tree_write_parenttree:
216     \__tag_tree_write_rolemap:
217     \__tag_tree_write_classmap:
218     \__tag_tree_write_namespaces:
219     \__tag_tree_write_structelements: %this is rather slow!!
220     \__tag_tree_write_structtreeroot:
221   }
222 }

```

(End definition for __tag_finish_structure:.)

1.8 StructParents entry for Page

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

```

223 \hook_gput_code:nnn{begindocument}{tagpdf}
224 {
225   \bool_if:NT\g__tag_active_tree_bool
226   {
227     \hook_gput_code:nnn{shipout/before} { tagpdf/structparents }
228     {
229       \pdfmanagement_add:nnx
230       { Page }
231       { StructParents }
232       { \int_eval:n { \g_shipout_readonly_int } }
233     }
234   }
235 }
236 </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{<key-values>}</code>
<code>\tag_mc_end:</code>	<code>\tag_mc_end:</code>

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

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

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

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

New: 2019-11-20

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

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

New: 2021-04-22

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

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

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.

`alttext_␣(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.

`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.

`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 splitted 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} {2022-01-09} {0.93}
4   {part of tagpdf - code related to marking chunks -
5     code shared by generic and luamode }
6 </header>
```

3.1 Variables and counters

MC chunks must be counted. I use a latex counter for the absolute count, so that it is added to `\cl@@ckpt` and restored e.g. in tabulars and align. `\int_new:N \c@g_@@_MCID_int` and `\tl_put_right:Nn\cl@@ckpt{\@elt{g_uf_test_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 (*shared)
8 \newcounter { g__tag_MCID_abs_int }
```

(End definition for g__tag_MCID_abs_int.)

`__tag_get_mc_abs_cnt:` A (expandable) function to get the current value of the cnt.

```
9 \cs_new:Npn __tag_get_mc_abs_cnt: { \int_use:N \c@g__tag_MCID_abs_int }
```

(End definition for __tag_get_mc_abs_cnt:.)

`g__tag_MCID_tmp_bypage_int`

The following hold the temporary by page number assigned to a mc. It must be defined in the shared code to avoid problems with labels.

```
10 \int_new:N g__tag_MCID_tmp_bypage_int
```

(End definition for g__tag_MCID_tmp_bypage_int.)

`g__tag_in_mc_bool`

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

```
11 \bool_new:N g__tag_in_mc_bool
```

(End 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

```
12 __tag_prop_new:N g__tag_mc_parenttree_prop
```

(End 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:

```
13 \seq_new:N g__tag_mc_stack_seq
```

(End 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.

```
14 \tl_new:N l__tag_mc_artifact_type_tl
```

(End 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`

```
15 \bool_new:N l__tag_mc_key_stash_bool
```

```
16 \bool_new:N l__tag_mc_artifact_bool
```

(End 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` 17 `\tl_new:N \l__tag_mc_key_tag_tl`

`\l__tag_mc_key_properties_tl` 18 `\tl_new:N \g__tag_mc_key_tag_tl`

19 `\tl_new:N \l__tag_mc_key_label_tl`

20 `\tl_new:N \l__tag_mc_key_properties_tl`

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

3.2 Functions

`__tag_mc_handle_mc_label:n` 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

`tagabspace`: the absolute page, `\g_shipout_readonly_int`,

`tagmcabs`: the absolute mc-counter `\c@g_@@MCID_abs_int`,

`tagmcid`: the ID of the chunk on the page `\g_@@MCID_tmp_bypage_int`, this typically settles down after a second compilation. The reference command is defined in `tagpdf.dtx` and is based on `l3ref`.

21 `\cs_new:Nn __tag_mc_handle_mc_label:n`

22 `{`

23 `__tag_ref_label:en{tagpdf-#1}{mc}`

24 `}`

(End definition for `__tag_mc_handle_mc_label:n`.)

`__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

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

26 `{`

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

28 `}`

(End 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?

29 `\cs_new_protected:Npn \tag_mc_use:n #1 %#1: label name`

30 `{`

31 `__tag_check_if_active_struct:T`

32 `{`

33 `\tl_set:Nx \l__tag_tmpa_tl { __tag_ref_value:nnn{tagpdf-#1}{tagmcabs}{ } }`

34 `\tl_if_empty:NTF\l__tag_tmpa_tl`

35 `{`

36 `\msg_warning:nnn {tag} {mc-label-unknown} {#1}`

37 `}`

38 `{`

39 `\cs_if_free:cTF { g__tag_mc_label_\tl_to_str:n{#1}_used_tl }`

40 `{`

41 `__tag_mc_handle_stash:x { \l__tag_tmpa_tl }`

```

42         \__tag_mc_set_label_used:n {#1}
43     }
44     {
45         \msg_warning:nnn {tag}{mc-used-twice}{#1}
46     }
47 }
48 }
49 }

```

(End definition for `\tag_mc_use:n`. This function is documented on page 51.)

`\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.

```

50 \cs_new_protected:Npn \tag_mc_artifact_group_begin:n #1
51 {
52     \tag_mc_end_push:
53     \tag_mc_begin:n {artifact=#1}
54     \tag_stop_group_begin:
55 }
56
57 \cs_new_protected:Npn \tag_mc_artifact_group_end:
58 {
59     \tag_stop_group_end:
60     \tag_mc_end:
61     \tag_mc_begin_pop:n{ }
62 }

```

(End definition for `\tag_mc_artifact_group_begin:n` and `\tag_mc_artifact_group_end:`. These functions are documented on page 51.)

`\tag_mc_end_push:`
`\tag_mc_begin_pop:n`

```

63 \cs_new_protected:Npn \tag_mc_end_push:
64 {
65     \__tag_check_if_active_mc:T
66     {
67         \__tag_mc_if_in:TF
68         {
69             \seq_gpush:Nx \g__tag_mc_stack_seq { \tag_get:n {mc_tag} }
70             \__tag_check_mc_pushed_popped:nn
71             { pushed }
72             { \tag_get:n {mc_tag} }
73             \tag_mc_end:
74         }
75         {
76             \seq_gpush:Nn \g__tag_mc_stack_seq {-1}
77             \__tag_check_mc_pushed_popped:nn { pushed }{-1}
78         }
79     }
80 }
81
82 \cs_new_protected:Npn \tag_mc_begin_pop:n #1
83 {
84     \__tag_check_if_active_mc:T
85     {
86         \seq_gpop:NNTF \g__tag_mc_stack_seq \l__tag_tmpa_tl

```

```

87     {
88         \tl_if_eq:NnTF \l__tag_tmpa_tl {-1}
89         {
90             \__tag_check_mc_pushed_popped:nn {popped}{-1}
91         }
92         {
93             \__tag_check_mc_pushed_popped:nn {popped}{\l__tag_tmpa_tl}
94             \tag_mc_begin:n {tag=\l__tag_tmpa_tl,#1}
95         }
96     }
97     {
98         \__tag_check_mc_pushed_popped:nn {popped}{empty~stack,~nothing}
99     }
100 }
101 }

```

(End definition for `\tag_mc_end_push:` and `\tag_mc_begin_pop:n`. These functions are documented on page 51.)

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.

```

102 \keys_define:nn { __tag / mc }
103 {
104     stash .bool_set:N = \l__tag_mc_key_stash_bool,
105     __artifact-bool .bool_set:N = \l__tag_mc_artifact_bool,
106     __artifact-type .choice:,
107     __artifact-type / pagination .code:n =
108     {
109         \tl_set:Nn \l__tag_mc_artifact_type_tl { Pagination }
110     },
111     __artifact-type / pagination/header .code:n =
112     {
113         \tl_set:Nn \l__tag_mc_artifact_type_tl { Pagination/Subtype/Header }
114     },
115     __artifact-type / pagination/footer .code:n =
116     {
117         \tl_set:Nn \l__tag_mc_artifact_type_tl { Pagination/Subtype/Footer }
118     },
119     __artifact-type / layout .code:n =
120     {
121         \tl_set:Nn \l__tag_mc_artifact_type_tl { Layout }
122     },
123     __artifact-type / page .code:n =
124     {
125         \tl_set:Nn \l__tag_mc_artifact_type_tl { Page }
126     },
127     __artifact-type / background .code:n =
128     {

```



```

129     \tl_set:Nn \l__tag_mc_artifact_type_tl { Background }
130   },
131   __artifact-type / notype .code:n =
132   {
133     \tl_set:Nn \l__tag_mc_artifact_type_tl {}
134   },
135   __artifact-type / .code:n =
136   {
137     \tl_set:Nn \l__tag_mc_artifact_type_tl {}
138   },
139 }

```

(End definition for stash (mc-key), __artifact-bool, and __artifact-type. This function is documented on page 52.)

```

140 </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} {2022-01-09} {0.93}
4 {part of tagpdf - code related to marking chunks - generic mode}
5 </generic>
6 <*debug>
7 \ProvidesExplPackage {tagpdf-debug-generic} {2022-01-09} {0.93}
8 {part of tagpdf - debugging code related to marking chunks - generic mode}
9 </debug>
```

1.1 Variables

`\g__tag_MCID_byabspage_prop` This property will hold the current maximum on a page it will contain key-value of type $\langle abspagenum \rangle = \langle max\ mcid \rangle$

```
10 <*generic>
11 \__tag_prop_new:N \g__tag_MCID_byabspage_prop
```

(End definition for `\g__tag_MCID_byabspage_prop`.)

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

```
12 \tl_new:N \l__tag_mc_ref_abspage_tl
```

(End definition for `\l__tag_mc_ref_abspage_tl`.)

`\l__tag_mc_tmpa_tl` temporary variable

```
13 \tl_new:N \l__tag_mc_tmpa_tl
```

(End 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.

```
14 \newmarks \g__tag_mc_marks
```

(End definition for `\g__tag_mc_marks`.)

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

```
15 \seq_new:N \g__tag_mc_main_marks_seq
16 \seq_new:N \g__tag_mc_footnote_marks_seq
17 \seq_new:N \g__tag_mc_multicol_marks_seq
```

(End 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.
`\l__tag_mc_botmarks_seq`

```
18 \seq_new:N \l__tag_mc_firstmarks_seq
19 \seq_new:N \l__tag_mc_botmarks_seq
```

(End 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).

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

```

49     #1 %type
50   }
51   \tex_marks:D \g__tag_mc_marks
52   {
53     b+, %first of begin pair
54     \int_use:N\c@g__tag_MCID_abs_int, %mc-num
55     -1, %structure num
56     #1 %Type
57   }
58 }
59
60 \cs_new_protected:Npn \__tag_mc_end_marks:
61 {
62   \tex_marks:D \g__tag_mc_marks
63   {
64     e-, %first of end pair
65     \int_use:N\c@g__tag_MCID_abs_int, %mc-num
66     \g__tag_struct_stack_current_tl, %structure num
67   }
68   \tex_marks:D \g__tag_mc_marks
69   {
70     e+, %second of end pair
71     \int_use:N\c@g__tag_MCID_abs_int, %mc-num
72     \g__tag_struct_stack_current_tl, %structure num
73   }
74 }

```

(End 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.

```

75 \cs_new_protected:Npn \__tag_mc_disable_marks:
76 {
77   \cs_set_eq:NN \__tag_mc_begin_marks:nn \use_none:nn
78   \cs_set_eq:NN \__tag_mc_artifact_begin_marks:n \use_none:n
79   \cs_set_eq:NN \__tag_mc_end_marks: \prg_do_nothing:
80 }

```

(End 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.

```

81 \cs_new_protected:Npn \__tag_mc_get_marks:
82 {
83   \exp_args:NNx
84   \seq_set_from_clist:Nn \l__tag_mc_firstmarks_seq
85   { \tex_firstmarks:D \g__tag_mc_marks }
86   \exp_args:NNx
87   \seq_set_from_clist:Nn \l__tag_mc_botmarks_seq
88   { \tex_botmarks:D \g__tag_mc_marks }
89 }

```

(End 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.

```

90 \cs_new_protected:Npn \__tag_mc_store:nnn #1 #2 #3 % #1 mc-prev, #2 mc-num #3 structure-
    num
91 {
92   %\prop_show:N \g__tag_struct_cont_mc_prop
93   \prop_get:NnTF \g__tag_struct_cont_mc_prop {#1} \l__tag_tmpa_tl
94   {
95     \prop_gput:Nnx \g__tag_struct_cont_mc_prop {#1}{ \l__tag_tmpa_tl \__tag_struct_mcid_d
96   }
97   {
98     \prop_gput:Nnx \g__tag_struct_cont_mc_prop {#1}{ \__tag_struct_mcid_dict:n {#2}}
99   }
100   \prop_gput:Nxx \g__tag_mc_parenttree_prop
101   {#2}
102   {#3}
103 }
104 \cs_generate_variant:Nn \__tag_mc_store:nnn {xxx}

```

(End definition for `__tag_mc_store:nnn`.)

`__tag_mc_insert_extra_tmb:n` `__tag_mc_insert_extra_tme: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.

```

105 \cs_new_protected:Npn \__tag_mc_insert_extra_tmb:n #1 % #1 stream: e.g. main or footnote
106 {
107   \__tag_check_typeout_v:n {>~ first~ \seq_use:Nn \l__tag_mc_firstmarks_seq {,~}}
108   \__tag_check_typeout_v:n {>~ bot~ \seq_use:Nn \l__tag_mc_botmarks_seq {,~}}
109   \__tag_check_if_mc_tmb_missing:TF
110   {
111     \__tag_check_typeout_v:n {>~ TMB~ ~ missing~ --- inserted}
112     %test if artifact
113     \int_compare:nNnTF { \seq_item:cn { g__tag_mc_#1_marks_seq } {3} } = {-
114       1}
115       {
116         \tl_set:Nx \l__tag_tmpa_tl { \seq_item:cn { g__tag_mc_#1_marks_seq } {4} }
117         \__tag_mc_handle_artifact:N \l__tag_tmpa_tl
118       }
119       {
120         \exp_args:Nx
121         \__tag_mc_bdc_mcid:n
122         {
123           \seq_item:cn { g__tag_mc_#1_marks_seq } {4}
124         }
125         \str_if_eq:eeTF
126         {

```

```

126         \seq_item:cn { g__tag_mc_#1_marks_seq } {5}
127     }
128     {}
129     {
130         %store
131         \__tag_mc_store:xxx
132         {
133             \seq_item:cn { g__tag_mc_#1_marks_seq } {2}
134         }
135         { \int_eval:n{\c@g__tag_MCID_abs_int} }
136         {
137             \seq_item:cn { g__tag_mc_#1_marks_seq } {3}
138         }
139     }
140     {
141         %stashed -> warning!!
142     }
143 }
144 }
145 {
146     \__tag_check_typeout_v:n {=>~ TMB~ not~ missing}
147 }
148 }
149
150 \cs_new_protected:Npn \__tag_mc_insert_extra_tme:n #1 % #1 stream, eg. main or footnote
151 {
152     \__tag_check_if_mc_tme_missing:TF
153     {
154         \__tag_check_typeout_v:n {=>~ TME~ ~ missing~ --- inserted}
155         \__tag_mc_emc:
156         \seq_gset_eq:cN
157         { g__tag_mc_#1_marks_seq }
158         \l__tag_mc_botmarks_seq
159     }
160     {
161         \__tag_check_typeout_v:n {=>~ TME~ not~ missing}
162     }
163 }

```

(End 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.

```

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

```

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

```

174   \box_set_ht:Nn \l__tag_tmpa_box \c_zero_dim
175   \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.

```

176   \box_set_ht:Nn \l__tag_tmpb_box \c_zero_dim
177   \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.

```

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

```

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

```

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

```

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

```

182   \nointerlineskip
183   \box_use_drop:N \l__tag_tmpb_box
184 }
185 }

```

(End 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 therefor 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.

```

186 \cs_new_protected:Npn \__tag_add_missing_mcs_to_stream:Nn #1#2
187 {
188   \__tag_check_if_active_mc:T {
189     First set up a temp box for trial splitting.
190     \vbadness\maxdimen
191     \box_set_eq:NN \l__tag_tmpa_box #1

```

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

```
191 \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`

```
192 \exp_args:NNx
193 \seq_set_from_clist:Nn \l__tag_mc_firstmarks_seq
194 { \tex_splitfirstmarks:D \g__tag_mc_marks }
```

Some debugging info:

```
195 % \iow_term:n { First~ mark~ from~ this~ box: }
196 % \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!

```
197 \seq_if_empty:NTF \l__tag_mc_firstmarks_seq
198 {
199     \__tag_check_typeout_v:n
200     {
201         No~ marks~ so~ use~ saved~ bot~ mark:~
202         \seq_use:cn {g__tag_mc_#2_marks_seq} {,~} \iow_newline:
203     }
204     \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).

```
205 \seq_set_eq:NN \l__tag_mc_botmarks_seq \l__tag_mc_firstmarks_seq
206 }
```

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.

```
207 {
208     \__tag_check_typeout_v:n
209     {
210         Pick~ up~ new~ bot~ mark!
211     }
212     \exp_args:NNx
213     \seq_set_from_clist:Nn \l__tag_mc_botmarks_seq
214     { \tex_splitbotmarks:D \g__tag_mc_marks }
215 }
```

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

```
216 \__tag_add_missing_mcs:Nn #1 {#2}
217 %%
218 \seq_gset_eq:cN {g__tag_mc_#2_marks_seq} \l__tag_mc_botmarks_seq
219 %%
220 }
221 }
```

(End 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:` 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.
`\tag_mc_if_in:TF`

```

222 \prg_new_conditional:Nnn \_tag_mc_if_in: {p,T,F,TF}
223 {
224   \bool_if:NTF \g__tag_in_mc_bool
225     { \prg_return_true: }
226     { \prg_return_false: }
227 }

```

```

228
229 \prg_new_eq_conditional:NNn \tag_mc_if_in: \_tag_mc_if_in: {p,T,F,TF}

```

(End definition for `_tag_mc_if_in:TF` and `\tag_mc_if_in:TF`. This function is documented on page 51.)

`_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 escape them, this should be done before using them.
`_tag_mc_bdc:nn`
`_tag_mc_bdc:nx`

```

230 % #1 tag, #2 properties
231 \cs_set_eq:NN \_tag_mc_bmc:n \pdf_bmc:n
232 \cs_set_eq:NN \_tag_mc_emc: \pdf_emc:
233 \cs_set_eq:NN \_tag_mc_bdc:nn \pdf_bdc:nn
234 \cs_generate_variant:Nn \_tag_mc_bdc:nn {nx}

```

(End 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 number value for the MCID: they must be numbered by page starting with 0 and then successively. The first argument is the tag, e.g. P or Span, the second is used to pass more properties. We also define a wrapper around the low-level command as luamode will need something different.
`_tag_mc_bdc_mcid:n`
`_tag_mc_handle_mcid:nn`
`_tag_mc_handle_mcid:VV`

```

235 \cs_new_protected:Npn \_tag_mc_bdc_mcid:nn #1 #2
236 {
237   \int_gincr:N \c@g__tag_MCID_abs_int
238   \tl_set:Nx \l__tag_mc_ref_abbrev_t1
239     {
240       \_tag_ref_value:enn %3 args
241       {
242         mcid-\int_use:N \c@g__tag_MCID_abs_int
243       }
244       { tagabbrev }
245       {-1}
246     }
247   \prop_get:NoNTF
248     \g__tag_MCID_byabbrev_prop
249     {
250       \l__tag_mc_ref_abbrev_t1

```

```

251     }
252     \l__tag_mc_tmpa_tl
253     {
254         %key already present, use value for MCID and add 1 for the next
255         \int_gset:Nn \g__tag_MCID_tmp_bypage_int { \l__tag_mc_tmpa_tl }
256         \__tag_prop_gput:Nxx
257             \g__tag_MCID_byabspage_prop
258             { \l__tag_mc_ref_abspage_tl }
259             { \int_eval:n {\l__tag_mc_tmpa_tl +1} }
260     }
261     {
262         %key not present, set MCID to 0 and insert 1
263         \int_gzero:N \g__tag_MCID_tmp_bypage_int
264         \__tag_prop_gput:Nxx
265             \g__tag_MCID_byabspage_prop
266             { \l__tag_mc_ref_abspage_tl }
267             {1}
268     }
269     \__tag_ref_label:en
270     {
271         mcid~\int_use:N \c@g__tag_MCID_abs_int
272     }
273     { mc }
274     \__tag_mc_bdc:nx
275     {#1}
276     { /MCID~\int_eval:n { \g__tag_MCID_tmp_bypage_int }~ \exp_not:n { #2 } }
277 }
278 \cs_new_protected:Npn \__tag_mc_bdc_mcid:n #1
279 {
280     \__tag_mc_bdc_mcid:nn {#1} {}
281 }
282
283 \cs_new_protected:Npn \__tag_mc_handle_mcid:nn #1 #2 %1 tag, #2 properties
284 {
285     \__tag_mc_bdc_mcid:nn {#1} {#2}
286 }
287
288 \cs_generate_variant:Nn \__tag_mc_handle_mcid:nn {VV}

```

(End 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?

```

289 \cs_new_protected:Npn \__tag_mc_handle_stash:n #1 %1 mcidnum
290 {
291     \__tag_check_mc_used:n {#1}
292     \__tag_struct_kid_mc_gput_right:nn
293         { \g__tag_struct_stack_current_tl }
294         {#1}
295     \prop_gput:Nxx \g__tag_mc_parenttree_prop
296         {#1}

```

```

297     { \g__tag_struct_stack_current_tl }
298   }
299   \cs_generate_variant:Nn \__tag_mc_handle_stash:n { x }

```

(End 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

```

300 \cs_new_protected:Npn \__tag_mc_bmc_artifact:
301   {
302     \__tag_mc_bmc:n {Artifact}
303   }
304 \cs_new_protected:Npn \__tag_mc_bmc_artifact:n #1
305   {
306     \__tag_mc_bdc:nn {Artifact}{/Type/#1}
307   }
308 \cs_new_protected:Npn \__tag_mc_handle_artifact:N #1
309   % #1 is a var containing the artifact type
310   {
311     \int_gincr:N \c@g__tag_MCID_abs_int
312     \tl_if_empty:NTF #1
313       { \__tag_mc_bmc_artifact: }
314       { \exp_args:NV\__tag_mc_bmc_artifact:n #1 }
315   }

```

(End 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.

```

316 \cs_new:Nn \__tag_get_data_mc_tag: { \g__tag_mc_key_tag_tl }
317 </generic>

```

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

\tag_mc_end:

```

318 <*generic | debug>
319 <*generic>
320 \cs_new_protected:Npn \tag_mc_begin:n #1 %#1 keyval
321   {
322     \__tag_check_if_active_mc:T
323     {
324       </generic>
325       <*debug>
326       \cs_set_protected:Npn \tag_mc_begin:n #1 %#1 keyval
327         {
328           \__tag_check_if_active_mc:TF
329           {
330             \__tag_debug_mc_begin_insert:n { #1 }
331           } </debug>
332           \group_begin: %hm

```

```

333     \__tag_check_mc_if_nested:
334     \bool_gset_true:N \g__tag_in_mc_bool
335     \keys_set:nn { __tag / mc } {#1}
336     \bool_if:NTF \l__tag_mc_artifact_bool
337     { %handle artifact
338       \__tag_mc_handle_artifact:N \l__tag_mc_artifact_type_tl
339       \exp_args:NV
340       \__tag_mc_artifact_begin_marks:n \l__tag_mc_artifact_type_tl
341     }
342     { %handle mcid type
343       \__tag_check_mc_tag:N \l__tag_mc_key_tag_tl
344       \__tag_mc_handle_mcid:VV
345       \l__tag_mc_key_tag_tl
346       \l__tag_mc_key_properties_tl
347       \__tag_mc_begin_marks:oo{\l__tag_mc_key_tag_tl}{\l__tag_mc_key_label_tl}
348       \tl_if_empty:NF {\l__tag_mc_key_label_tl}
349       {
350         \exp_args:NV
351         \__tag_mc_handle_mc_label:n \l__tag_mc_key_label_tl
352       }
353       \bool_if:NF \l__tag_mc_key_stash_bool
354       {
355         \__tag_mc_handle_stash:x { \int_use:N \c@g__tag_MCID_abs_int }
356       }
357     }
358   \group_end:
359 }
360 <debug>
361 {
362   \__tag_debug_mc_begin_ignore:n { #1 }
363 }
364 </debug>
365 }
366 <*generic>
367 \cs_new_protected:Nn \tag_mc_end:
368 {
369   \__tag_check_if_active_mc:T
370   {
371 </generic>
372 <*debug>
373 \cs_set_protected:Nn \tag_mc_end:
374 {
375   \__tag_check_if_active_mc:TF
376   {
377     \__tag_debug_mc_end_insert:
378 </debug>
379     \__tag_check_mc_if_open:
380     \bool_gset_false:N \g__tag_in_mc_bool
381     \tl_gset:Nn \g__tag_mc_key_tag_tl { }
382     \__tag_mc_emc:
383     \__tag_mc_end_marks:
384   }
385 <*debug>
386   {

```

```

387     \__tag_debug_mc_end_ignore:
388 }
389 </debug>
390 }
391 </generic | debug>

```

(End definition for \tag_mc_begin:n and \tag_mc_end:. These functions are documented on page 51.)

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) 392 <*generic>
alttext_(mc-key) 393 \keys_define:nn { __tag / mc }
actualtext_(mc-key) 394 {
    label_(mc-key) 395   tag .code:n = % the name (H,P,Span) etc
    artifact_(mc-key) 396   {
397       \tl_set:Nx \l__tag_mc_key_tag_tl { #1 }
398       \tl_gset:Nx \g__tag_mc_key_tag_tl { #1 }
399   },
400   raw .code:n =
401   {
402       \tl_put_right:Nx \l__tag_mc_key_properties_tl { #1 }
403   },
404   alttext .code:n = % Alt property
405   {
406       \str_set_convert:Noon
407       \l__tag_tmpa_str
408       { #1 }
409       { default }
410       { utf16/hex }
411       \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
412       \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
413   },
414   actualtext .code:n = % ActualText property
415   {
416       \str_set_convert:Noon
417       \l__tag_tmpa_str
418       { #1 }
419       { default }
420       { utf16/hex }
421       \tl_put_right:Nn \l__tag_mc_key_properties_tl { /ActualText~< }
422       \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
423   },
424   label .tl_set:N = \l__tag_mc_key_label_tl,
425   artifact .code:n =
426   {
427       \exp_args:Nnx
428       \keys_set:nn
429       { __tag / mc }
430       { __artifact-bool, __artifact-type=#1 }
431   },

```

```
432     artifact .default:n    = {notype}  
433   }  
434 </generic>
```

(End definition for tag `mc-key` and others. These functions are documented on page 52.)

Part VI

The **tagpdf-mc-luacode** module

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

Part of the tagpdf package

The code is splitted 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 ... \tagmccend` 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} {2022-01-09} {0.93}
4   {tagpdf - mc code only for the luamode }
5 </luamode>
```

The main function which wanders through the shipout box to inject the literals. if the new callback is there, it is used.

```
6 <*luamode>
7 \hook_gput_code:nnn{begindocument}{tagpdf/mc}
8 {
```

```

9      \bool_if:NT\g__tag_active_space_bool
10      {
11          \lua_now:e
12          {
13              if~luatexbase.callbacktypes.pre_shipout_filter~then~
14                  luatexbase.add_to_callback("pre_shipout_filter", function(TAGBOX)~
15                      ltx.__tag.func.space_chars_shipout(TAGBOX)~return~true~
16                      end, "tagpdf")~
17              end
18          }
19      \lua_now:e
20      {
21          if~luatexbase.callbacktypes.pre_shipout_filter~then~
22              token.get_next()~
23              end
24          }\@secondoftwo\@gobble
25          {
26              \hook_gput_code:nnn{shipout/before}{tagpdf/lua}
27              {
28                  \lua_now:e
29                  { ltx.__tag.func.space_chars_shipout (tex.box["ShipoutBox"]) }
30              }
31          }
32      }
33      \bool_if:NT\g__tag_active_mc_bool
34      {
35          \lua_now:e
36          {
37              if~luatexbase.callbacktypes.pre_shipout_filter~then~
38                  luatexbase.add_to_callback("pre_shipout_filter", function(TAGBOX)~
39                      ltx.__tag.func.mark_shipout(TAGBOX)~return~true~
40                      end, "tagpdf")~
41              end
42          }
43      \lua_now:e
44      {
45          if~luatexbase.callbacktypes.pre_shipout_filter~then~
46              token.get_next()~
47              end
48          }\@secondoftwo\@gobble
49          {
50              \hook_gput_code:nnn{shipout/before}{tagpdf/lua}
51              {
52                  \lua_now:e
53                  { ltx.__tag.func.mark_shipout (tex.box["ShipoutBox"]) }
54              }
55          }
56      }
57  }

```

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.


```
58 \cs_new_protected:Npn \__tag_add_missing_mcs_to_stream:Nn #1#2 {}
```

(End 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.

```
\__tag_mc_if_in:TF 59 \prg_new_conditional:Nnn \__tag_mc_if_in: {p,T,F,TF}
\tag_mc_if_in_p: 60 {
\tag_mc_if_in:TF 61 \int_compare:nNnTF
62 { -2147483647 }
63 =
64 {\lua_now:e
65 {
66 tex.print(tex.getattribute(luatexbase.attributes.g__tag_mc_type_attr))
67 }
68 }
69 { \prg_return_false: }
70 { \prg_return_true: }
71 }
72
73 \prg_new_eq_conditional:Nnn \tag_mc_if_in: \__tag_mc_if_in: {p,T,F,TF}
```

(End definition for __tag_mc_if_in:TF and \tag_mc_if_in:TF. This function is documented on page 51.)

_tag_mc_lua_set_mc_type_attr:n This takes a tag name, and sets the attributes to the related number. It is not decided yet if this will be global or local, see the global-mc option.

```
\_tag_mc_lua_set_mc_type_attr:o
\_tag_mc_lua_unset_mc_type_attr: 74 \cs_new:Nn \__tag_mc_lua_set_mc_type_attr:n % #1 is a tag name
75 {
76 %TODO ltx.__tag.func.get_num_from("#1") seems not to return a suitable number??
77 \tl_set:Nx\l__tag_tmpa_tl{\lua_now:e{ltx.__tag.func.output_num_from ("#1")}}
78 \lua_now:e
79 {
80 tex.setattribute
81 (
82 "global",
83 luatexbase.attributes.g__tag_mc_type_attr,
84 \l__tag_tmpa_tl
85 )
86 }
87 \lua_now:e
88 {
89 tex.setattribute
90 (
91 "global",
92 luatexbase.attributes.g__tag_mc_cnt_attr,
93 \__tag_get_mc_abs_cnt:
94 )
95 }
96 }
97
98 \cs_generate_variant:Nn\__tag_mc_lua_set_mc_type_attr:n { o }
99
100 \cs_new:Nn \__tag_mc_lua_unset_mc_type_attr:
101 {
102 \lua_now:e
```

```

103     {
104         tex.setattribute
105         (
106             "global",
107             luatexbase.attributes.g__tag_mc_type_attr,
108             -2147483647
109         )
110     }
111 \lua_now:e
112 {
113     tex.setattribute
114     (
115         "global",
116         luatexbase.attributes.g__tag_mc_cnt_attr,
117         -2147483647
118     )
119 }
120 }
121

```

(End 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

```

122 \cs_new:Nn \_tag_mc_insert_mcid_kids:n
123 {
124     \lua_now:e { ltx.__tag.func.mc_insert_kids (#1,0) }
125 }
126
127 \cs_new:Nn \_tag_mc_insert_mcid_single_kids:n
128 {
129     \lua_now:e {ltx.__tag.func.mc_insert_kids (#1,1) }
130 }

```

(End 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:x` structure.

```

131 \cs_new:Nn \_tag_mc_handle_stash:n %1 mcidnum
132 {
133     \_tag_check_mc_used:n { #1 }
134     \seq_gput_right:cn % Don't fill a lua table due to the command in the item,
135                       % so use the kernel command
136     { g__tag_struct_kids\_g__tag_struct_stack_current_tl _seq }
137     {
138         \_tag_mc_insert_mcid_kids:n {#1}%
139     }
140     \lua_now:e
141     {
142         ltx.__tag.func.store_struct_mcabs
143         (
144             \g__tag_struct_stack_current_tl,#1
145         )
146     }

```

```

147     \prop_gput:Nxx
148     \g__tag_mc_parenttree_prop
149     { #1 }
150     { \g__tag_struct_stack_current_tl }
151   }
152
153   \cs_generate_variant:Nn \__tag_mc_handle_stash:n { x }

(End 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.

```

154   \cs_new_protected:Nn \tag_mc_begin:n
155   {
156     \__tag_check_if_active_mc:T
157     {
158       \group_begin:
159       %\__tag_check_mc_if_nested:
160       \bool_gset_true:N \g__tag_in_mc_bool
161       \bool_set_false:N \l__tag_mc_artifact_bool
162       \tl_clear:N \l__tag_mc_key_properties_tl
163       \int_gincr:N \c@g__tag_MCID_abs_int
164       \keys_set:nn { __tag / mc }{ label={}, #1 }
165       %check that a tag or artifact has been used
166       \__tag_check_mc_tag:N \l__tag_mc_key_tag_tl
167       %set the attributes:
168       \__tag_mc_lua_set_mc_type_attr:o { \l__tag_mc_key_tag_tl }
169       \bool_if:NF \l__tag_mc_artifact_bool
170       { % store the absolute num name in a label:
171         \tl_if_empty:NF {\l__tag_mc_key_label_tl}
172         {
173           \exp_args:NV
174           \__tag_mc_handle_mc_label:n \l__tag_mc_key_label_tl
175         }
176         % if not stashed record the absolute number
177         \bool_if:NF \l__tag_mc_key_stash_bool
178         {
179           \__tag_mc_handle_stash:x { \__tag_get_mc_abs_cnt: }
180         }
181       }
182       \group_end:
183     }
184   }

```

(End definition for \tag_mc_begin:n. This function is documented on page 51.)

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

```

185   \cs_new_protected:Nn \tag_mc_end:
186   {
187     \__tag_check_if_active_mc:T
188     {
189       %\__tag_check_mc_if_open:
190       \bool_gset_false:N \g__tag_in_mc_bool
191       \bool_set_false:N \l__tag_mc_artifact_bool

```

```

192     \__tag_mc_lua_unset_mc_type_attr:
193     \tl_set:Nn \l__tag_mc_key_tag_tl { }
194     \tl_gset:Nn \g__tag_mc_key_tag_tl { }
195   }
196 }

```

(End definition for \tag_mc_end:. This function is documented on page 51.)

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

```

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

```

(End 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)  198 \keys_define:nn { __tag / mc }
alttext_(mc-key) 199 {
actualtext_(mc-key) 200   tag .code:n = %
label_(mc-key)    201   {
artifact_(mc-key) 202     \tl_set:Nx \l__tag_mc_key_tag_tl { #1 }
                  203     \tl_gset:Nx \g__tag_mc_key_tag_tl { #1 }
                  204     \lua_now:e
                  205     {
                  206       ltx.__tag.func.store_mc_data(\__tag_get_mc_abs_cnt:,"tag","#1")
                  207     }
                  208   },
raw .code:n =    209   {
                  210     \tl_put_right:Nx \l__tag_mc_key_properties_tl { #1 }
                  211     \lua_now:e
                  212     {
                  213       ltx.__tag.func.store_mc_data(\__tag_get_mc_abs_cnt:,"raw","#1")
                  214     }
                  215   },
alttext .code:n = % Alt property
                  216   {
                  217     \str_set_convert:Noon
                  218     \l__tag_tmpa_str
                  219     { #1 }
                  220     { default }
                  221     { utf16/hex }
                  222     \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
                  223     \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
                  224     \lua_now:e
                  225     {
                  226       ltx.__tag.func.store_mc_data
                  227       (
                  228         \__tag_get_mc_abs_cnt:,"alt","/Alt~<\str_use:N \l__tag_tmpa_str>"
                  229       )
                  230     }
                  231   },
actualtext .code:n = % Alt property
                  232   {
                  233     \str_set_convert:Noon
                  234     \l__tag_tmpa_str
                  235     { #1 }
                  236     { default }
                  237     { utf16/hex }
                  238     \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
                  239     \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
                  240     \lua_now:e
                  241     {
                  242       ltx.__tag.func.store_mc_data
                  243       (
                  244         \__tag_get_mc_abs_cnt:,"alt","/Alt~<\str_use:N \l__tag_tmpa_str>"
                  245       )
                  246     }
                  247   },

```

```

235     {
236         \str_set_convert:Noon
237         \l__tag_tmpa_str
238         { #1 }
239         { default }
240         { utf16/hex }
241         \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
242         \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
243         \lua_now:e
244         {
245             ltx.__tag.func.store_mc_data
246             (
247                 \__tag_get_mc_abs_cnt:,
248                 "actualtext",
249                 "/ActualText~<\str_use:N \l__tag_tmpa_str>"
250             )
251         }
252     },
253     label .code:n =
254     {
255         \tl_set:Nn\l__tag_mc_key_label_tl { #1 }
256         \lua_now:e
257         {
258             ltx.__tag.func.store_mc_data
259             (
260                 \__tag_get_mc_abs_cnt:,"label","#1"
261             )
262         }
263     },
264     __artifact-store .code:n =
265     {
266         \lua_now:e
267         {
268             ltx.__tag.func.store_mc_data
269             (
270                 \__tag_get_mc_abs_cnt:,"artifact","#1"
271             )
272         }
273     },
274     artifact .code:n      =
275     {
276         \exp_args:Nnx
277         \keys_set:nn
278         { __tag / mc}
279         { __artifact-bool, __artifact-type=#1, tag=Artifact }
280         \exp_args:Nnx
281         \keys_set:nn
282         { __tag / mc }
283         { __artifact-store=\l__tag_mc_artifact_type_tl }
284     },
285     artifact .default:n    = { notype }
286 }
287
288 </luamode>

```

(End definition for tag `mc-key` and others. These functions are documented on page 52.)

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

These commands start and end a new structure. They don't start a group. They set all their values globally.

<code>\tag_struct_use:n</code>	<code>\tag_struct_use:n{<label>}</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.

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

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

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

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

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

2 Public keys

2.1 Keys for the structure commands

<code>tag_␣(struct-key)</code>	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 <code>type/NS</code> , where <code>NS</code> is the shorthand of a declared name space. Currently the names spaces <code>pdf</code> , <code>pdf2</code> , <code>mathml</code> and <code>user</code> 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.
--------------------------------	--

<u>stash_□(struct-key)</u>	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.
<u>label_□(struct-key)</u>	This key sets a label by which one can use the structure later in another structure. Internally the label name will start with tagpdfstruct- .
<u>title_□(struct-key)</u> <u>title-o_□(struct-key)</u>	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.
<u>alttext_□(struct-key)</u>	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.
<u>actualtext_□(struct-key)</u>	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.
<u>lang_□(struct-key)</u>	This key allows to set the language for a structure element. The value should be a bcp-identifier, e.g. de-De .
<u>ref_□(struct-key)</u>	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} .
<u>E_□(struct-key)</u>	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 sticked to E).
<u>AF_□(struct-key)</u>	AF = <i>⟨object name⟩</i>
<u>AFinline_□(struct-key)</u>	AF-inline = <i>⟨text content⟩</i>
<u>AFinline-o_□(struct-key)</u>	These keys allows to reference an associated file in the structure element. The value <i>⟨object name⟩</i> should be the name of an object pointing to the /Filespec dictionary as expected by \pdf_object_ref:n from a current l3kernel . The value AF-inline is some text, which is embedded in the PDF as a text file with mime type text/plain . AF-inline-o is like AF-inline but expands the value once. Future versions will perhaps extend this to more mime types, but it is still a research task to find out what is really needed. AF can be used more than once, to associate more than one file. The inline keys can be used only once per structure. Additional calls are ignored.

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

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

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

attribute-class_□(struct-key)

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

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

2.2 Setup keys

newattribute_□(setup-key) `newattribute = {<name>}{<Content>}`

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
{
  newattribute =
    {TH-col}{/O /Table /Scope /Column},
  newattribute =
    {TH-row}{/O /Table /Scope /Row},
}
```

root-AF_□(setup-key) `root-AF = <object name>`

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

```
1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-struct-code} {2022-01-09} {0.93}
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 <*package>
7 \newcounter { g__tag_struct_abs_int }
8 \int_gzero:N \c@g__tag_struct_abs_int
```

(End 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.

⁹ __tag_seq_new:N \g__tag_struct_objR_seq

(End definition for \g__tag_struct_objR_seq.)

\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 absolut mc num, the value the pdf directory.

¹⁰ __tag_prop_new:N \g__tag_struct_cont_mc_prop

(End 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.

¹¹ \seq_new:N \g__tag_struct_stack_seq

¹² \seq_gpush:Nn \g__tag_struct_stack_seq {0}

(End 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.

¹³ \seq_new:N \g__tag_struct_tag_stack_seq

¹⁴ \seq_gpush:Nn \g__tag_struct_tag_stack_seq {Root}

(End definition for \g__tag_struct_tag_stack_seq.)

\g__tag_struct_stack_current_tl The global variable will hold the current structure number. The local temporary variable will hold the parent when we fetch it from the stack.

¹⁵ \tl_new:N \g__tag_struct_stack_current_tl

¹⁶ \tl_new:N \l__tag_struct_stack_parent_tmpa_tl

(End definition for \g__tag_struct_stack_current_tl and \l__tag_struct_stack_parent_tmpa_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_0_prop for the root and \g_@@_struct_N_prop, $N \geq 1$ 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,lange,alt,E,actualtext)

`\c__tag_struct_StructTreeRoot_entries_seq` 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.

```

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

```

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

3.1 Variables used by the keys

`\g__tag_struct_tag_tl` Use by the tag key to store the tag and the namespace.

```

\g__tag_struct_tag_NS_tl
54 \tl_new:N \g__tag_struct_tag_tl
55 \tl_new:N \g__tag_struct_tag_NS_tl

```

(End definition for `\g__tag_struct_tag_tl` and `\g__tag_struct_tag_NS_tl`.)

`\l__tag_struct_key_label_tl` This will hold the label value.

```

56 \tl_new:N \l__tag_struct_key_label_tl

```

(End definition for \l__tag_struct_key_label_tl.)

```
\l__tag_struct_elem_stash_bool This will keep track of the stash status
57 \bool_new:N \l__tag_struct_elem_stash_bool
(End definition for \l__tag_struct_elem_stash_bool.)
```

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
58 \cs_new:Npn \__tag_struct_output_prop_aux:nn #1 #2 % #1 num, #2 key
59 {
60   \prop_if_in:cnT
61     { g__tag_struct_#1_prop }
62     { #2 }
63   {
64     \c_space_tl/#2~ \prop_item:cn{ g__tag_struct_#1_prop } { #2 }
65   }
66 }
67
68 \cs_new_protected:Npn \__tag_new_output_prop_handler:n #1
69 {
70   \cs_new:cn { __tag_struct_output_prop_#1:n }
71   {
72     \__tag_struct_output_prop_aux:nn {#1}{##1}
73   }
74 }
(End definition for \__tag_struct_output_prop_aux:nn and \__tag_new_output_prop_handler:n.)
```

4.1 Initialization of the StructTreeRoot

The first structure element, the StructTreeRoot is special, so created manually. The underlying object is @@/struct/0 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.

```
75 \tl_gset:Nn \g__tag_struct_stack_current_tl {0}

g__tag_struct_0_prop
g__tag_struct_kids_0_seq
76 \__tag_prop_new:c { g__tag_struct_0_prop }
77 \__tag_new_output_prop_handler:n {0}
78 \__tag_seq_new:c { g__tag_struct_kids_0_seq }
79
80 \__tag_prop_gput:cn
```

```

81 { g__tag_struct_0_prop }
82 { Type }
83 { /StructTreeRoot }
84
85
86

```

Namespaces are pdf 2.0 but it doesn't harm to have an empty entry. We could add a test, but if the code moves into the kernel, timing could get tricky.

```

87 \__tag_prop_gput:cnx
88 { g__tag_struct_0_prop }
89 { Namespaces }
90 { \pdf_object_ref:n { __tag/tree/namespaces } }

```

(End definition for g__tag_struct_0_prop and g__tag_struct_kids_0_seq.)

4.2 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:nx

```

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!

```

91 \cs_new:Npn \__tag_struct_mcid_dict:n #1 %#1 MCID absnum
92 {
93   <<
94   /Type \c_space_tl /MCR \c_space_tl
95   /Pg
96   \c_space_tl
97   \pdf_pageobject_ref:n { \__tag_ref_value:enn{mcid-#1}{tagabspage}{1} }
98   /MCID \c_space_tl \__tag_ref_value:enn{mcid-#1}{tagmcid}{1}
99   >>
100 }
101 \cs_new_protected:Npn \__tag_struct_kid_mc_gput_right:nn #1 #2 %#1 structure num, #2 MCID absn
102 {
103   \__tag_seq_gput_right:cx
104   { g__tag_struct_kids_#1_seq }
105   {
106     \__tag_struct_mcid_dict:n {#2}
107   }
108   \__tag_seq_gput_right:cn
109   { g__tag_struct_kids_#1_seq }
110   {
111     \prop_item:Nn \g__tag_struct_cont_mc_prop {#2}
112   }
113 }
114 \cs_generate_variant:Nn \__tag_struct_kid_mc_gput_right:nn {nx}
115

```

(End definition for `_tag_struct_kid_mc_gput_right:nn`.)

`_tag_struct_kid_struct_gput_right:nn`
`_tag_struct_kid_struct_gput_right:xx`

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

```

116 \cs_new_protected:Npn\_tag_struct_kid_struct_gput_right:nn #1 #2 %#1 num of parent struct, #2
117 {
118   \_tag_seq_gput_right:cx
119   { g\_tag_struct_kids_#1_seq }
120   {
121     \pdf_object_ref:n { \_tag/struct/#2 }
122   }
123 }
124
125 \cs_generate_variant:Nn \_tag_struct_kid_struct_gput_right:nn {xx}

```

(End definition for `_tag_struct_kid_struct_gput_right:nn`.)

`_tag_struct_kid_OBJR_gput_right:nn`
`_tag_struct_kid_OBJR_gput_right:xx`

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.

```

126 \cs_new_protected:Npn\_tag_struct_kid_OBJR_gput_right:nn #1 #2 %#1 num of parent struct,
127                                     %#2 obj reference
128 {
129   \pdf_object_unnamed_write:nn
130   { dict }
131   {
132     /Type/OBJR/Obj~#2
133   }
134   \_tag_seq_gput_right:cx
135   { g\_tag_struct_kids_#1_seq }
136   {
137     \pdf_object_ref_last:
138   }
139 }
140
141 \cs_generate_variant:Nn \_tag_struct_kid_OBJR_gput_right:nn { xx }
142

```

(End definition for `_tag_struct_kid_OBJR_gput_right:nn`.)

`_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.

```

143 \cs_new_protected:Npn\_tag_struct_exchange_kid_command:N #1 %#1 = seq var
144 {
145   \seq_gpop_left:NN #1 \l\_tag_tmpa_tl
146   \regex_replace_once:nnN
147   { \c{\_tag_mc_insert_mcid_kids:n} }
148   { \c{\_tag_mc_insert_mcid_single_kids:n} }
149   \l\_tag_tmpa_tl
150   \seq_gput_left:NV #1 \l\_tag_tmpa_tl
151 }
152
153 \cs_generate_variant:Nn \_tag_struct_exchange_kid_command:N { c }

```

(End 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.

```

154 \cs_new_protected:Npn \_tag_struct_fill_kid_key:n #1 %#1 is the struct num
155 {
156   \bool_if:NF\g__tag_mode_lua_bool
157   {
158     \seq_clear:N \l__tag_tmpa_seq
159     \seq_map_inline:cn { g__tag_struct_kids_#1_seq }
160     { \seq_put_right:Nx \l__tag_tmpa_seq { ##1 } }
161     %\seq_show:c { g__tag_struct_kids_#1_seq }
162     %\seq_show:N \l__tag_tmpa_seq
163     \seq_remove_all:Nn \l__tag_tmpa_seq {}
164     %\seq_show:N \l__tag_tmpa_seq
165     \seq_gset_eq:cn { g__tag_struct_kids_#1_seq } \l__tag_tmpa_seq
166   }
167
168   \int_case:nnF
169   {
170     \seq_count:c
171     {
172       g__tag_struct_kids_#1_seq
173     }
174   }
175   {
176     { 0 }
177     { } %no kids, do nothing
178     { 1 } % 1 kid, insert
179     {
180       % in this case we need a special command in
181       % luamode to get the array right. See issue #13
182       \bool_if:NT\g__tag_mode_lua_bool
183       {
184         \_tag_struct_exchange_kid_command:c
185         {g__tag_struct_kids_#1_seq}
186       }
187       \_tag_prop_gput:cnx { g__tag_struct_#1_prop } {K}
188       {
189         \seq_item:cn
190         {
191           g__tag_struct_kids_#1_seq
192         }
193         {1}
194       }
195     } %
196   }
197   { %many kids, use an array
198     \_tag_prop_gput:cnx { g__tag_struct_#1_prop } {K}
199     {
200       [
201         \seq_use:cn
202         {
203           g__tag_struct_kids_#1_seq

```

```

204         }
205         {
206             \c_space_tl
207         }
208     ]
209 }
210 }
211 }
212

```

(End definition for _tag_struct_fill_kid_key:n.)

_tag_struct_get_dict_content:nN This maps the dictionary content of a structure into a tl-var. Basically it does what \pdfdict_use:n does. TODO!! this looks over-complicated. Check if it can be done with pdfdict now.

```

213 \cs_new_protected:Npn \_tag_struct_get_dict_content:nN #1 #2 %#1: structure num
214 {
215     \tl_clear:N #2
216     \seq_map_inline:cn
217     {
218         c__tag_struct_
219         \int_compare:nNnTF{#1}={0}{StructTreeRoot}{StructElem}
220         _entries_seq
221     }
222     {
223         \tl_put_right:Nx
224         #2
225         {
226             \prop_if_in:cnT
227             { g__tag_struct_#1_prop }
228             { ##1 }
229             {
230                 \c_space_tl/##1~\prop_item:cn{ g__tag_struct_#1_prop } { ##1 }
231             }
232         }
233     }
234 }

```

(End definition for _tag_struct_get_dict_content:nN.)

_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.

```

235 \cs_new_protected:Npn \_tag_struct_write_obj:n #1 % #1 is the struct num
236 {
237     \pdf_object_if_exist:nTF { __tag/struct/#1 }
238     {
239         \_tag_struct_fill_kid_key:n { #1 }
240         \_tag_struct_get_dict_content:nN { #1 } \l__tag_tmpa_tl
241         \exp_args:Nx
242         \pdf_object_write:nx
243         { __tag/struct/#1 }
244         {
245             \l__tag_tmpa_tl
246         }

```



```

247     }
248     {
249     \msg_error:nnn { tag } { struct-no-objnum } { #1}
250     }
251 }

```

(End definition for _tag_struct_write_obj:n.)

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

Annotations used as structure content must

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

For a link this looks like this

```

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

```

```

252 \cs_new_protected:Npn \_tag_struct_insert_annot:nn #1 #2 %1 object reference to the annotation
253                                     %#2 structparent number
254 {
255   \bool_if:NT \g__tag_active_struct_bool
256   {
257     %get the number of the parent structure:
258     \seq_get:NNF
259       \g__tag_struct_stack_seq
260       \l__tag_struct_stack_parent_tmpa_tl
261       {
262         \msg_error:nn { tag } { struct-faulty-nesting }
263       }
264     %put the obj number of the annot in the kid entry, this also creates
265     %the OBJR object
266     \_tag_struct_kid_OBJR_gput_right:xx
267     {
268       \l__tag_struct_stack_parent_tmpa_tl
269     }
270     {
271       #1 %
272     }
273     % add the parent obj number to the parent tree:
274     \exp_args:Nnx
275     \_tag_parenttree_add_objr:nn

```

```

276         {
277             #2
278         }
279         {
280             \pdf_object_ref:e { __tag/struct/\l__tag_struct_stack_parent_tmpa_tl }
281         }
282         % increase the int:
283         \stepcounter{ g__tag_parenttree_obj_int }
284     }
285 }

```

(End 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**. We will need to handle nesting

```

286 \cs_new:Npn \__tag_get_data_struct_tag:
287 {
288     \exp_args:Ne
289     \tl_tail:n
290     {
291         \prop_item:cn {g__tag_struct\_g__tag_struct_stack_current_tl _prop}{S}
292     }
293 }

```

(End definition for __tag_get_data_struct_tag:.)

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.

```

label_(struct-key)
stash_(struct-key)
tag_(struct-key)
title_(struct-key)
title-o_(struct-key)
alttext_(struct-key)
actualtext_(struct-key)
lang_(struct-key)
ref_(struct-key)
E_(struct-key)
294 \keys_define:nn { __tag / struct }
295 {
296     label .tl_set:N      = \l__tag_struct_key_label_tl,
297     stash .bool_set:N    = \l__tag_struct_elem_stash_bool,
298     tag .code:n          = % S property
299     {
300         \seq_set_split:Nne \l__tag_tmpa_seq { / } {#1/\prop_item:Nn\g__tag_role_tags_NS_prop{
301         \tl_gset:Nx \g__tag_struct_tag_tl { \seq_item:Nn\l__tag_tmpa_seq {1} }
302         \tl_gset:Nx \g__tag_struct_tag_NS_tl { \seq_item:Nn\l__tag_tmpa_seq {2} }
303         \__tag_check_structure_tag:N \g__tag_struct_tag_tl
304         \__tag_prop_gput:cnx
305         { g__tag_struct\_int_eval:n {\c@g__tag_struct_abs_int}_prop }
306         { S }
307         { \pdf_name_from_unicode_e:n{ \g__tag_struct_tag_tl } } %
308     \prop_get:NVNT \g__tag_role_NS_prop\g__tag_struct_tag_NS_tl\l__tag_tmpa_tl
309     {
310         \__tag_prop_gput:cnx
311         { g__tag_struct\_int_eval:n {\c@g__tag_struct_abs_int}_prop }
312         { NS }
313         { \l__tag_tmpa_tl } %
314     }

```

```

315     },
316     title .code:n          = % T property
317     {
318         \str_set_convert:Nnon
319         \l__tag_tmpa_str
320         { #1 }
321         { default }
322         { utf16/hex }
323         \__tag_prop_gput:cnx
324         { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
325         { T }
326         { <\l__tag_tmpa_str> }
327     },
328     title-o .code:n        = % T property
329     {
330         \str_set_convert:Nnon
331         \l__tag_tmpa_str
332         { #1 }
333         { default }
334         { utf16/hex }
335         \__tag_prop_gput:cnx
336         { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
337         { T }
338         { <\l__tag_tmpa_str> }
339     },
340     alttext .code:n        = % Alt property
341     {
342         \str_set_convert:Noon
343         \l__tag_tmpa_str
344         { #1 }
345         { default }
346         { utf16/hex }
347         \__tag_prop_gput:cnx
348         { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
349         { Alt }
350         { <\l__tag_tmpa_str> }
351     },
352     actualtext .code:n     = % ActualText property
353     {
354         \str_set_convert:Noon
355         \l__tag_tmpa_str
356         { #1 }
357         { default }
358         { utf16/hex }
359         \__tag_prop_gput:cnx
360         { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
361         { ActualText }
362         { <\l__tag_tmpa_str>}
363     },
364     lang .code:n           = % Lang property
365     {
366         \__tag_prop_gput:cnx
367         { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
368         { Lang }

```

```

369         { (#1) }
370     },
371     ref .code:n          = % Lang property
372     {
373         \tl_clear:N\l__tag_tmpa_tl
374         \clist_map_inline:nn {#1}
375         {
376             \tl_put_right:Nx \l__tag_tmpa_tl
377             {~\ref_value:nn{tagpdfstruct-##1}{tagstructobj} }
378         }
379         \__tag_prop_gput:cnx
380         { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
381         { Ref }
382         { [\l__tag_tmpa_tl] }
383     },
384     E .code:n           = % E property
385     {
386         \str_set_convert:Nnon
387         \l__tag_tmpa_str
388         { #1 }
389         { default }
390         { utf16/hex }
391         \__tag_prop_gput:cnx
392         { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
393         { E }
394         { <\l__tag_tmpa_str> }
395     },
396 }

```

(End definition for label (struct-key) and others. These functions are documented on page 80.)

AF_⊔(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 is an array and can be used more than once, so we store it in a tl. which is expanded. AFinline can be use only once (more quite probably doesn't make sense).

```

397 \cs_new_protected:Npn \__tag_struct_add_AF:nn #1 #2 % #1 struct num #2 object name
398 {
399     \tl_if_exist:cTF
400     {
401         g__tag_struct_#1_AF_tl
402     }
403     {
404         \tl_gput_right:cx
405         { g__tag_struct_#1_AF_tl }
406         { ~ \pdf_object_ref:n {#2} }
407     }
408     {
409         \tl_new:c
410         { g__tag_struct_#1_AF_tl }
411         \tl_gset:cx
412         { g__tag_struct_#1_AF_tl }
413         { \pdf_object_ref:n {#2} }
414     }

```

```

415 }
416 \cs_generate_variant:Nn \__tag_struct_add_AF:nn {en,ee}
417 \keys_define:nn { __tag / struct }
418 {
419     AF .code:n          = % AF property
420     {
421         \pdf_object_if_exist:nTF {#1}
422         {
423             \__tag_struct_add_AF:en { \int_eval:n {\c@g__tag_struct_abs_int} }{#1}
424             \__tag_prop_gput:cnx
425             { g__tag_struct\_int_eval:n {\c@g__tag_struct_abs_int}_prop }
426             { AF }
427             {
428                 [
429                     \tl_use:c
430                     { g__tag_struct\_int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
431                 ]
432             }
433         }
434     }
435 }
436 },
437 ,AFinline .code:n =
438 {
439     \group_begin:
440     \pdf_object_if_exist:eF {__tag/fileobj\int_use:N\c@g__tag_struct_abs_int}
441     {
442         \pdffile_embed_stream:nxx
443         {#1}
444         {tag-AFfile\int_use:N\c@g__tag_struct_abs_int.txt}
445         {__tag/fileobj\int_use:N\c@g__tag_struct_abs_int}
446         \__tag_struct_add_AF:ee
447         { \int_eval:n {\c@g__tag_struct_abs_int} }
448         { __tag/fileobj\int_use:N\c@g__tag_struct_abs_int }
449         \__tag_prop_gput:cnx
450         { g__tag_struct\_int_use:N\c@g__tag_struct_abs_int _prop }
451         { AF }
452         {
453             [
454                 \tl_use:c
455                 { g__tag_struct\_int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
456             ]
457         }
458     }
459 }
460 \group_end:
461 }
462 ,AFinline-o .code:n =
463 {
464     \group_begin:
465     \pdf_object_if_exist:eF {__tag/fileobj\int_use:N\c@g__tag_struct_abs_int}
466     {
467         \pdffile_embed_stream:oxx
468         {#1}

```

```

469         {tag-AFfile\int_use:N\c@g__tag_struct_abs_int.txt}
470         {__tag/fileobj\int_use:N\c@g__tag_struct_abs_int}
471     \__tag_struct_add_AF:ee
472     { \int_eval:n {\c@g__tag_struct_abs_int} }
473     { __tag/fileobj\int_use:N\c@g__tag_struct_abs_int }
474     \__tag_prop_gput:cnx
475     { g__tag_struct_\int_use:N\c@g__tag_struct_abs_int _prop }
476     { AF }
477     {
478     [
479         \tl_use:c
480         { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
481     ]
482     }
483 }
484 \group_end:
485 }
486 }
487 \end{package}

```

(End definition for AF (struct-key), AFinline (struct-key), and AFinline-o (struct-key). These functions are documented on page 80.)

root-AF_□(setup-key) The root structure can take AF keys too, so we provide a key for it. This key is used with `\tagpdfsetup`, not in a structure!

```

488 \keys_define:nn { __tag / setup }
489 {
490     root-AF .code:n =
491     {
492         \pdf_object_if_exist:nTF {#1}
493         {
494             \__tag_struct_add_AF:en { 0 }{#1}
495             \__tag_prop_gput:cnx
496             { g__tag_struct_0_prop }
497             { AF }
498             {
499             [
500                 \tl_use:c
501                 { g__tag_struct_0_AF_tl }
502             ]
503             }
504         }
505     }
506 }
507 },
508 }
509 }

```

(End definition for root-AF (setup-key). This function is documented on page 81.)

6 User commands

```

\tag_struct_begin:n
\tag_struct_end:

```

```

510 (*package | debug)
511 (package)\cs_new_protected:Npn \tag_struct_begin:n #1 %1 key-val
512 (debug)\cs_set_protected:Npn \tag_struct_begin:n #1 %1 key-val
513 {
514 (package)\__tag_check_if_active_struct:T
515 (debug)\__tag_check_if_active_struct:TF
516 {
517   \group_begin:
518   \int_gincr:N \c@g__tag_struct_abs_int
519   \__tag_prop_new:c { g__tag_struct\_int_eval:n { \c@g__tag_struct_abs_int }_prop }
520   \__tag_new_output_prop_handler:n {\int_eval:n { \c@g__tag_struct_abs_int }}
521   \__tag_seq_new:c { g__tag_struct_kids\_int_eval:n { \c@g__tag_struct_abs_int }_seq}
522   \exp_args:Ne
523   \pdf_object_new:nn
524     { __tag/struct/\int_eval:n { \c@g__tag_struct_abs_int } }
525     { dict }
526   \__tag_prop_gput:cno
527     { g__tag_struct\_int_eval:n { \c@g__tag_struct_abs_int }_prop }
528     { Type }
529     { /StructElem }
530   \keys_set:nn { __tag / struct} { #1 }
531   \__tag_check_structure_has_tag:n { \int_eval:n { \c@g__tag_struct_abs_int} }
532   \tl_if_empty:NF
533     \l__tag_struct_key_label_tl
534     {
535       \__tag_ref_label:en{tagpdfstruct-\l__tag_struct_key_label_tl}{struct}
536     }
537   %get the potential parent from the stack:
538   \seq_get:NNF
539     \g__tag_struct_stack_seq
540     \l__tag_struct_stack_parent_tmpa_tl
541     {
542       \msg_error:nn { tag } { struct-faulty-nesting }
543     }
544   \seq_gpush:NV \g__tag_struct_stack_seq \c@g__tag_struct_abs_int
545   \seq_gpush:NV \g__tag_struct_tag_stack_seq \g__tag_struct_tag_tl
546   \tl_gset:NV \g__tag_struct_stack_current_tl \c@g__tag_struct_abs_int
547   %\seq_show:N \g__tag_struct_stack_seq
548   \bool_if:NF
549     \l__tag_struct_elem_stash_bool
550     {%set the parent
551       \__tag_prop_gput:cnx
552         { g__tag_struct\_int_eval:n { \c@g__tag_struct_abs_int}_prop }
553         { P }
554         {
555           \pdf_object_ref:e { __tag/struct/\l__tag_struct_stack_parent_tmpa_tl }
556         }
557       %record this structure as kid:
558       %\tl_show:N \g__tag_struct_stack_current_tl
559       %\tl_show:N \l__tag_struct_stack_parent_tmpa_tl
560       \__tag_struct_kid_struct_gput_right:xx
561         { \l__tag_struct_stack_parent_tmpa_tl }
562         { \g__tag_struct_stack_current_tl }
563       %\prop_show:c { g__tag_struct\_g__tag_struct_stack_current_tl _prop }

```

```

564         \%seq_show:c {g__tag_struct_kids\_l__tag_struct_stack_parent_tmpa_tl _seq}
565     }
566     \%prop_show:c { g__tag_struct\_g__tag_struct_stack_current_tl _prop }
567     \%seq_show:c {g__tag_struct_kids\_l__tag_struct_stack_parent_tmpa_tl _seq}
568 <debug> \__tag_debug_struct_begin_insert:n { #1 }
569     \group_end:
570 }
571 <debug>{ \__tag_debug_struct_begin_ignore:n { #1 } }
572 }
573 <package>\cs_new_protected:Nn \tag_struct_end:
574 <debug>\cs_set_protected:Nn \tag_struct_end:
575 { %take the current structure num from the stack:
576     %the objects are written later, lua mode hasn't all needed info yet
577     \%seq_show:N \g__tag_struct_stack_seq
578 <package>\__tag_check_if_active_struct:T
579 <debug>\__tag_check_if_active_struct:TF
580 {
581     \seq_gpop:NN \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
582     \seq_gpop:NNTF \g__tag_struct_stack_seq \l__tag_tmpa_tl
583     {
584         \__tag_check_info_closing_struct:o { \g__tag_struct_stack_current_tl }
585     }
586     { \__tag_check_no_open_struct: }
587     % get the previous one, shouldn't be empty as the root should be there
588     \seq_get:NNTF \g__tag_struct_stack_seq \l__tag_tmpa_tl
589     {
590         \tl_gset:NV \g__tag_struct_stack_current_tl \l__tag_tmpa_tl
591     }
592     {
593         \__tag_check_no_open_struct:
594     }
595     \seq_get:NNT \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
596     {
597         \tl_gset:NV \g__tag_struct_tag_tl \l__tag_tmpa_tl
598     }
599 <debug>\__tag_debug_struct_end_insert:
600 }
601 <debug>{\__tag_debug_struct_end_ignore:}
602 }
603 </package|debug>

```

(End definition for \tag_struct_begin:n and \tag_struct_end:. These functions are documented on page 79.)

\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.

```

604 <*package>
605 \cs_new_protected:Nn \tag_struct_use:n {%#1 is the label
606 {
607     \__tag_check_if_active_struct:T
608     {
609         \prop_if_exist:cTF
610         { g__tag_struct\___tag_ref_value:enn{tagpdfstruct-#1}{tagstruct}{unknown}_prop } %
611         {

```



```

612         \__tag_check_struct_used:n {#1}
613         %add the label structure as kid to the current structure (can be the root)
614         \__tag_struct_kid_struct_gput_right:xx
615         { \g__tag_struct_stack_current_tl }
616         { \__tag_ref_value:enn{tagpdfstruct-#1}{tagstruct}{0} }
617         %add the current structure to the labeled one as parents
618         \__tag_prop_gput:cnx
619         { g__tag_struct_\__tag_ref_value:enn{tagpdfstruct-#1}{tagstruct}{0}_prop }
620         { P }
621         {
622             \pdf_object_ref:e { __tag/struct/\g__tag_struct_stack_current_tl }
623         }
624     }
625     {
626         \msg_warning:nnn{ tag }{struct-label-unknown}{#1}
627     }
628 }
629 }

```

(End definition for `\tag_struct_use:n`. This function is documented on page 79.)

`\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:.`

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.

```

630 \cs_new_protected:Npn \tag_struct_insert_annot:nn #1 #2 {%#1 should be an object reference
631                                     %#2 struct parent num
632 {
633     \__tag_check_if_active_struct:T
634     {
635         \__tag_struct_insert_annot:nn {#1}{#2}
636     }
637 }
638
639 \cs_generate_variant:Nn \tag_struct_insert_annot:nn {xx}
640 \cs_new:Npn \tag_struct_parent_int: {\int_use:c { c@g__tag_parenttree_obj_int }}
641
642 \</package>
643

```

(End definition for `\tag_struct_insert_annot:nn` and `\tag_struct_parent_int:.` These functions are documented on page 79.)

7 Attributes and attribute classes

```

644 \<header>
645 \ProvidesExplPackage {tagpdf-attr-code} {2022-01-09} {0.93}
646 {part of tagpdf - code related to attributes and attribute classes}
647 \</header>

```

7.1 Variables

`\g__tag_attr_entries_prop` `\g_@@_attr_entries_prop` will store attribute names and their dictionary content.
`\g__tag_attr_class_used_seq` `\g_@@_attr_class_used_seq` will hold the attributes which have been used as class
`\g__tag_attr_objref_prop`
`\l__tag_attr_value_tl`

name. `\l_@@_attr_value_tl` is used to build the attribute array or key. Everytime an attribute is used for the first time, and object is created with its content, the name-object reference relation is stored in `\g_@@_attr_objref_prop`

```

648 (*package)
649 \prop_new:N \g__tag_attr_entries_prop
650 \seq_new:N \g__tag_attr_class_used_seq
651 \tl_new:N \l__tag_attr_value_tl
652 \prop_new:N \g__tag_attr_objref_prop %will contain obj num of used attributes

(End definition for \g__tag_attr_entries_prop and others.)

```

7.2 Commands and keys

`__tag_attr_new_entry:nn` This allows to define attributes. Defined attributes are stored in a global property. `newattribute_␣(setup-key)` `newattribute` expects two brace group, the name and the content. The content typically needs an `/O` key for the owner. An example look like this.

```

\tagpdfsetup
{
  newattribute =
    {TH-col}{/O /Table /Scope /Column},
  newattribute =
    {TH-row}{/O /Table /Scope /Row},
}

653 \cs_new_protected:Npn \__tag_attr_new_entry:nn #1 #2 %#1:name, #2: content
654 {
655   \prop_gput:Nnn \g__tag_attr_entries_prop
656     {#1}{#2}
657 }
658
659 \keys_define:nn { __tag / setup }
660 {
661   newattribute .code:n =
662     {
663       \__tag_attr_new_entry:nn #1
664     }
665 }

```

(End definition for `__tag_attr_new_entry:nn` and `newattribute (setup-key)`. This function is documented on page 81.)

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

```

666 \keys_define:nn { __tag / struct }
667 {
668   attribute-class .code:n =
669     {
670       \clist_set:No \l__tag_tmpa_clist { #1 }
671       \seq_set_from_clist:NN \l__tag_tmpa_seq \l__tag_tmpa_clist
672       \seq_map_inline:Nn \l__tag_tmpa_seq
673         {
674           \prop_if_in:NnF \g__tag_attr_entries_prop {##1}
675             {

```

```

676         \msg_error:nnn { tag } { attr-unknown } { ##1 }
677     }
678     \seq_gput_left:Nn\g__tag_attr_class_used_seq { ##1}
679 }
680 \seq_set_map:NNn \l__tag_tmpb_seq \l__tag_tmpa_seq
681 {
682     /##1
683 }
684 \tl_set:Nx \l__tag_tmpa_tl
685 {
686     \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}
687     \seq_use:Nn \l__tag_tmpb_seq { \c_space_tl }
688     \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}
689 }
690 \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 0 }
691 {
692     \__tag_prop_gput:cnx
693     { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
694     { C }
695     { \l__tag_tmpa_tl }
696     %\prop_show:c { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
697 }
698 }
699 }

```

(End definition for attribute-class (struct-key). This function is documented on page 81.)

attribute_□(struct-key)

```

700 \keys_define:nn { __tag / struct }
701 {
702     attribute .code:n = % A property (attribute, value currently a dictionary)
703     {
704         \clist_set:No          \l__tag_tmpa_clist { #1 }
705         \seq_set_from_clist:NN \l__tag_tmpa_seq \l__tag_tmpa_clist
706         \tl_set:Nx \l__tag_attr_value_tl
707         {
708             \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}%
709         }
710         \seq_map_inline:Nn \l__tag_tmpa_seq
711         {
712             \prop_if_in:NnF \g__tag_attr_entries_prop {##1}
713             {
714                 \msg_error:nnn { tag } { attr-unknown } { ##1 }
715             }
716             \prop_if_in:NnF \g__tag_attr_objref_prop {##1}
717             {%\prop_show:N \g__tag_attr_entries_prop
718              \pdf_object_unnamed_write:nx
719              { dict }
720              {
721                  \prop_item:Nn\g__tag_attr_entries_prop {##1}
722              }
723              \prop_gput:Nnx \g__tag_attr_objref_prop {##1} {\pdf_object_ref_last:}
724             }
725         \tl_put_right:Nx \l__tag_attr_value_tl

```

```

726         {
727             \c_space_tl
728             \prop_item:Nn \g__tag_attr_objref_prop {##1}
729         }
730 % \tl_show:N \l__tag_attr_value_tl
731     }
732     \tl_put_right:Nx \l__tag_attr_value_tl
733     { %[
734         \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}%
735     }
736 % \tl_show:N \l__tag_attr_value_tl
737     \__tag_prop_gput:cnx
738     { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
739     { A }
740     { \l__tag_attr_value_tl }
741 },
742 }
743 \end{package}

```

(End definition for attribute (struct-key). This function is documented on page 81.)

Part VIII

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

```
1 <@@=tag>
2 <*luatex>
3 \ProvidesExplFile {tagpdf-luatex.def} {2022-01-09} {0.93}
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.@@.tables.\cs_to_str:N#1 = {} }
13 }
14
15
16 \cs_set_protected:Npn \__tag_seq_new:N #1
17 {
18   \seq_new:N #1
19   \lua_now:e { ltx.@@.tables.\cs_to_str:N#1 = {} }
20 }
21
22
23 \cs_set_protected:Npn \__tag_prop_gput:Nnn #1 #2 #3
24 {
25   \prop_gput:Nnn #1 { #2 } { #3 }
26   \lua_now:e { ltx.@@.tables.\cs_to_str:N#1 ["#2"] = "#3" }
27 }
28
29
```

```

30 \cs_set_protected:Npn \__tag_seq_gput_right:Nn #1 #2
31 {
32   \seq_gput_right:Nn #1 { #2 }
33   \lua_now:e { table.insert(ltx.__tag.tables.\cs_to_str:N#1, "#2") }
34 }
35
36 %Hm not quite sure about the naming
37
38 \cs_set:Npn \__tag_seq_item:cn #1 #2
39 {
40   \lua_now:e { tex.print(ltx.__tag.tables.#1[#2]) }
41 }
42
43 \cs_set:Npn \__tag_prop_item:cn #1 #2
44 {
45   \lua_now:e { tex.print(ltx.__tag.tables.#1["#2"]) }
46 }
47
48 %for debugging commands that show both the seq/prop and the lua tables
49 \cs_set_protected:Npn \__tag_seq_show:N #1
50 {
51   \seq_show:N #1
52   \lua_now:e { ltx.__tag.trace.log ("lua~sequence~array~\cs_to_str:N#1",1) }
53   \lua_now:e { ltx.__tag.trace.show_seq (ltx.__tag.tables.\cs_to_str:N#1) }
54 }
55
56 \cs_set_protected:Npn \__tag_prop_show:N #1
57 {
58   \prop_show:N #1
59   \lua_now:e {ltx.__tag.trace.log ("lua~property~table~\cs_to_str:N#1",1) }
60   \lua_now:e {ltx.__tag.trace.show_prop (ltx.__tag.tables.\cs_to_str:N#1) }
61 }

```

(End definition for __tag_prop_new:N and others.)

62 </luatex>

The module declaration

```

63 <lua>
64 -- tagpdf.lua
65 -- Ulrike Fischer
66
67 local ProvidesLuaModule = {
68   name      = "tagpdf",
69   version   = "0.93",      --TAGVERSION
70   date      = "2022-01-09", --TAGDATE
71   description = "tagpdf lua code",
72   license    = "The LATEX Project Public License 1.3c"
73 }
74
75 if luatexbase and luatexbase.provides_module then
76   luatexbase.provides_module (ProvidesLuaModule)
77 end
78
79 --[[

```

```

80 The code has quite probably a number of problems
81 - more variables should be local instead of global
82 - the naming is not always consistent due to the development of the code
83 - the traversing of the shipout box must be tested with more complicated setups
84 - it should probably handle more node types
85 -
86 --]]
87

```

Some comments about the lua structure.

```

88 --[[
89 the main table is named ltx.__tag. It contains the functions and also the data
90 collected during the compilation.
91
92 ltx.__tag.mc      will contain mc connected data.
93 ltx.__tag.struct will contain structure related data.
94 ltx.__tag.page    will contain page data
95 ltx.__tag.tables contains also data from mc and struct (from older code). This needs cleaning
96     There are certainly dublettes, but I don't dare yet ...
97 ltx.__tag.func    will contain (public) functions.
98 ltx.__tag.trace   will contain tracing/logging functions.
99 local funktions starts with __
100 functions meant for users will be in ltx.tag
101
102 functions
103 ltx.__tag.func.get_num_from (tag):    takes a tag (string) and returns the id number
104 ltx.__tag.func.output_num_from (tag): takes a tag (string) and prints (to tex) the id number
105 ltx.__tag.func.get_tag_from (num):    takes a num and returns the tag
106 ltx.__tag.func.output_tag_from (num): takes a num and prints (to tex) the tag
107 ltx.__tag.func.store_mc_data (num,key,data): stores key=data in ltx.__tag.mc[num]
108 ltx.__tag.func.store_mc_label (label,num): stores label=num in ltx.__tag.mc.labels
109 ltx.__tag.func.store_mc_kid (mcnum,kid,page): stores the mc-kids of mcnum on page page
110 ltx.__tag.func.store_mc_in_page(mcnum,mcpagecnt,page): stores in the page table the number of
111 ltx.__tag.func.store_struct_mcabs (structnum,mcnum): stores relations structnum<->mcnum (abs)
112 ltx.__tag.func.mc_insert_kids (mcnum): inserts the /K entries for mcnum by wandering through
113 ltx.__tag.func.mark_page_elements(box,mcpagecnt,mccntprev,mcopen,name,mctypeprev) : the main
114 ltx.__tag.func.mark_shipout (): a wrapper around the core function which inserts the last EM
115 ltx.__tag.func.fill_parent_tree_line (page): outputs the entries of the parenttree for this p
116 ltx.__tag.func.output_parenttree(): outputs the content of the parenttree
117 ltx.__tag.func.pdf_object_ref(name): outputs the object reference for the object name
118 ltx.__tag.func.markspaceon(), ltx.__tag.func.markspaceoff(): (de)activates the marking of pos
119 ltx.__tag.trace.show_mc_data (num,loglevel): shows ltx.__tag.mc[num] is the current log level
120 ltx.__tag.trace.show_all_mc_data (max,loglevel): shows a maximum about mc's if the current lo
121 ltx.__tag.trace.show_seq: shows a sequence (array)
122 ltx.__tag.trace.show_struct_data (num): shows data of structure num
123 ltx.__tag.trace.show_prop: shows a prop
124 ltx.__tag.trace.log
125 ltx.__tag.trace.showspace : boolean
126 --]]
127

```

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.

```

128 local mctypeattributeid = luatexbase.new_attribute ("g__tag_mc_type_attr")
129 local mcntattributeid   = luatexbase.new_attribute ("g__tag_mc_cnt_attr")
130 local iwspaceattributeid = luatexbase.new_attribute ("g__tag_interwordspace_attr")
131 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

```

132 local tagunmarkedbool= token.create("g__tag_tagunmarked_bool")
133 local truebool       = token.create("c_true_bool")

```

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

```

134 local catlatex      = luatexbase.registernumber("catcodetable@latex")
135 local tableinsert   = table.insert
136 local nodeid        = node.id
137 local nodecopy      = node.copy
138 local nodegetattribute = node.get_attribute
139 local nodesetattribute = node.set_attribute
140 local nodehasattribute = node.has_attribute
141 local nodenew       = node.new
142 local nodetail      = node.tail
143 local nodeslide     = node.slide
144 local noderemove    = node.remove
145 local nodetraverseid = node.traverse_id
146 local nodetraverse  = node.traverse
147 local nodeinsertafter = node.insert_after
148 local nodeinsertbefore = node.insert_before
149 local pdfpageref    = pdf.pageref
150
151 local HLIST          = node.id("hlist")
152 local VLIST          = node.id("vlist")
153 local RULE           = node.id("rule")
154 local DISC           = node.id("disc")
155 local GLUE           = node.id("glue")
156 local GLYPH          = node.id("glyph")
157 local KERN           = node.id("kern")
158 local PENALTY        = node.id("penalty")
159 local LOCAL_PAR      = node.id("local_par")
160 local MATH           = node.id("math")

```

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

```

161 ltx          = ltx          or { }
162 ltx.__tag    = ltx.__tag    or { }
163 ltx.__tag.mc  = ltx.__tag.mc or { } -- mc data
164 ltx.__tag.struct = ltx.__tag.struct or { } -- struct data
165 ltx.__tag.tables = ltx.__tag.tables or { } -- tables created with new prop and new seq.
166                                     -- wasn't a so great idea ...
167                                     -- g__tag_role_tags_seq used by tag<-> is in this table
168 ltx.__tag.page = ltx.__tag.page or { } -- page data, currently only i->{0->mcnum,1->mcnum}
169 ltx.__tag.trace = ltx.__tag.trace or { } -- show commands
170 ltx.__tag.func  = ltx.__tag.func  or { } -- functions

```



```
171 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`.

```
172 local __tag_log =
173   function (message,loglevel)
174     if (loglevel or 3) <= tex.count["l__tag_loglevel_int"] then
175       texio.write_nl("tagpdf: ".. message)
176     end
177   end
178
179 ltx.__tag.trace.log = __tag_log
```

(End 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.

```
180 function ltx.__tag.trace.show_seq (seq)
181   if (type(seq) == "table") then
182     for i,v in ipairs(seq) do
183       __tag_log ("[" .. i .. "] => " .. tostring(v),1)
184     end
185   else
186     __tag_log ("sequence " .. tostring(seq) .. " not found",1)
187   end
188 end
```

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

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

`ltx.__tag.trace.show_prop`

```
189 local __tag_pairs_prop =
190   function (prop)
191     local a = {}
192     for n in pairs(prop) do tableinsert(a, n) end
193     table.sort(a)
194     local i = 0                -- iterator variable
195     local iter = function ()   -- iterator function
196       i = i + 1
197       if a[i] == nil then return nil
198       else return a[i], prop[a[i]]
199     end
200   end
201   return iter
202 end
203
204
205 function ltx.__tag.trace.show_prop (prop)
206   if (type(prop) == "table") then
```

```

207   for i,v in __tag_pairs_prop (prop) do
208     __tag_log ("[" .. i .. "] => " .. tostring(v),1)
209   end
210 else
211   __tag_log ("prop " .. tostring(prop) .. " not found or not a table",1)
212 end
213 end

```

(End 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

```

214 function ltx.__tag.trace.show_mc_data (num,loglevel)
215   if ltx.__tag and ltx.__tag.mc and ltx.__tag.mc[num] then
216     for k,v in pairs(ltx.__tag.mc[num]) do
217       __tag_log ("mc"..num..": " ..tostring(k)..=>" ..tostring(v),loglevel)
218     end
219     if ltx.__tag.mc[num]["kids"] then
220       __tag_log ("mc" .. num .. " has " .. #ltx.__tag.mc[num]["kids"] .. " kids",loglevel)
221       for k,v in ipairs(ltx.__tag.mc[num]["kids"]) do
222         __tag_log ("mc " .. num .. " kid "..k.." =>" .. v.kid.." on page " ..v.page,loglevel)
223       end
224     end
225   else
226     __tag_log ("mc"..num.." not found",loglevel)
227   end
228 end

```

(End 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.

```

229 function ltx.__tag.trace.show_all_mc_data (min,max,loglevel)
230   for i = min, max do
231     ltx.__tag.trace.show_mc_data (i,loglevel)
232   end
233   texio.write_nl("")
234 end

```

(End 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.

```

235 function ltx.__tag.trace.show_struct_data (num)
236   if ltx.__tag and ltx.__tag.struct and ltx.__tag.struct[num] then
237     for k,v in ipairs(ltx.__tag.struct[num]) do
238       __tag_log ("struct "..num..": " ..tostring(k)..=>" ..tostring(v),1)
239     end
240   else
241     __tag_log ("struct "..num.." not found ",1)
242   end
243 end

```

(End 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 the tag (calculated from the mc-cnt).

```

244 local __tag_get_mc_cnt_type_tag = function (n)
245   local mcnt      = nodegetattribute(n,mcntattributeid) or -1
246   local mctype    = nodegetattribute(n,mctypeattributeid) or -1
247   local tag       = ltx.__tag.func.get_tag_from(mctype)
248   return mcnt,mctype,tag
249 end

```

(End 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.

```

250 local function __tag_get_mathsubtype (mathnode)
251   if mathnode.subtype == 0 then
252     subtype = "beginmath"
253   else
254     subtype = "endmath"
255   end
256   return subtype
257 end

```

(End definition for __tag_get_mathsubtype.)

`__tag_get_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, `ltx.__tag.func.get_num_from` while the output function outputs to tex, `ltx.__tag.func.output_num_from`.

```

258 local __tag_get_num_from =
259   function (tag)
260     if ltx.__tag.tables["g__tag_role_tags_prop"][tag] then
261       a = ltx.__tag.tables["g__tag_role_tags_prop"][tag]
262     else
263       a = -1
264     end
265     return a
266   end
267
268 ltx.__tag.func.get_num_from = __tag_get_num_from
269
270 function ltx.__tag.func.output_num_from (tag)
271   local num = __tag_get_num_from (tag)
272   tex.sprint(catlatex,num)
273   if num == -1 then
274     __tag_log ("Unknown tag "..tag.." used")
275   end
276 end

```

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

<pre> __tag_get_tag_from ltx.__tag.func.get_tag_from ltx.__tag.func.output_tag_from </pre>	<p>These functions are the opposites to the previous function: they take as argument a number (the attribute value) and return the string <code>tag</code>. The first function outputs the number for lua, while the <code>output</code> function outputs to tex.</p>
<pre>ltx.__tag.func.store_mc_data</pre>	<pre> 277 local __tag_get_tag_from = 278 function (num) 279 if ltx.__tag.tables["g__tag_role_tags_seq"][num] then 280 a = ltx.__tag.tables["g__tag_role_tags_seq"][num] 281 else 282 a= "UNKNOWN" 283 end 284 return a 285 end 286 287 ltx.__tag.func.get_tag_from = __tag_get_tag_from 288 289 function ltx.__tag.func.output_tag_from (num) 290 tex.sprint(catlatex,__tag_get_tag_from (num)) 291 end </pre> <p><i>(End definition for __tag_get_tag_from, ltx.__tag.func.get_tag_from, and ltx.__tag.func.output_tag_from.)</i></p> <p>This function stores for <code>key=data</code> for mc-chunk <code>num</code>. It is used in the tagpdf-mc code, to store for example the tag string, and the raw options.</p> <pre> 292 function ltx.__tag.func.store_mc_data (num,key,data) 293 ltx.__tag.mc[num] = ltx.__tag.mc[num] or { } 294 ltx.__tag.mc[num][key] = data 295 __tag_log ("INFO TEX-STORE-MC-DATA: "..num.." => "..tostring(key).. " => "..tostring(data),3) 296 end </pre> <p><i>(End definition for ltx.__tag.func.store_mc_data.)</i></p>
<pre>ltx.__tag.func.store_mc_label</pre>	<p>This function stores the <code>label=num</code> relationship in the <code>labels</code> subtable. TODO: this is probably unused and can go.</p> <pre> 297 function ltx.__tag.func.store_mc_label (label,num) 298 ltx.__tag.mc["labels"] = ltx.__tag.mc["labels"] or { } 299 ltx.__tag.mc.labels[label] = num 300 end </pre> <p><i>(End definition for ltx.__tag.func.store_mc_label.)</i></p>
<pre>ltx.__tag.func.store_mc_kid</pre>	<p>This function is used in the traversing code. It stores a sub-chunk of a mc <code>mcnum</code> into the <code>kids</code> table.</p> <pre> 301 function ltx.__tag.func.store_mc_kid (mcnum,kid,page) 302 ltx.__tag.trace.log("INFO TAG-STORE-MC-KID: "..mcnum.." => " .. kid.." on page " .. page,3) 303 ltx.__tag.mc[mcnum]["kids"] = ltx.__tag.mc[mcnum]["kids"] or { } 304 local kidtable = {kid=kid,page=page} 305 tableinsert(ltx.__tag.mc[mcnum]["kids"], kidtable) 306 end </pre> <p><i>(End definition for ltx.__tag.func.store_mc_kid.)</i></p>

ltx.__tag.func.mc_num_of_kids This function returns the number of kids a mc mcnun has. We need to account for the case that a mc can have no kids.

```

307 function ltx.__tag.func.mc_num_of_kids (mcnum)
308   local num = 0
309   if ltx.__tag.mc[mcnun] and ltx.__tag.mc[mcnun]["kids"] then
310     num = #ltx.__tag.mc[mcnun]["kids"]
311   end
312   ltx.__tag.trace.log ("INFO MC-KID-NUMBERS: " .. mcnun .. "has " .. num .. "KIDS",4)
313   return num
314 end

```

(End definition for ltx.__tag.func.mc_num_of_kids.)

3.2 Functions to insert the pdf literals

__tag_insert_emc_node This insert the emc node.

```

315 local function __tag_insert_emc_node (head,current)
316   local emcnode = nodenew("whatsit","pdf_literal")
317   emcnode.data = "EMC"
318   emcnode.mode=1
319   head = node.insert_before(head,current,emcnode)
320   return head
321 end

```

(End definition for __tag_insert_emc_node.)

__tag_insert_bmc_node This inserts a simple bmc node

```

322 local function __tag_insert_bmc_node (head,current,tag)
323   local bmcnode = nodenew("whatsit","pdf_literal")
324   bmcnode.data = "/"..tag.." BMC"
325   bmcnode.mode=1
326   head = node.insert_before(head,current,bmcnode)
327   return head
328 end

```

(End definition for __tag_insert_bmc_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.

```

329 local function __tag_insert_bdc_node (head,current,tag,dict)
330   local bdcnode = nodenew("whatsit","pdf_literal")
331   bdcnode.data = "/"..tag.."<<..dict..>> BDC"
332   bdcnode.mode=1
333   head = node.insert_before(head,current,bdcnode)
334   return head
335 end

```

(End definition for __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. TODO: is uses internal l3pdf commands, this should be properly supported by l3pdf

```

336 local function __tag_pdf_object_ref (name)
337   local tokename = 'c_pdf_backend_object_'..name..'_int'

```

```

338     local object = token.create(tokenname).index..' 0 R'
339     return object
340 end
341 ltx.__tag.func.pdf_object_ref=__tag_pdf_object_ref

(End definition for __tag_pdf_object_ref and ltx.__tag.func.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.

```

342 local function __tag_show_spacemark (head,current,color,height)
343     local markcolor = color or "1 0 0"
344     local markheight = height or 10
345     local pdfstring = node.new("whatsit","pdf_literal")
346     pdfstring.data =
347         string.format("q "..markcolor.." RG "..markcolor.." rg 0.4 w 0 %g m 0 %g l S Q",-
348             3,markheight)
349     head = node.insert_after(head,current,pdfstring)
350     return head
351 end

(End definition for __tag_show_spacemark.)

```

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

```

ltx.__tag.func.fakespace 351 local function __tag_fakespace()
352     tex.setattribute(iwspaceattributeid,1)
353     tex.setattribute(iwfontattributeid,font.current())
354 end
355 ltx.__tag.func.fakespace = __tag_fakespace

(End 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.

```

356 --[[ a function to mark up places where real space chars should be inserted
357     it only sets an attribute.
358 --]]
359
360 local function __tag_mark_spaces (head)
361     local inside_math = false
362     for n in nodetraverse(head) do
363         local id = n.id
364         if id == GLYPH then
365             local glyph = n
366             if glyph.next and (glyph.next.id == GLUE)
367                 and not inside_math and (glyph.next.width > 0)
368             then
369                 nodesetattribute(glyph.next,iwspaceattributeid,1)
370                 nodesetattribute(glyph.next,iwfontattributeid,glyph.font)
371             -- for debugging
372             if ltx.__tag.trace.showspace then

```

```

373     __tag_show_spacemark (head, glyph)
374   end
375   elseif glyph.next and (glyph.next.id == KERN) and not inside_math then
376     local kern = glyph.next
377     if kern.next and (kern.next.id == GLUE) and (kern.next.width > 0)
378     then
379       nodesetattribute(kern.next, iwspaceattributeid, 1)
380       nodesetattribute(kern.next, iwfontattributeid, glyph.font)
381     end
382   end
383   -- look also back
384   if glyph.prev and (glyph.prev.id == GLUE)
385     and not inside_math
386     and (glyph.prev.width > 0)
387     and not nodehasattribute(glyph.prev, iwspaceattributeid)
388   then
389     nodesetattribute(glyph.prev, iwspaceattributeid, 1)
390     nodesetattribute(glyph.prev, iwfontattributeid, glyph.font)
391   -- for debugging
392     if ltx.__tag.trace.showspace then
393       __tag_show_spacemark (head, glyph)
394     end
395   end
396   elseif id == PENALTY then
397     local glyph = n
398     -- ltx.__tag.trace.log ("PENALTY ".. n.subtype.."VALUE"..n.penalty, 3)
399     if glyph.next and (glyph.next.id == GLUE)
400       and not inside_math and (glyph.next.width > 0) and n.subtype == 0
401     then
402       nodesetattribute(glyph.next, iwspaceattributeid, 1)
403       -- nodesetattribute(glyph.next, iwfontattributeid, glyph.font)
404     -- for debugging
405       if ltx.__tag.trace.showspace then
406         __tag_show_spacemark (head, glyph)
407       end
408     end
409   elseif id == MATH then
410     inside_math = (n.subtype == 0)
411   end
412 end
413 return head
414 end

```

(End 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
415 local function __tag_activate_mark_space ()
416   if not luatexbase.in_callback ("pre_linebreak_filter", "markspaces") then
417     luatexbase.add_to_callback("pre_linebreak_filter", __tag_mark_spaces, "markspaces")
418     luatexbase.add_to_callback("hpack_filter", __tag_mark_spaces, "markspaces")
419   end
420 end
421

```

```
422 ltx.__tag.func.markspaceon=__tag_activate_mark_space
```

```
423
```

```
424 local function __tag_deactivate_mark_space ()
```

```
425   if luatexbase.in_callback ("pre_linebreak_filter","markspaces") then
```

```
426     luatexbase.remove_from_callback("pre_linebreak_filter","markspaces")
```

```
427     luatexbase.remove_from_callback("hpack_filter","markspaces")
```

```
428   end
```

```
429 end
```

```
430
```

```
431 ltx.__tag.func.markspaceoff=__tag_deactivate_mark_space
```

(End 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.

```
432 local default_space_char = node.new(GLYPH)
```

```
433 local default_fontid     = font.id("TU/lmr/m/n/10")
```

```
434 default_space_char.char  = 32
```

```
435 default_space_char.font  = default_fontid
```

```
--tag_space_chars_shipout
```

```
ltx.__tag.func.space_chars_shipout
```

These is the main function to insert real space chars. It inserts a glyph before every glue which has been marked previously. The attributes are copied from the glue, so if the tagging is done later, it will be tagged like it.

```
436 local function __tag_space_chars_shipout (box)
```

```
437   local head = box.head
```

```
438   if head then
```

```
439     for n in node.traverse(head) do
```

```
440       local spaceattr = nodegetattribute(n,iwspaceattributeid) or -1
```

```
441       if n.id == HLIST then -- enter the hlist
```

```
442         __tag_space_chars_shipout (n)
```

```
443       elseif n.id == VLIST then -- enter the vlist
```

```
444         __tag_space_chars_shipout (n)
```

```
445       elseif n.id == GLUE then
```

```
446         if ltx.__tag.trace.showspace and spaceattr==1 then
```

```
447           __tag_show_spacemark (head,n,"0 1 0")
```

```
448         end
```

```
449         if spaceattr==1 then
```

```
450           local space
```

```
451           local space_char = node.copy(default_space_char)
```

```
452           local curfont     = nodegetattribute(n,iwfontattributeid)
```

```
453           ltx.__tag.trace.log ("INFO SPACE-FUNCTION-FONT: ".. tostring(curfont),3)
```

```
454           if curfont and luaotfload.aux.slot_of_name(curfont,"space") then
```

```
455             space_char.font=curfont
```

```
456           end
```

```
457           head, space = node.insert_before(head, n, space_char) --
```

```
458           n.width     = n.width - space.width
```

```
459           space.attr  = n.attr
```

```
460         end
```

```
461       end
```

```
462     end
```

```
463   end
```

```
464 end
```

```
465
```

```
466 function ltx.__tag.func.space_chars_shipout (box)
```

```
467   __tag_space_chars_shipout (box)
```

```
468 end
```


(End 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.

```

469 function ltx.__tag.func.mc_insert_kids (mcnum,single)
470   if ltx.__tag.mc[mcnum] then
471     ltx.__tag.trace.log("INFO TEX-MC-INSERT-KID-TEST: " .. mcnum,4)
472     if ltx.__tag.mc[mcnum]["kids"] then
473       if #ltx.__tag.mc[mcnum]["kids"] > 1 and single==1 then
474         tex.sprint("(")
475       end
476       for i,kidstable in ipairs( ltx.__tag.mc[mcnum]["kids"] ) do
477         local kidnum = kidstable["kid"]
478         local kidpage = kidstable["page"]
479         local kidpageobjnum = pdfpageref(kidpage)
480         ltx.__tag.trace.log("INFO TEX-MC-INSERT-KID: " .. mcnum ..
481           " insert KID " .. i ..
482           " with num " .. kidnum ..
483           " on page " .. kidpage.."/"..kidpageobjnum,3)
484         tex.sprint(catlatex,"<</Type /MCR /Pg "..kidpageobjnum .. " 0 R /MCID "..kidnum.. ">> " )
485       end
486       if #ltx.__tag.mc[mcnum]["kids"] > 1 and single==1 then
487         tex.sprint("]")
488       end
489     else
490       -- this is typically not a problem, e.g. empty hbox in footer/header can
491       -- trigger this warning.
492       ltx.__tag.trace.log("WARN TEX-MC-INSERT-NO-KIDS: "..mcnum.." has no kids",2)
493       if single==1 then
494         tex.sprint("null")
495       end
496     end
497   else
498     ltx.__tag.trace.log("WARN TEX-MC-INSERT-MISSING: "..mcnum.." doesn't exist",0)
499   end
500 end

```

(End 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.

```

501 function ltx.__tag.func.store_struct_mcabs (structnum,mcnum)
502   ltx.__tag.struct[structnum]=ltx.__tag.struct[structnum] or { }
503   ltx.__tag.struct[structnum]["mc"]=ltx.__tag.struct[structnum]["mc"] or { }
504   -- a structure can contain more than on mc chunk, the content should be ordered
505   tableinsert(ltx.__tag.struct[structnum]["mc"],mcnum)
506   ltx.__tag.trace.log("INFO TEX-MC-INTO-STRUCT: " ..
507     mcnum.." inserted in struct "..structnum,3)
508   -- but every mc can only be in one structure

```

```

509 ltx.__tag.mc[mcnum]= ltx.__tag.mc[mcnum] or { }
510 ltx.__tag.mc[mcnum]["parent"] = structnum
511 end
512

```

(End 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.

```

513 -- pay attention: lua counts arrays from 1, tex pages from one
514 -- mcid and arrays in pdf count from 0.
515 function ltx.__tag.func.store_mc_in_page (mcnum,mcpagecnt,page)
516   ltx.__tag.page[page] = ltx.__tag.page[page] or {}
517   ltx.__tag.page[page][mcpagecnt] = mcnum
518   ltx.__tag.trace.log("INFO TAG-MC-INTO-PAGE: page " .. page ..
519                      ": inserting MCID " .. mcpagecnt .. " => " .. mcnum,3)
520 end

```

(End definition for ltx.__tag.func.store_mc_in_page.)

ltx.__tag.func.mark_page_elements This is the main traversing function. See the lua comment for more details.

```

521 --[[
522   Now follows the core function
523   It wades through the shipout box and checks the attributes
524   ARGUMENTS
525   box: is a box,
526   mcpagecnt: num, the current page cnt of mc (should start at -1 in shipout box), needed for
527   mcntprev: num, the attribute cnt of the previous node/whatever - if different we have a c
528   mcpopen: num, records if some bdc/emc is open
529   These arguments are only needed for log messages, if not present are replaces by fix string
530   name: string to describe the box
531   mctypeprev: num, the type attribute of the previous node/whatever
532
533   there are lots of logging messages currently. Should be cleaned up in due course.
534   One should also find ways to make the function shorter.
535 --]]
536
537 function ltx.__tag.func.mark_page_elements (box,mcpagecnt,mcntprev,mcpopen,name,mctypeprev)
538   local name = name or ("SOMEBBOX")
539   local mctypeprev = mctypeprev or -1
540   local abspage = status.total_pages + 1 -- the real counter is increased
541                                           -- inside the box so one off
542                                           -- if the callback is not used. (???)
543   ltx.__tag.trace.log ("INFO TAG-ABSPAGE: " .. abspage,3)
544   ltx.__tag.trace.log ("INFO TAG-ARGS: pagecnt".. mcpagecnt..
545                      " prev "..mcpopen..
546                      " type prev "..mctypeprev,4)
547   ltx.__tag.trace.log ("INFO TAG-TRAVERSING-BOX: ".. tostring(name)..
548                      " TYPE ".. node.type(node.getid(box)),3)
549   local head = box.head -- ShipoutBox is a vlist?
550   if head then
551     mcntthead, mctypehead,taghead = __tag_get_mc_cnt_type_tag (head)
552     ltx.__tag.trace.log ("INFO TAG-HEAD: " ..
553                        node.type(node.getid(head))..

```

```

554         " MC"..tostring(mccnthead)..
555         " => TAG " .. tostring(mctypehead)..
556         " => ".. tostring(taghead),3)
557     else
558         ltx.__tag.trace.log ("INFO TAG-NO-HEAD: head is "..
559                             tostring(head),3)
560     end
561     for n in node.traverse(head) do
562         local mccnt, mctype, tag = __tag_get_mc_cnt_type_tag (n)
563         local spaceattr = nodegetattribute(n,iwspaceattributeid) or -1
564         ltx.__tag.trace.log ("INFO TAG-NODE: "..
565                             node.type(node.getid(n))..
566                             " MC".. tostring(mccnt)..
567                             " => TAG " .. tostring(mctype)..
568                             " => " .. tostring(tag),3)
569         if n.id == HLIST
570         then -- enter the hlist
571             mcopy,mcpagecnt,mccntprev,mctypeprev=
572             ltx.__tag.func.mark_page_elements (n,mcpagecnt,mccntprev,mcopy,"INTERNAL HLIST",mctypeprev)
573         elseif n.id == VLIST then -- enter the vlist
574             mcopy,mcpagecnt,mccntprev,mctypeprev=
575             ltx.__tag.func.mark_page_elements (n,mcpagecnt,mccntprev,mcopy,"INTERNAL VLIST",mctypeprev)
576         elseif n.id == GLUE then -- at glue real space chars are inserted, but this has
577                                 -- been done if the previous shipout wandering, so here it
578         elseif n.id == LOCAL_PAR then -- local_par is ignored
579         elseif n.id == PENALTY then -- penalty is ignored
580         elseif n.id == KERN then -- kern is ignored
581             ltx.__tag.trace.log ("INFO TAG-KERN-SUBTYPE: "..
582                                 node.type(node.getid(n)).." "..n.subtype,4)
583         else
584             -- math is currently only logged.
585             -- we could mark the whole as math
586             -- for inner processing the mlist_to_hlist callback is probably needed.
587             if n.id == MATH then
588                 ltx.__tag.trace.log("INFO TAG-MATH-SUBTYPE: "..
589                                     node.type(node.getid(n)).." "..__tag_get_mathsubtype(n),4)
590             end
591             -- endmath
592             ltx.__tag.trace.log("INFO TAG-MC-COMPARE: current "..
593                                 mccnt.." prev "..mccntprev,4)
594             if mccnt~=mccntprev then -- a new mc chunk
595                 ltx.__tag.trace.log ("INFO TAG-NEW-MC-NODE: "..
596                                     node.type(node.getid(n))..
597                                     " MC"..tostring(mccnt)..
598                                     " <=> PREVIOUS "..tostring(mccntprev),4)
599                 if mcopy~=0 then -- there is a chunk open, close it (hope there is only one ...
600                     box.list=__tag_insert_emc_node (box.list,n)
601                     mcopy = mcopy - 1
602                     ltx.__tag.trace.log ("INFO TAG-INSERT-EMC: " ..
603                                         mcpagecnt .. " MCOPEN = " .. mcopy,3)
604                     if mcopy ~=0 then
605                         ltx.__tag.trace.log ("WARN TAG-OPEN-MC: " .. mcopy,1)
606                     end
607                 end

```

```

608     if ltx.__tag.mc[mccnt] then
609         if ltx.__tag.mc[mccnt]["artifact"] then
610             ltx.__tag.trace.log("INFO TAG-INSERT-ARTIFACT: "..
611                 tostring(ltx.__tag.mc[mccnt]["artifact"]),3)
612             if ltx.__tag.mc[mccnt]["artifact"] == "" then
613                 box.list = __tag_insert_bmc_node (box.list,n,"Artifact")
614             else
615                 box.list = __tag_insert_bdc_node (box.list,n,"Artifact", "/Type /"..ltx.__tag.mc[mccnt]
616             end
617         else
618             ltx.__tag.trace.log("INFO TAG-INSERT-TAG: "..
619                 tostring(tag),3)
620             mcpagecnt = mcpagecnt +1
621             ltx.__tag.trace.log ("INFO TAG-INSERT-BDC: "..mcpagecnt,3)
622             local dict= "/MCID "..mcpagecnt
623             if ltx.__tag.mc[mccnt]["raw"] then
624                 ltx.__tag.trace.log("INFO TAG-USE-RAW: "..
625                     tostring(ltx.__tag.mc[mccnt]["raw"]),3)
626                 dict= dict .. " " .. ltx.__tag.mc[mccnt]["raw"]
627             end
628             if ltx.__tag.mc[mccnt]["alt"] then
629                 ltx.__tag.trace.log("INFO TAG-USE-ALT: "..
630                     tostring(ltx.__tag.mc[mccnt]["alt"]),3)
631                 dict= dict .. " " .. ltx.__tag.mc[mccnt]["alt"]
632             end
633             if ltx.__tag.mc[mccnt]["actualtext"] then
634                 ltx.__tag.trace.log("INFO TAG-USE-ACTUALTEXT: "..
635                     tostring(ltx.__tag.mc[mccnt]["actualtext"]),3)
636                 dict= dict .. " " .. ltx.__tag.mc[mccnt]["actualtext"]
637             end
638             box.list = __tag_insert_bdc_node (box.list,n,tag, dict)
639             ltx.__tag.func.store_mc_kid (mccnt,mcpagecnt,abspage)
640             ltx.__tag.func.store_mc_in_page(mccnt,mcpagecnt,abspage)
641             ltx.__tag.trace.show_mc_data (mccnt,3)
642         end
643         mcopen = mcopen + 1
644     else
645         if tagunmarkedbool.mode == truebool.mode then
646             ltx.__tag.trace.log("INFO TAG-NOT-TAGGED: this has not been tagged, using artifact",2)
647             box.list = __tag_insert_bmc_node (box.list,n,"Artifact")
648             mcopen = mcopen + 1
649         else
650             ltx.__tag.trace.log("WARN TAG-NOT-TAGGED: this has not been tagged",1)
651         end
652     end
653     mcntprev = mccnt
654 end
655 end -- end if
656 end -- end for
657 if head then
658     mcnthead, mctypehead,taghead = __tag_get_mc_cnt_type_tag (head)
659     ltx.__tag.trace.log ("INFO TAG-ENDHEAD: " ..
660         node.type(node.getid(head))..
661         " MC"..tostring(mcnthead)..

```

```

662         " => TAG "..tostring(mctypehead)..
663         " => "..tostring(taghead),4)
664     else
665         ltx.__tag.trace.log ("INFO TAG-ENDHEAD: ".. tostring(head),4)
666     end
667     ltx.__tag.trace.log ("INFO TAG-QUITTING-BOX "..
668                         tostring(name)..
669                         " TYPE ".. node.type(node.getid(box)),4)
670     return mcopen,mcpagecnt,mccntprev,mctypeprev
671 end
672

```

(End 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.

```

673 function ltx.__tag.func.mark_shipout (box)
674     mcopen = ltx.__tag.func.mark_page_elements (box,-1,-100,0,"Shipout",-1)
675     if mcopen~=0 then -- there is a chunk open, close it (hope there is only one ...
676         local emcnode = nodenew("whatsit","pdf_literal")
677         local list = box.list
678         emcnode.data = "EMC"
679         emcnode.mode=1
680         if list then
681             list = node.insert_after (list,node.tail(list),emcnode)
682             mcopen = mcopen - 1
683             ltx.__tag.trace.log ("INFO SHIPOUT-INSERT-LAST-EMC: MCOPEN " .. mcopen,3)
684         else
685             ltx.__tag.trace.log ("WARN SHIPOUT-UPS: this shouldn't happen",0)
686         end
687         if mcopen ~=0 then
688             ltx.__tag.trace.log ("WARN SHIPOUT-MC-OPEN: " .. mcopen,1)
689         end
690     end
691 end

```

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

6 Parenttree

ltx.__tag.func.fill_parent_tree_line 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.

```

692 function ltx.__tag.func.fill_parent_tree_line (page)
693     -- we need to get page-> i=kid -> mcnum -> structnum
694     -- pay attention: the kid numbers and the page number in the parent tree start with 0!
695     local numsentry = ""
696     local pdfpage = page-1
697     if ltx.__tag.page[page] and ltx.__tag.page[page][0] then
698         mcchunks=#ltx.__tag.page[page]
699         ltx.__tag.trace.log("INFO PARENTTREE-NUM: page "..
700                             page.." has "..mcchunks.." +1 Elements ",4)
701         for i=0,mcchunks do

```

```

702 -- what does this log??
703 ltx.__tag.trace.log("INFO PARENTTREE-CHUNKS: "..
704 ltx.__tag.page[page][i],4)
705 end
706 if mcchunks == 0 then
707 -- only one chunk so no need for an array
708 local mcnum = ltx.__tag.page[page][0]
709 local structnum = ltx.__tag.mc[mcnum]["parent"]
710 local propname = "g__tag_struct"..structnum.."_prop"
711 --local objref = ltx.__tag.tables[propname]["objref"] or "XXXX"
712 local objref = __tag_pdf_object_ref('__tag/struct/'..structnum)
713 ltx.__tag.trace.log("INFO PARENTTREE-STRUCT-OBJREF: =====>"..
714 tostring(objref),5)
715 numsentry = pdfpage .. " [".. objref .. "]"
716 ltx.__tag.trace.log("INFO PARENTTREE-NUMENTRY: page " ..
717 page.. " num entry = ".. numsentry,3)
718 else
719 numsentry = pdfpage .. " ["
720 for i=0,mcchunks do
721 local mcnum = ltx.__tag.page[page][i]
722 local structnum = ltx.__tag.mc[mcnum]["parent"] or 0
723 local propname = "g__tag_struct"..structnum.."_prop"
724 --local objref = ltx.__tag.tables[propname]["objref"] or "XXXX"
725 local objref = __tag_pdf_object_ref('__tag/struct/'..structnum)
726 numsentry = numsentry .. " " .. objref
727 end
728 numsentry = numsentry .. "]"
729 ltx.__tag.trace.log("INFO PARENTTREE-NUMENTRY: page " ..
730 page.. " num entry = ".. numsentry,3)
731 end
732 else
733 ltx.__tag.trace.log ("INFO PARENTTREE-NO-DATA: page "..page,3)
734 end
735 return numsentry
736 end
737
738 function ltx.__tag.func.output_parenttree (abspage)
739 for i=1,abspage do
740 line = ltx.__tag.func.fill_parent_tree_line (i) .. "^^J"
741 tex.sprint(catlatex,line)
742 end
743 end

```

(End definition for ltx.__tag.func.fill_parent_tree_line and ltx.__tag.func.output_parenttree.)

```

744 </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)
```

This 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 type as it should then be used in `\tagstructbegin`.

namespace This is the namespace of the new type. The value should be a shorthand of a namespace. The allowed values are currently `pdf`, `pdf2`, `mathml` 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 type the tag should be mapped too. In a PDF 1.7 or earlier this is normally a type from the `pdf` set, in PDF 2.0 from the `pdf`, `pdf2` and `mathml` set. It can also be a user type, or a still unknown type. 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 type the default value is the default namespace of this type. If the role is unknown, `user` is used and the code hopes that the type will be defined later. With this key a specific namespace can be forced.

```
1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-roles-code} {2022-01-09} {0.93}
4 {part of tagpdf - code related to roles and structure names}
5 </header>
```

1 Code related to roles and structure names

1.1 Variables

Tags have both a name (a string) and a number (for the lua attribute). Testing a name is easier with a prop, while accessing with a number is better done with a seq. So both are used and must be kept in sync if a new tag is added. The number is only relevant for the MC type, tags with the same name from different names spaces can have the same number.

```

\g__tag_role_tags_seq
\g__tag_role_tags_prop
6 (*package)
7 \__tag_seq_new:N \g__tag_role_tags_seq %to get names (type/NS) from numbers
8 \__tag_prop_new:N \g__tag_role_tags_prop %to get numbers from names (type/NS)
(End definition for \g__tag_role_tags_seq and \g__tag_role_tags_prop.)

\g__tag_role_tags_NS_prop in pdf 2.0 tags belong to a name space. For every tag we store a default name space.
The keys are the tags, the value shorthands like pdf2, or mathml. There is no need to
access this from lua, so we use the standard prop commands.
9 \prop_new:N \g__tag_role_tags_NS_prop %to namespace info
(End definition for \g__tag_role_tags_NS_prop.)

\g__tag_role_NS_prop The standard names spaces are the following. The keys are the name tagpdf will use, the
urls are the identifier in the namespace 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)

More namespaces are possible and their objects references and the ones of the namespaces
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 also the object reference as it will be needed rather
often.
10 \prop_new:N \g__tag_role_NS_prop % collect namespaces
(End definition for \g__tag_role_NS_prop.)
We need also a bunch of temporary variables:

\l__tag_role_tag_tmpa_tl
\l__tag_role_tag_namespace_tmpa_tl
11 \tl_new:N \l__tag_role_tag_tmpa_tl
\l__tag_role_role_tmpa_tl
12 \tl_new:N \l__tag_role_tag_namespace_tmpa_tl
\l__tag_role_role_namespace_tmpa_tl
13 \tl_new:N \l__tag_role_role_tmpa_tl
14 \tl_new:N \l__tag_role_role_namespace_tmpa_tl
(End definition for \l__tag_role_tag_tmpa_tl and others.)

```

1.2 Namespaces

The following commands setups a names space. Namespace 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 prop to collect the rolemaps if needed.

```

\__tag_role_NS_new:nnn \__tag_role_NS_new:nnn{<shorthand>}{<URI-ID>}Schema

```

__tag_role_NS_new:nnn

```
15 \cs_new_protected:Npn \__tag_role_NS_new:nnn #1 #2 #3
16 {
17   \pdf_object_new:nn {tag/NS/#1}{dict}
18   \pdfdict_new:n      {g__tag_role/Namespace_#1_dict}
19   \pdf_object_new:nn {__tag/RoleMapNS/#1}{dict}
20   \pdfdict_new:n      {g__tag_role/RoleMapNS_#1_dict}
21   \pdfdict_gput:nnn
22     {g__tag_role/Namespace_#1_dict}
23     {Type}
24     {/Namespace}
25   \pdf_string_from_unicode:nnN{utf8/string}{#2}\l_tmpa_str
26   \tl_if_empty:NF \l_tmpa_str
27   {
28     \pdfdict_gput:nnx
29       {g__tag_role/Namespace_#1_dict}
30       {NS}
31       {\l_tmpa_str}
32   }
33   %RoleMapNS is added in tree
34   \tl_if_empty:NF {#3}
35   {
36     \pdfdict_gput:nnx{g__tag_role/Namespace_#1_dict}
37     {Schema}{#3}
38   }
39   \prop_gput:Nnx \g__tag_role_NS_prop {#1}{\pdf_object_ref:n{tag/NS/#1}~}
40 }
```

(End 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 not try to be really exact as it doesn't matter ...

\c__tag_role_userNS_id_str

```
41 \str_const:Nx \c__tag_role_userNS_id_str
42 { data:,
43   \int_to_Hex:n{\int_rand:n {65535}}
44   \int_to_Hex:n{\int_rand:n {65535}}
45   -
46   \int_to_Hex:n{\int_rand:n {65535}}
47   -
48   \int_to_Hex:n{\int_rand:n {65535}}
49   -
50   \int_to_Hex:n{\int_rand:n {65535}}
51   -
52   \int_to_Hex:n{\int_rand:n {16777215}}
53   \int_to_Hex:n{\int_rand:n {16777215}}
54 }
```

(End definition for \c__tag_role_userNS_id_str.)

Now we setup the standard names spaces. Currently only if we detect pdf2.0 but this will perhaps have to change if the structure code gets to messy.

```
55 \pdf_version_compare:NnT > {1.9}
56 {
```

```

57     \_tag_role_NS_new:nnn {pdf}    {http://iso.org/pdf/ssn}{ }
58     \_tag_role_NS_new:nnn {pdf2}  {http://iso.org/pdf2/ssn}{ }
59     \_tag_role_NS_new:nnn {mathml}{http://www.w3.org/1998/Math/MathML}{ }
60     \exp_args:Nnx
61     \_tag_role_NS_new:nnn {user}{\c_tag_role_userNS_id_str}{ }
62 }

```

1.3 Data

In this section we setup the standard data. At first the list of structure types. We split them in three lists, the tags with which are both in the pdf and pdf2 namespace, the one only in pdf and the one with the tags only in pdf2. We also define a rolemap for the pdfII only type to pdf so that they can always be used.

```

\c_tag_role_sttags_pdf_pdfII_clist
\c_tag_role_sttags_only_pdf_clist
\c_tag_role_sttags_only_pdfII_clist
\c_tag_role_sttags_mathml_clist
\c_tag_role_sttags_pdfII_to_pdf_prop
63 %
64 \clist_const:Nn \c_tag_role_sttags_pdf_pdfII_clist
65 {
66     Document,    %A complete document. This is the root element
67                  %of any structure tree containing
68                  %multiple parts or multiple articles.
69     Part,        %A large-scale division of a document.
70     Sect,        %A container for grouping related content elements.
71     Div,         %A generic block-level element or group of elements
72     Caption,     %A brief portion of text describing a table or figure.
73     Index,
74     NonStruct,   %probably not needed
75     H,
76     H1,
77     H2,
78     H3,
79     H4,
80     H5,
81     H6,
82     P,
83     L,           %list
84     LI,          %list item (around label and list item body)
85     Lbl,         %list label
86     LBody,       %list item body
87     Table,
88     TR,          %table row
89     TH,          %table header cell
90     TD,          %table data cell
91     THead,       %table header (n rows)
92     TBody,       %table rows
93     TFoot,       %table footer
94     Span,        %generic inline marker
95     Link,        %
96     Annot,
97     Figure,
98     Formula,
99     Form,
100    % ruby warichu etc ..
101    Ruby,

```

```

102     RB,
103     RT,
104     Warichu,
105     WT,
106     WP,
107     Artifact % only MC-tag ?...
108 }
109
110 \clist_const:Nn \c__tag_role_sttags_only_pdf_clist
111 {
112     Art,          %A relatively self-contained body of text
113                  %constituting a single narrative or exposition
114     BlockQuote,  %A portion of text consisting of one or more paragraphs
115                  %attributed to someone other than the author of the
116                  %surrounding text.
117     TOC,          %A list made up of table of contents item entries
118                  %%(structure tag TOCI; see below) and/or other
119                  %nested table of contents entries
120     TOCI,         %An individual member of a table of contents.
121                  %This entry's children can be any of the following structure tags:
122                  %Lbl,Reference,NonStruct,P,TOC
123     Index,
124     Private,
125     Quote,        %inline quote
126     Note,         %footnote, endnote. Lbl can be child
127     Reference,    %A citation to content elsewhere in the document.
128     BibEntry,     %bibentry
129     Code
130 }
131
132 \clist_const:Nn \c__tag_role_sttags_only_pdfII_clist
133 {
134     DocumentFragment
135     ,Aside
136     ,H7
137     ,H8
138     ,H9
139     ,H10
140     ,Title
141     ,FENote
142     ,Sub
143     ,Em
144     ,Strong
145     ,Artifact
146 }
147
148 \clist_const:Nn \c__tag_role_sttags_mathml_clist
149 {
150     abs
151     ,and
152     ,annotation
153     ,apply
154     ,approx
155     ,arccos

```

156 ,arccosh
 157 ,arccot
 158 ,arccoth
 159 ,arccsc
 160 ,arccsch
 161 ,arcsec
 162 ,arcsech
 163 ,arcsin
 164 ,arcsinh
 165 ,arctan
 166 ,arctanh
 167 ,arg
 168 ,bind
 169 ,bvar
 170 ,card
 171 ,cartesianproduct
 172 ,cbytes
 173 ,ceiling
 174 ,cerror
 175 ,ci
 176 ,cn
 177 ,codomain
 178 ,complexes
 179 ,compose
 180 ,condition
 181 ,conjugate
 182 ,cos
 183 ,cosh
 184 ,cot
 185 ,coth
 186 ,cs
 187 ,csc
 188 ,csch
 189 ,csymbol
 190 ,curl
 191 ,declare
 192 ,degree
 193 ,determinant
 194 ,diff
 195 ,divergence
 196 ,divide
 197 ,domain
 198 ,domainofapplication
 199 ,emptyset
 200 ,eq
 201 ,equivalent
 202 ,eulergamma
 203 ,exists
 204 ,exp
 205 ,exponentiale
 206 ,factorial
 207 ,factorof
 208 ,false
 209 ,floor

```

210 ,fn
211 ,forall
212 ,gcd
213 ,geq
214 ,grad
215 ,gt
216 ,ident
217 ,image
218 ,imaginary
219 ,imaginaryi
220 ,implies
221 ,in
222 ,infinity
223 ,int
224 ,integers
225 ,intersect
226 ,interval
227 ,inverse
228 ,lambda
229 ,laplacian
230 ,lcm
231 ,leq
232 ,limit
233 ,ln
234 ,log
235 ,logbase
236 ,lowlimit
237 ,lt
238 ,maction
239 ,maligngroup
240 ,malignmark
241 ,math
242 ,matrix
243 ,matrixrow
244 ,max
245 ,mean
246 ,median
247 ,menclose
248 ,merror
249 ,mfenced
250 ,mfrac
251 ,mglyph
252 ,mi
253 ,min
254 ,minus
255 ,mlabeledtr
256 ,mlongdiv
257 ,mmultiscripts
258 ,mn
259 ,mo
260 ,mode
261 ,moment
262 ,momentabout
263 ,mover

```

264 ,mpadded
 265 ,mphantom
 266 ,mprescripts
 267 ,mroot
 268 ,mrow
 269 ,ms
 270 ,mscarries
 271 ,mscarry
 272 ,msgroup
 273 ,msline
 274 ,mspace
 275 ,msqrt
 276 ,msrow
 277 ,mstack
 278 ,mstyle
 279 ,msub
 280 ,msubsup
 281 ,msup
 282 ,mtable
 283 ,mtd
 284 ,mtext
 285 ,mtr
 286 ,munder
 287 ,munderover
 288 ,naturalnumbers
 289 ,neq
 290 ,none
 291 ,not
 292 ,notanumber
 293 ,notin
 294 ,notprsubset
 295 ,notsubset
 296 ,or
 297 ,otherwise
 298 ,outerproduct
 299 ,partialdiff
 300 ,pi
 301 ,piece
 302 ,piecewise
 303 ,plus
 304 ,power
 305 ,primes
 306 ,product
 307 ,prsubset
 308 ,quotient
 309 ,rationals
 310 ,real
 311 ,reals
 312 ,reln
 313 ,rem
 314 ,root
 315 ,scalarproduct
 316 ,sdev
 317 ,sec

```

318     ,sech
319     ,selector
320     ,semantics
321     ,sep
322     ,set
323     ,setdiff
324     ,share
325     ,sin
326     ,sinh
327     ,subset
328     ,sum
329     ,tan
330     ,tanh
331     ,tendsto
332     ,times
333     ,transpose
334     ,true
335     ,union
336     ,uplimit
337     ,variance
338     ,vector
339     ,vectorproduct
340     ,xor
341 }
342
343 \prop_const_from_keyval:Nn \c__tag_role_sttags_pdfII_to_pdf_prop
344 {
345     DocumentFragment = Art,
346     Aside = Note,
347     Title = H1,
348     Sub   = Span,
349     H7    = H6 ,
350     H8    = H6 ,
351     H9    = H6 ,
352     H10   = H6 ,
353     FENote= Note,
354     Em    = Span,
355     Strong= Span,
356 }

```

(End definition for \c__tag_role_sttags_pdf_pdfII_clist and others.)

We fill the structure tags in to the seq. We allow all pdf1.7 and pdf2.0, and role map if needed the 2.0 tags.

```

357 % get tag name from number: \seq_item:Nn \g__tag_role_tags_seq { n }
358 % get tag number from name: \prop_item:Nn \g__tag_role_tags_prop { name }
359
360 \clist_map_inline:Nn \c__tag_role_sttags_pdf_pdfII_clist
361 {
362     \__tag_seq_gput_right:Nn \g__tag_role_tags_seq { #1 }
363     \prop_gput:Nnn \g__tag_role_tags_NS_prop { #1 }{ pdf2 }
364 }
365 \clist_map_inline:Nn \c__tag_role_sttags_only_pdf_clist
366 {
367     \__tag_seq_gput_right:Nn \g__tag_role_tags_seq { #1 }

```

```

368     \prop_gput:Nnn \g__tag_role_tags_NS_prop    { #1 }{ pdf }
369   }
370 \clist_map_inline:Nn \c__tag_role_sttags_only_pdfII_clist
371   {
372     \__tag_seq_gput_right:Nn \g__tag_role_tags_seq { #1 }
373     \prop_gput:Nnn \g__tag_role_tags_NS_prop    { #1 }{ pdf2 }
374   }
375 \pdf_version_compare:NnT > {1.9}
376   {
377     \clist_map_inline:Nn \c__tag_role_sttags_mathml_clist
378       {
379         \__tag_seq_gput_right:Nn \g__tag_role_tags_seq { #1 }
380         \prop_gput:Nnn \g__tag_role_tags_NS_prop    { #1 }{ mathml }
381       }
382   }

```

For luatex and the MC we need a name/number relation. The name space is not relevant.

```

383 \int_step_inline:nnnn { 1 }{ 1 }{ \seq_count:N \g__tag_role_tags_seq }
384   {
385     \__tag_prop_gput:Nxn \g__tag_role_tags_prop
386       {
387         \seq_item:Nn \g__tag_role_tags_seq { #1 }
388       }
389     { #1 }
390   }

```

1.4 Adding new tags and rolemapping

1.4.1 pdf 1.7 and earlier

With this versions only RoleMap is filled. At first the dictionary:

`g__tag_role/RoleMap_dict`

```

391 \pdfdict_new:n {g__tag_role/RoleMap_dict}

```

(End definition for `g__tag_role/RoleMap_dict`.)

`__tag_role_add_tag:nn` The pdf 1.7 version has only two arguments: new and rolemap name. To make pdf 2.0 types usable we directly define a rolemapping for them.

```

392 \cs_new_protected:Nn \__tag_role_add_tag:nn %(new) name, reference to old
393   {
394     \prop_if_in:NnF \g__tag_role_tags_prop {#1}
395       {
396         \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
397           {
398             \msg_info:nnn { tag }{new-tag}{#1}
399           }
400         \__tag_seq_gput_right:Nn \g__tag_role_tags_seq { #1 }
401         \__tag_prop_gput:Nnx \g__tag_role_tags_prop    { #1 }
402         {
403           \seq_count:N \g__tag_role_tags_seq
404         }
405         \prop_gput:Nnn \g__tag_role_tags_NS_prop    { #1 }{ user }
406       }
407     \__tag_check_add_tag_role:nn {#1}{#2}

```



```

408     \tl_if_empty:nF { #2 }
409     {
410         \pdfdict_gput:nxx {g__tag_role/RoleMap_dict}
411         {#1}
412         {\pdf_name_from_unicode_e:n{#2}}
413     }
414 }
415 \cs_generate_variant:Nn \__tag_role_add_tag:nn {VV}
416
417 \pdf_version_compare:NnT < {2.0}
418 {
419     \prop_map_inline:Nn \c__tag_role_sttags_pdfII_to_pdf_prop
420     {
421         \__tag_role_add_tag:nn {#1}{#2}
422     }
423 }
424

```

(End definition for __tag_role_add_tag:nn.)

1.4.2 The pdf 2.0 version

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

```

425 \cs_new_protected:Nn \__tag_role_add_tag:nnnn %tag/namespace/role/namespace
426 {
427     \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
428     {
429         \msg_info:nnn { tag } {new-tag}{#1}
430     }
431     \__tag_seq_gput_right:Nn \g__tag_role_tags_seq { #1 }
432     \__tag_prop_gput:Nnx \g__tag_role_tags_prop { #1 }
433     {
434         \seq_count:N \g__tag_role_tags_seq
435     }
436     \prop_gput:Nnn \g__tag_role_tags_NS_prop { #1 } { #2 }
437     \__tag_check_add_tag_role:nn {#1}{#3}
438     \pdfdict_gput:nxx {g__tag_role/RoleMapNS_#2_dict}{#1}
439     {
440         [
441             \pdf_name_from_unicode_e:n{#3}
442             \c_space_tl
443             \pdf_object_ref:n {tag/NS/#4}
444         ]
445     }
446 }
447 \cs_generate_variant:Nn \__tag_role_add_tag:nnnn {VVVV}

```

(End definition for __tag_role_add_tag:nnnn.)

1.5 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) 448 \keys_define:nn { __tag / tag-role }
role_(rolemap-key) 449 {
role-namespace_(rolemap-key) 450 ,tag .tl_set:N = \l__tag_role_tag_tmpa_tl
add-new-tag_(setup-key) 451 ,tag-namespace .tl_set:N = \l__tag_role_tag_namespace_tmpa_tl
452 ,role .tl_set:N = \l__tag_role_role_tmpa_tl
453 ,role-namespace .tl_set:N = \l__tag_role_role_namespace_tmpa_tl
454 }
455
456 \keys_define:nn { __tag / setup }
457 {
458   add-new-tag .code:n =
459   {
460     \keys_set_known:nnnN
461     {__tag/tag-role}
462     {
463       tag-namespace=user,
464       role-namespace=, %so that we can test for it.
465       #1
466     }{__tag/tag-role}\l_tmpa_tl
467     \tl_if_empty:NF \l_tmpa_tl
468     {
469       \exp_args:NNno \seq_set_split:Nnn \l_tmpa_seq { / } {\l_tmpa_tl/}
470       \tl_set:Nx \l__tag_role_tag_tmpa_tl { \seq_item:Nn \l_tmpa_seq {1} }
471       \tl_set:Nx \l__tag_role_role_tmpa_tl { \seq_item:Nn \l_tmpa_seq {2} }
472     }
473     \tl_if_empty:NT \l__tag_role_role_namespace_tmpa_tl
474     {
475       \prop_get:NVNTF
476       \g__tag_role_tags_NS_prop
477       \l__tag_role_role_tmpa_tl
478       \l__tag_role_role_namespace_tmpa_tl
479       {
480         \prop_if_in:NVF\g__tag_role_NS_prop \l__tag_role_role_namespace_tmpa_tl
481         {
482           \tl_set:Nn \l__tag_role_role_namespace_tmpa_tl {user}
483         }
484       }
485       {
486         \tl_set:Nn \l__tag_role_role_namespace_tmpa_tl {user}
487       }
488     }
489     \pdf_version_compare:NnTF < {2.0}
490     {
491       %TODO add check for emptyness?
492       \__tag_role_add_tag:VV
493       \l__tag_role_tag_tmpa_tl
494       \l__tag_role_role_tmpa_tl
495     }
496     {
497       \__tag_role_add_tag:VVVV
498       \l__tag_role_tag_tmpa_tl
499       \l__tag_role_tag_namespace_tmpa_tl
500       \l__tag_role_role_tmpa_tl

```

```

501         \l__tag_role_role_namespace_tmpa_tl
502     }
503 }
504 }
505 </package>

```

(End definition for tag (rolemap-key) and others. These functions are documented on page 119.)

Part X

The tagpdf-space module

Code related to real space chars

Part of the tagpdf package

`interwordspace_␣(setup-key)` This key allows to activate/deactivate the real space chars if the engine supports it. The allowed values are `true`, `on`, `false`, `off`.

`show-spaces_␣(setup-key)` 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} {2022-01-09} {0.93}
4 {part of tagpdf - code related to real space chars}
5 </header>
```

1 Code for interword spaces

The code is engine/backend dependant. 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.

```
interwordspace_␣(setup-key)
show-spaces_␣(setup-key)
6 (*package)
7 \sys_if_engine_pdftex:T
8 {
9   \sys_if_output_pdf:TF
10   {
11     \pdfglyphtounicode{space}{0020}
12     \keys_define:nn { __tag / setup }
13     {
14       interwordspace .choices:nn = { true, on } { \pdfinterwordspaceon },
15       interwordspace .choices:nn = { false, off } { \pdfinterwordspaceon },
16       interwordspace .default:n = true,
17       show-spaces .bool_set:N = \l__tag_showspaces_bool
18     }
19   }
20   {
21     \keys_define:nn { __tag / setup }
22     {
23       interwordspace .choices:nn = { true, on, false, off }
24       { \msg_warning:nnn {tag}{sys-no-interwordspace}{dvi} },
25       interwordspace .default:n = true,
```

```

26         show-spaces .bool_set:N = \l__tag_showspaces_bool
27     }
28 }
29 }
30
31
32 \sys_if_engine luatex:T
33 {
34     \keys_define:nn { __tag / setup }
35     {
36         interwordspace .choices:nn =
37             { true, on }
38             {
39                 \bool_gset_true:N \g__tag_active_space_bool
40                 \lua_now:e{ltx.__tag.func.markspaceon()}
41             },
42         interwordspace .choices:nn =
43             { false, off }
44             {
45                 \bool_gset_false:N \g__tag_active_space_bool
46                 \lua_now:e{ltx.__tag.func.markspaceoff()}
47             },
48         interwordspace .default:n = true,
49         show-spaces .choice:,
50         show-spaces / true .code:n =
51             {\lua_now:e{ltx.__tag.trace.showspaces=true}},
52         show-spaces / false .code:n =
53             {\lua_now:e{ltx.__tag.trace.showspaces=nil}},
54         show-spaces .default:n = true
55     }
56 }
57
58 \sys_if_engine xetex:T
59 {
60     \keys_define:nn { __tag / setup }
61     {
62         interwordspace .choices:nn = { true, on }
63         { \msg_warning:nnn {tag}{sys-no-interwordspace}{xetex} },
64         interwordspace .choices:nn = { false, off }
65         { \msg_warning:nnn {tag}{sys-no-interwordspace}{xetex} },
66         interwordspace .default:n = true,
67         show-spaces .bool_set:N = \l__tag_showspaces_bool
68     }
69 }

```

(End definition for `interwordspace` (setup-key) and `show-spaces` (setup-key). These functions are documented on page 132.)

`__tag_fakespace`: For luatex we need a command for the fake space as equivalent of the pdfTeX primitive.

```

70 \sys_if_engine luatex:T
71 {
72     \cs_new_protected:Nn \__tag_fakespace:
73     {
74         \group_begin:

```

```

75         \lua_now:e{ltx.__tag.func.fakespace()}
76         \skip_horizontal:n{\c_zero_skip}
77         \group_end:
78     }
79 }
80 \end{package}

(End definition for \__tag_fakespace:.)

```

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
<code>_</code>	232, 246
A	
<code>activate_</code> (setup-key)	29, <u>178</u>
<code>activate-all_</code> (setup-key)	6, <u>167</u>
<code>activate-mc_</code> (setup-key)	6, <u>167</u>
<code>activate-space_</code> (setup-key)	6, <u>167</u>
<code>activate-struct_</code> (setup-key)	6, <u>167</u>
<code>activate-tree_</code> (setup-key)	6, <u>167</u>
<code>actualtext_</code> (mc-key)	52, <u>198</u> , <u>392</u>
<code>actualtext_</code> (struct-key)	80, <u>294</u>
<code>add-new-tag_</code> (setup-key)	119, <u>448</u>
<code>\AddToHook</code>	13,
	16, 63, 193, 224, 238, 252, 263, 297
<code>AF_</code> (struct-key)	80, <u>397</u>
<code>AFinline_</code> (struct-key)	80, <u>397</u>
<code>AFinline-o_</code> (struct-key)	80, <u>397</u>
<code>alttext_</code> (mc-key)	52, <u>198</u> , <u>392</u>
<code>alttext_</code> (struct-key)	80, <u>294</u>
<code>artifact_</code> (mc-key)	52, <u>198</u> , <u>392</u>
artifact-bool internal commands:	
<code>__artifact-bool</code>	<u>102</u>
artifact-type internal commands:	
<code>__artifact-type</code>	<u>102</u>
<code>attr-unknown</code>	17, <u>33</u>
<code>attribute_</code> (struct-key)	81, <u>700</u>
<code>attribute-class_</code> (struct-key)	81, <u>666</u>
B	
<code>\begin</code>	32
bool commands:	
<code>\bool_gset_eq:Nn</code>	317, 330, 342, 358
<code>\bool_gset_false:N</code>	43, 45, 190, 318, 343, 380
<code>\bool_gset_true:N</code>	39, 42, 99, 160, 334
<code>\bool_if:NTF</code>	9, 9, 18, 23, 28, 33, 37, 69, 133,
	156, 169, 177, 182, 208, 212, 217,
	224, 225, 226, 230, 240, 244, 255,
	265, 312, 325, 336, 337, 353, 353, 548
<code>\bool_if:nTF</code>	6, 258
<code>\bool_lazy_all:nTF</code>	50, 195
<code>\bool_lazy_and:nnTF</code>	67, 77
<code>\bool_lazy_and_p:nn</code>	8
<code>\bool_new:N</code>	11, 15, 16, 41, 57, 94, 95,
	96, 97, 98, 100, 102, 104, 214, 215, 308
<code>\bool_set_false:N</code>	161, 163, 164, 191, 289, 311, 336
<code>\bool_set_true:N</code>	101, 103, 288
box commands:	
<code>\box_dp:N</code>	177, 181
<code>\box_ht:N</code>	167
<code>\box_new:N</code>	89, 90
<code>\box_set_dp:Nn</code>	175, 177
<code>\box_set_eq:NN</code>	190
<code>\box_set_ht:Nn</code>	174, 176
<code>\box_use_drop:N</code>	179, 183
<code>\boxmaxdepth</code>	63, 178
C	
<code>\c</code>	147, 148
c@g internal commands:	
<code>\c@_tag_MCID_abs_int</code>	9, 25, 34, 47, 54, 57, 65, 71,
	126, 135, 163, 237, 242, 271, 311, 355
<code>\c@_tag_parenttree_obj_int</code>	52
<code>\c@_tag_struct_abs_int</code>	46, 115, 118, 120, 305, 311, 324,
	336, 348, 360, 367, 380, 392, 423,
	425, 430, 441, 445, 446, 448, 449,
	451, 456, 465, 469, 470, 472, 473,
	475, 480, 518, 519, 520, 521, 524,
	527, 531, 544, 546, 552, 693, 696, 738
clist commands:	
<code>\clist_const:Nn</code>	64, 91, 92, 110, 132, 148
<code>\clist_map_inline:Nn</code>	360, 365, 370, 377
<code>\clist_map_inline:nn</code>	374
<code>\clist_new:N</code>	87
<code>\clist_set:Nn</code>	670, 704
color commands:	
<code>\color_select:n</code>	232, 246
cs commands:	
<code>\cs_generate_variant:Nn</code>	41, 98, 104,
	106, 107, 108, 108, 109, 110, 111,
	112, 113, 114, 125, 135, 140, 141,
	153, 153, 154, 155, 156, 157, 158,
	159, 234, 288, 299, 415, 416, 447, 639
<code>\cs_gset_eq:NN</code>	183
<code>\cs_if_exist:NTF</code>	65, 267, 299
<code>\cs_if_exist_p:N</code>	9, 199
<code>\cs_if_free:NTF</code>	39
<code>\cs_new:Nn</code>	21, 70, 74, 100, 122, 127, 131, 316

<code>\cs_new:Npn</code>	9, 47, 58, 67, 73, 91, 136, 141, 164, 197, 286, 640
<code>\cs_new_protected:Nn</code>	72, 154, 185, 367, 392, 425, 573, 605
<code>\cs_new_protected:Npn</code>	15, 20, 25, 29, 32, 42, 44, 47, 50, 57, 58, 60, 60, 63, 68, 69, 75, 81, 82, 85, 90, 93, 100, 101, 105, 109, 113, 116, 118, 123, 126, 129, 131, 142, 143, 147, 150, 150, 154, 154, 157, 160, 169, 178, 180, 185, 186, 187, 204, 210, 213, 235, 235, 252, 278, 278, 283, 285, 289, 292, 292, 293, 294, 295, 299, 300, 304, 308, 309, 315, 320, 322, 323, 330, 334, 338, 349, 397, 511, 630, 653
<code>\cs_set:Nn</code>	373, 374
<code>\cs_set:Npn</code>	38, 43
<code>\cs_set_eq:NN</code>	43, 46, 47, 48, 77, 78, 79, 145, 146, 147, 148, 149, 150, 151, 152, 166, 231, 232, 233, 366, 367, 368, 369, 375, 376, 380, 381, 382, 383
<code>\cs_set_protected:Nn</code>	183, 373, 574
<code>\cs_set_protected:Npn</code>	9, 16, 23, 30, 49, 56, 326, 512
<code>\cs_to_str:N</code>	12, 19, 26, 33, 52, 53, 59, 60
D	
<code>\DeclareDocumentMetadata</code>	21
<code>\DeclareOption</code>	42, 43
dim commands:	
<code>\c_max_dim</code>	166, 191
<code>\c_zero_dim</code>	174, 175, 176
<code>\documentclass</code>	22
E	
<code>E_␣(struct-key)</code>	80, 294
<code>\end</code>	29
<code>\endinput</code>	28
<code>\enquote</code>	31
<code>exclude-header-footer_␣(setup-key)</code>	31, 361
<code>\ExecuteOptions</code>	44
exp commands:	
<code>\exp_args:Ne</code>	288, 522
<code>\exp_args:Nee</code>	69
<code>\exp_args:NNno</code>	469
<code>\exp_args:NNnx</code>	51
<code>\exp_args:NNx</code>	51, 83, 86, 192, 212
<code>\exp_args:Nnx</code>	60, 274, 276, 280, 427
<code>\exp_args:NV</code>	173, 314, 339, 350
<code>\exp_args:Nx</code>	119, 241
<code>\exp_not:n</code>	276
F	
fi commands:	
<code>\fi</code>	19
file commands:	
<code>\file_input:n</code>	204
<code>\fontencoding</code>	6
<code>\fontfamily</code>	6
<code>\fontseries</code>	6
<code>\fontshape</code>	6
<code>\fontsize</code>	6
<code>\footins</code>	270
G	
group commands:	
<code>\group_begin:</code>	74, 158, 162, 332, 440, 464, 517
<code>\group_end:</code>	77, 166, 182, 358, 460, 484, 569
H	
hbox commands:	
<code>\hbox_set:Nn</code>	168, 169
hook commands:	
<code>\hook_gput_code:nnn</code>	7, 7, 21, 26, 50, 53, 180, 181, 223, 227, 388, 401, 411, 424
<code>\hook_use:n</code>	214
I	
if commands:	
<code>\if_mode_horizontal:</code>	19
<code>\ignorespaces</code>	29, 13
int commands:	
<code>\int_case:nnTF</code>	168
<code>\int_compare:nNnTF</code>	61, 73, 102, 113, 132, 159, 162, 170, 187, 193, 219, 254, 280, 287, 294, 301, 317, 325, 332, 340, 396, 427
<code>\int_compare:nTF</code>	77, 227, 686, 688, 690, 708, 734
<code>\int_eval:n</code>	88, 135, 232, 259, 276, 305, 311, 324, 336, 348, 360, 367, 380, 392, 423, 425, 430, 448, 456, 472, 480, 519, 520, 521, 524, 527, 531, 552, 693, 696, 738
<code>\int_gincr:N</code>	163, 228, 237, 242, 311, 518
<code>\int_gset:Nn</code>	55, 255
<code>\int_gzero:N</code>	8, 263
<code>\int_new:N</code>	10, 88, 93, 216, 217
<code>\int_rand:n</code>	43, 44, 46, 48, 50, 52, 53
<code>\int_set:Nn</code>	179, 182, 185, 186, 187
<code>\int_step_inline:nnnn</code>	46, 71, 74, 91, 212, 218, 383
<code>\int_to_Hex:n</code>	43, 44, 46, 48, 50, 52, 53

`\int_use:N` 9, 25, 34,
 47, 54, 56, 57, 65, 71, 115, 118, 120,
 124, 126, 126, 128, 232, 242, 246,
 259, 260, 271, 355, 441, 445, 446,
 449, 451, 465, 469, 470, 473, 475, 640
 intarray commands:
`\intarray_gset:Nnn` 190
`\intarray_item:Nn` 192, 195
`\intarray_new:Nn` 182
`interwordspace_␣(setup-key)` 132, 6
 iow commands:
`\iow_newline:` 171, 202
`\iow_now:Nn` 51
`\iow_term:n` 136, 139, 145, 149, 183, 195

K

keys commands:
`\keys_define:nn`
 .. 12, 21, 34, 54, 60, 66, 102, 128,
 167, 171, 183, 198, 218, 294, 361,
 393, 417, 448, 456, 488, 659, 666, 700
`\keys_set:nn` 9,
 51, 164, 187, 277, 281, 335, 428, 530
`\keys_set_known:nnnN` 460

L

`label_␣(mc-key)` 52, 198, 392
`label_␣(struct-key)` 80, 294
`lang_␣(struct-key)` 80, 294
 legacy commands:
`\legacy_if:nTF` 49
`\llap` 232
`log_␣(setup-key)` 6, 178
 ltx. internal commands:
`ltx.__tag.func.fakespace` 351
`ltx.__tag.func.fill_parent_tree_-
 line` 692
`ltx.__tag.func.get_num_from` ... 258
`ltx.__tag.func.get_tag_from` ... 277
`ltx.__tag.func.mark_page_-
 elements` 521
`ltx.__tag.func.mark_shipout` ... 673
`ltx.__tag.func.markspaceoff` ... 415
`ltx.__tag.func.markspaceon` ... 415
`ltx.__tag.func.mc_insert_kids` .. 469
`ltx.__tag.func.mc_num_of_kids` .. 307
`ltx.__tag.func.output_num_from` . 258
`ltx.__tag.func.output_parenttree` 692
`ltx.__tag.func.output_tag_from` . 277
`ltx.__tag.func.pdf_object_ref` .. 336
`ltx.__tag.func.space_chars_-
 shipout` 436
`ltx.__tag.func.store_mc_data` .. 292
`ltx.__tag.func.store_mc_in_page` 513

`ltx.__tag.func.store_mc_kid` ... 301
`ltx.__tag.func.store_mc_label` .. 297
`ltx.__tag.func.store_struct_-
 mcabs` 501
`ltx.__tag.trace.log` 172
`ltx.__tag.trace.show_all_mc_data` 229
`ltx.__tag.trace.show_mc_data` .. 214
`ltx.__tag.trace.show_prop` 189
`ltx.__tag.trace.show_seq` 180
`ltx.__tag.trace.show_struct_data` 235
 lua commands:
`\lua_now:n` ... 8, 11, 12, 19, 19, 26,
 28, 33, 35, 40, 40, 43, 45, 46, 51, 52,
 52, 53, 53, 59, 60, 60, 64, 75, 77, 77,
 78, 85, 87, 97, 102, 111, 122, 124,
 129, 140, 204, 212, 226, 243, 256, 266

M

`\maxdimen` 189
`mc-current` 17, 16
`mc-current_␣(show-key)` 30, 66
`mc-data_␣(show-key)` 30, 54
`mc-label-unknown` 17, 9
`mc-marks_␣(show-key)` 30, 128
`mc-nested` 17, 6
`mc-not-open` 17, 13
`mc-popped` 17, 14
`mc-pushed` 17, 14
`mc-tag-missing` 17, 8
`mc-used-twice` 17, 12
`\MessageBreak` 15, 19, 20, 21
 msg commands:
`\msg_error:nn` 90, 111, 262, 542
`\msg_error:nnn` 173, 249, 676, 714
`\msg_error:nnnn` 256
`\msg_info:nnn` . 104, 161, 165, 398, 429
`\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:nn` 124
`\msg_note:nnn` 296, 303, 334, 342
`\msg_note:nnnn` 282, 289, 319, 327
`\msg_warning:nn` 108
`\msg_warning:nnn`
 24, 36, 45, 63, 65, 97,
 120, 127, 138, 146, 154, 177, 200, 626

N

`namespace_␣(rolemap-key)` 119

new-tag 17, 37
 newattribute_␣(setup-key) 81, 653
 \newcommand 288, 289
 \newcounter 7, 8, 52
 \NewDocumentCommand 7,
 11, 17, 23, 28, 32, 37, 42, 49, 209, 290
 \newlabeldata 53
 \newmarks 14
 no-struct-dest_␣(setup-key) 6, 167
 \nointerlineskip 182

P

\PackageError 13
 \PackageWarning 28
 para-hook-count-wrong 18, 44
 paratagging_␣(setup-key) 31, 218
 paratagging-show_␣(setup-key) ... 31, 218
 pdf commands:
 \pdf_activate_structure_destination:
 199, 204
 \pdf_bdc:nn 233
 \pdf_bmc:n 231
 \l_pdf_current_structure_
 destination_tl 203
 \pdf_emc: 232
 \pdf_name_from_unicode_e:n
 307, 412, 441
 \pdf_object_if_exist:n 105
 \pdf_object_if_exist:nTF
 100, 118, 237, 421, 441, 465, 492
 \pdf_object_new:nn
 17, 19, 20, 51, 146, 176, 186, 523
 \pdf_object_ref:n 29,
 37, 39, 41, 90, 102, 106, 120, 121,
 183, 198, 280, 406, 413, 443, 555, 622
 \pdf_object_ref_last: 137, 723
 \pdf_object_unnamed_write:nn 129, 718
 \pdf_object_write:nn
 141, 149, 177, 193, 200, 205, 242
 \pdf_pageobject_ref:n 97
 \pdf_string_from_unicode:nnN ... 25
 \pdf_uncompress: 200
 \pdf_version_compare:NnTF
 55, 375, 417, 489
 \pdf_version_compare_p:Nn 200
 pdfannot commands:
 \pdfannot_dict_put:nnn
 107, 395, 418, 436, 441
 \pdfannot_link_ref_last: ... 405, 428
 pdfdict commands:
 \pdfdict_gput:nnn
 21, 28, 36, 197, 410, 438
 \pdfdict_if_empty:nTF 191
 \pdfdict_new:n 18, 20, 391
 \pdfdict_use:n 151, 195, 202
 \pdffakespace 31, 207
 pdf file commands:
 \pdffile_embed_stream:nnn
 108, 443, 467
 \pdfglyphtoupper 11
 \pdfinterwordspaceon 14, 15
 pdfmanagement commands:
 \pdfmanagement_add:nnn
 25, 26, 192, 194, 196, 229
 \pdfmanagement_if_active_p: .. 9, 10
 \pdfmanagement_remove:nn 198
 prg commands:
 \prg_do_nothing:
 79, 183, 380, 381, 382, 383
 \prg_generate_conditional_
 variant:Nnn 105
 \prg_new_conditional:Nnn 59, 222
 \prg_new_conditional:Npnn
 48, 65, 75, 250, 256, 267
 \prg_new_eq_conditional:NNn . 73, 229
 \prg_return_false:
 62, 69, 72, 82, 226, 253, 265, 271
 \prg_return_true:
 59, 69, 70, 79, 225, 254, 264, 270
 \ProcessOptions 45
 prop commands:
 \prop_clear:N 73
 \prop_const_from_keyval:Nn 343
 \prop_count:N 94
 \prop_get:NnNTF
 93, 96, 115, 130, 247, 308, 475
 \prop_gput:Nnn 25, 34, 37, 38,
 39, 95, 98, 100, 109, 147, 147, 295,
 363, 368, 373, 380, 405, 436, 655, 723
 \prop_if_exist:NnTF 609
 \prop_if_in:NnTF 60,
 87, 95, 175, 226, 394, 480, 674, 712, 716
 \prop_item:Nn 32, 64, 83, 111,
 150, 162, 230, 291, 300, 358, 721, 728
 \prop_map_inline:Nn 189, 419
 \prop_map_tokens:Nn 207
 \prop_new:N 9, 10, 11, 84, 145, 649, 652
 \prop_put:Nnn 80, 110
 \prop_show:N
 58, 92, 152, 563, 566, 696, 717
 \ProvidesExplFile 3
 \ProvidesExplPackage 3,
 3, 3, 3, 3, 3, 3, 3, 3, 3, 7, 26, 645

Q

\quad 158, 159

R

raw_␣(mc-key) 52, 198, 392

`ref_␣(struct-key)` 80, [294](#)
 ref commands:
 `\ref_attribute_gset:nnnn`
 114, 116, 123, 125, 127
 `\ref_label:nn` 111, 132
 `\ref_value:nn` 377
 `\ref_value:nnn` . 6, [65](#), 65, 67, 138, 143
 ref internal commands:
 `_ref_value:nnn` 70, 73
 regex commands:
 `\regex_replace_once:nnN` 146
`\RequirePackage` 20, 46, 210, 213, 219, 222
`\rlap` 246
`role_␣(rolemap-key)` [119](#), [448](#)
`role-missing` [17](#), [34](#)
`role-namespace_␣(rolemap-key)` .. [119](#), [448](#)
`role-tag` [17](#), [37](#)
`role-unknown` [17](#), [34](#)
`role-unknown-tag` [17](#), [34](#)
`root-AF_␣(setup-key)` [81](#), [488](#)

S

`\selectfont` 6
 seq commands:
 `\seq_clear:N` 158, 217
 `\seq_const_from_clist:Nn` 17, 30
 `\seq_count:N` 170,
 383, 403, 434, 686, 688, 690, 708, 734
 `\seq_get:NNTF` 258, 538, 588, 595
 `\seq_gpop:NN` 581
 `\seq_gpop:NNTF` 86, 582
 `\seq_gpop_left:NN` 145
 `\seq_gpush:Nn` . 12, 14, 69, 76, 544, 545
 `\seq_gput_left:Nn` 150, 678
 `\seq_gput_right:Nn` .. 32, 134, 148, 237
 `\seq_gremove_duplicates:N` 157
 `\seq_gset_eq:NN` 156, 165, 218
 `\seq_if_empty:NTF` 197
 `\seq_item:Nn` 113, 115,
 122, 126, 133, 137, 149, 189, 260,
 262, 269, 301, 302, 357, 387, 470, 471
 `\seq_log:N` . 166, 172, 174, 196, 320, 335
 `\seq_map_inline:Nn` . 159, 216, 672, 710
 `\seq_new:N` 11, 13, 13,
 15, 16, 17, 18, 18, 19, 85, 86, 146, 650
 `\seq_put_right:Nn` 160
 `\seq_remove_all:Nn` 163
 `\seq_set_eq:NN` 204, 205
 `\seq_set_from_clist:NN` 671, 705
 `\seq_set_from_clist:Nn` 84, 87, 193, 213
 `\seq_set_map:NNn` 158, 680
 `\seq_set_split:Nnn` 112, 300, 469
 `\seq_show:N` . 51, 141, 142, 151, 161,
 162, 164, 175, 247, 547, 564, 567, 577

`\seq_use:Nn`
 107, 108, 158, 159, 169, 201, 202, 687
`\l_tmpa_seq` 217, 237, 247, 469, 470, 471
 shipout commands:
 `\g_shipout_readonly_int`
 56, 124, 126, 232
`show-spaces_␣(setup-key)` [132](#), 6
`\ShowTagging` [15](#), [30](#), [49](#)
 skip commands:
 `\skip_horizontal:n` 76
 `\c_zero_skip` 76
`stash_␣(mc-key)` [52](#), [102](#)
`stash_␣(struct-key)` 80, [294](#)
`\stepcounter` 283
 str commands:
 `\str_const:Nn` 41
 `\str_if_eq:nnTF` 124, 269
 `\str_if_eq_p:nn` 260, 262
 `\str_new:N` 83
 `\str_set_convert:Nnnn` .. 113, 219,
 236, 318, 330, 342, 354, 386, 406, 416
 `\str_use:N` 230, 249
 `\l_tmpa_str` 25, 26, 31
`\string` 20, 21, 22, 278
`struct-faulty-nesting` [17](#), [23](#)
`struct-label-unknown` [17](#), [29](#)
`struct-missing-tag` [17](#), [26](#)
`struct-no-objnum` [17](#), [22](#)
`struct-show-closing` [17](#), [31](#)
`struct-stack_␣(show-key)` [30](#), [171](#)
`struct-used-twice` [17](#), [27](#)
 sys commands:
 `\sys_if_engine luatex:TF`
 ... 32, 42, 46, 47, 58, 70, 71, 202, 207
 `\sys_if_engine pdftex:TF` 7, 48
 `\sys_if_engine xetex:TF` 58
 `\sys_if_output_pdf:TF` 9, 11
`sys-no-interwordspace` [18](#), [41](#)

T

`tabsorder_␣(setup-key)` 6, [190](#)
`tag_␣(mc-key)` [52](#), [198](#), [392](#)
`tag_␣(rolemap-key)` [119](#), [448](#)
`tag_␣(struct-key)` [79](#), [294](#)
 tag commands:
 `\tag_get:n` [15](#), [90](#), [47](#), 47, 69, 72
 `\tag_if_active:` 48
 `\tag_if_active:TF` [15](#), [48](#)
 `\tag_if_active_p:` [15](#), [48](#)
 `\tag_mc_artifact_group_begin:n` ..
 [51](#), [50](#), 50
 `\tag_mc_artifact_group_end:`
 [51](#), [50](#), 57

\tag_mc_begin:n	9, 51, 13, 53, 94, 154, 154, 231, 235, 245, 318, 320, 320, 326, 346, 394, 417
\tag_mc_begin_pop:n	51, 61, 63, 82, 327, 355, 408, 431
\tag_mc_end:	51, 20, 60, 73, 185, 185, 233, 243, 247, 318, 324, 351, 367, 373, 406, 429
\tag_mc_end_push:	51, 52, 63, 63, 314, 339, 392, 415
\tag_mc_if_in:	73, 229
\tag_mc_if_in:TF	51, 30, 59, 222
\tag_mc_if_in_p:	51, 59, 222
\tag_mc_use:n	51, 25, 29, 29
\tag_stop_group_begin:	6, 54, 160, 160
\tag_stop_group_end:	6, 59, 160, 166
\tag_struct_begin:n	79, 34, 229, 345, 393, 416, 510, 511, 512
\tag_struct_end:	79, 39, 249, 352, 407, 430, 510, 573, 574
\tag_struct_insert_annot:nn	79, 97, 405, 428, 630, 630, 639
\tag_struct_parent_int:	79, 97, 398, 405, 421, 428, 630, 640
\tag_struct_use:n	79, 44, 604, 605
tag internal commands:	
__tag_activate_mark_space	415
\g__tag_active_mc_bool	33, 53, 67, 94, 170
\l__tag_active_mc_bool	56, 67, 100, 164
\g__tag_active_space_bool	9, 39, 45, 94, 169
\g__tag_active_struct_bool	52, 77, 94, 172, 198, 255
\l__tag_active_struct_bool	55, 77, 100, 163
\g__tag_active_struct_dest_bool	94, 176, 197
\g__tag_active_tree_bool	9, 23, 54, 94, 171, 212, 225
__tag_add_document_structure:n	178, 178, 189
__tag_add_missing_mcs:Nn	64, 164, 164, 216
__tag_add_missing_mcs_to_-stream:Nn	58, 58, 186, 186, 270, 274, 281, 283
\g__tag_attr_class_used_seq	157, 158, 648, 678
\g__tag_attr_entries_prop	163, 648, 655, 674, 712, 717, 721
__tag_attr_new_entry:nn	333, 653, 653, 663
\g__tag_attr_objref_prop	648, 716, 723, 728
\l__tag_attr_value_tl	648, 706, 725, 730, 732, 736, 740
__tag_check_add_tag_role:nn	123, 123, 407, 437
__tag_check_if_active_mc:	65
__tag_check_if_active_mc:TF	65, 84, 156, 187, 188, 322, 328, 369, 375
__tag_check_if_active_struct:	75
__tag_check_if_active_struct:TF	31, 65, 514, 515, 578, 579, 607, 633
__tag_check_if_mc_in_galley:	250
__tag_check_if_mc_in_galley:TF	134, 155
__tag_check_if_mc_tmb_missing:	256
__tag_check_if_mc_tmb_missing:TF	109, 143, 160, 256
__tag_check_if_mc_tmb_missing_p:	256
__tag_check_if_mc_tme_missing:	267
__tag_check_if_mc_tme_missing:TF	147, 152, 164, 267
__tag_check_if_mc_tme_missing_p:	267
__tag_check_info_closing_struct:n	100, 100, 108, 584
__tag_check_init_mc_used:	180, 180, 183, 189
__tag_check_mc_if_nested:	142, 142, 159, 333
__tag_check_mc_if_open:	142, 150, 189, 379
__tag_check_mc_in_galley:TF	250
__tag_check_mc_in_galley_p:	250
__tag_check_mc_pushed_popped:nn	70, 77, 90, 93, 98, 157, 157
__tag_check_mc_tag:N	166, 169, 169, 343
__tag_check_mc_used:n	133, 185, 185, 291
\g__tag_check_mc_used_intarray	180, 190, 192, 195
__tag_check_no_open_struct:	109, 109, 586, 593
__tag_check_show_MCID_by_page:	204, 204
__tag_check_struct_used:n	113, 113, 612
__tag_check_structure_has_tag:n	85, 85, 531
__tag_check_structure_tag:N	93, 93, 303

__tag_check_typeout_v:n	43, 43, 107, 108, 111,
		146, 154, 161, 183, 199, 208, 273, 278
__tag_debug_mc_begin_ignore:n	..	
	285, 362
__tag_debug_mc_begin_insert:n	..	
	278, 330
__tag_debug_mc_end_ignore:		299, 387
__tag_debug_mc_end_insert:		292, 377
__tag_debug_struct_begin_		
ignore:n	323, 571
__tag_debug_struct_begin_		
insert:n	315, 568
__tag_debug_struct_end_ignore:		
	338, 601
__tag_debug_struct_end_insert:		
	330, 599
__tag_exclude_headfoot_begin:	..	
	309, 366, 367
__tag_exclude_headfoot_end:	...	
	322, 368, 369
__tag_exclude_struct_headfoot_		
begin:n	334, 373, 374
__tag_exclude_struct_headfoot_		
end:	349, 375, 376
__tag_fakespace	351
__tag_fakespace:	70, 72, 211
__tag_finish_structure:	
	13, 16, 210, 210
__tag_get_data_mc_tag:	
	197, 197, 316, 316
__tag_get_data_struct_tag:		286, 286
__tag_get_mathsubtype	250
__tag_get_mc_abs_cnt:	9,
		9, 19, 20, 60, 90, 93, 101, 146, 154,
		173, 179, 206, 214, 230, 247, 260, 270
__tag_get_mc_cnt_type_tag	244
__tag_get_num_from	258
__tag_get_tag_from	277
__tag_hook_kernel_after_foot:	..	
	295, 304, 369, 376, 383
__tag_hook_kernel_after_head:	..	
	293, 302, 368, 375, 382
__tag_hook_kernel_before_foot:	..	
	294, 303, 367, 374, 381
__tag_hook_kernel_before_head:	..	
	292, 301, 366, 373, 380
__tag_in_mc_bool	
	11, 18, 160, 190, 224,
		317, 318, 330, 334, 342, 343, 358, 380
__tag_insert_bdc_node	329
__tag_insert_bmc_node	322
__tag_insert_emc_node	315
__tag_lastpagelabel:	47, 47, 64
__tag_log	172
__tag_loglevel_int	
	..	93, 102, 132, 160, 163, 170, 179,
		182, 185, 186, 187, 187, 280, 287,
		294, 301, 317, 325, 332, 340, 396, 427
__tag_mark_spaces	356
__tag_mc_artifact_begin_marks:n	20, 42, 78, 340
__tag_mc_artifact_bool	
	15, 105, 161, 169, 191, 336
__tag_mc_artifact_type_tl	
	14, 109, 113, 117,
		121, 125, 129, 133, 137, 283, 338, 340
__tag_mc_bdc:nn		230, 233, 234, 274, 306
__tag_mc_bdc_mcid:n	..	120, 235, 278
__tag_mc_bdc_mcid:nn	
	235, 235, 280, 285
__tag_mc_begin_marks:nn	
	20, 20, 41, 77, 347
__tag_mc_bmc:n	230, 231, 302
__tag_mc_bmc_artifact:		300, 300, 313
__tag_mc_bmc_artifact:n		300, 304, 314
__tag_mc_botmarks_seq	
	64, 18, 87, 108,
		142, 158, 159, 205, 213, 218, 252, 269
__tag_mc_disable_marks:	75, 75
__tag_mc_emc:	155, 230, 232, 382
__tag_mc_end_marks:	..	20, 60, 79, 383
__tag_mc_firstmarks_seq	
	64, 18, 84, 107, 141, 158,
		193, 196, 197, 204, 205, 252, 260, 262
__tag_mc_footnote_marks_seq	...	15
__tag_mc_get_marks:		81, 81, 133, 154
__tag_mc_handle_artifact:N	
	116, 300, 308, 338
__tag_mc_handle_mc_label:n	
	21, 21, 174, 351
__tag_mc_handle_mcid:nn	
	235, 283, 288, 344
__tag_mc_handle_stash:n	41,
		131, 131, 153, 179, 289, 289, 299, 355
__tag_mc_if_in:	59, 73, 222, 229
__tag_mc_if_in:TF		59, 67, 144, 152, 222
__tag_mc_if_in_p:	59, 222
__tag_mc_insert_extra_tmb:n	...	
	105, 105, 168
__tag_mc_insert_extra_tme:n	...	
	105, 150, 169
__tag_mc_insert_mcid_kids:n	...	
	122, 122, 138, 147
__tag_mc_insert_mcid_single_		
kids:n	122, 127, 148
__tag_mc_key_label_tl	
	..	17, 171, 174, 255, 347, 348, 351, 424

\l__tag_mc_key_properties_tl ...
 [17](#), [162](#), [211](#), [224](#), [225](#),
 [241](#), [242](#), [346](#), [402](#), [411](#), [412](#), [421](#), [422](#)
 \l__tag_mc_key_stash_bool
 [15](#), [28](#), [37](#), [104](#), [177](#), [353](#)
 \g__tag_mc_key_tag_tl
 .. [17](#), [19](#), [194](#), [197](#), [203](#), [316](#), [381](#), [398](#)
 \l__tag_mc_key_tag_tl [17](#),
 [166](#), [168](#), [193](#), [202](#), [343](#), [345](#), [347](#), [397](#)
 __tag_mc_lua_set_mc_type_attr:n
 [74](#), [74](#), [98](#), [168](#)
 __tag_mc_lua_unset_mc_type_-
 attr: [74](#), [100](#), [192](#)
 \g__tag_mc_main_marks_seq [15](#)
 \g__tag_mc_marks [14](#),
 [22](#), [31](#), [44](#), [51](#), [62](#), [68](#), [85](#), [88](#), [194](#), [214](#)
 \g__tag_mc_multicol_marks_seq ... [15](#)
 \g__tag_mc_parenttree_prop
 [12](#), [13](#), [83](#), [100](#), [148](#), [295](#)
 \l__tag_mc_ref_abspage_tl
 [12](#), [238](#), [250](#), [258](#), [266](#)
 __tag_mc_set_label_used:n [25](#), [25](#), [42](#)
 \g__tag_mc_stack_seq [13](#), [69](#), [76](#), [86](#), [166](#)
 __tag_mc_store:nnn . [90](#), [90](#), [104](#), [131](#)
 \l__tag_mc_tmpa_tl .. [13](#), [252](#), [255](#), [259](#)
 g__tag_MCID_abs_int [7](#)
 \g__tag_MCID_byabspage_prop
 [10](#), [248](#), [257](#), [265](#)
 \g__tag_MCID_tmp_bypage_int
 [10](#), [128](#), [255](#), [263](#), [276](#)
 \g__tag_mode_lua_bool
 [41](#), [42](#), [43](#), [69](#), [133](#), [156](#),
 [182](#), [208](#), [217](#), [265](#), [312](#), [325](#), [337](#), [353](#)
 __tag_new_output_prop_handler:n
 [58](#), [68](#), [77](#), [520](#)
 __tag_pairs_prop [189](#)
 \g__tag_para_begin_int
 [216](#), [228](#), [232](#), [254](#), [259](#)
 \l__tag_para_bool
 [214](#), [220](#), [226](#), [240](#), [288](#), [289](#), [311](#), [336](#)
 \g__tag_para_end_int
 [217](#), [242](#), [246](#), [254](#), [260](#)
 \g__tag_para_int [214](#)
 \l__tag_para_show_bool
 [214](#), [221](#), [230](#), [244](#)
 __tag_parenttree_add_objr:nn ...
 [60](#), [60](#), [275](#)
 \l__tag_parenttree_content_tl ...
 [67](#), [86](#), [98](#), [112](#), [120](#), [140](#), [143](#)
 \g__tag_parenttree_objr_tl [59](#), [62](#), [140](#)
 __tag_pdf_object_ref [336](#)
 __tag_prop_gput:Nnn
 [9](#), [23](#), [34](#), [38](#), [80](#), [87](#),
 [145](#), [147](#), [154](#), [180](#), [187](#), [198](#), [256](#),
 [264](#), [304](#), [310](#), [323](#), [335](#), [347](#), [359](#),
 [366](#), [379](#), [385](#), [391](#), [401](#), [424](#), [432](#),
 [450](#), [474](#), [495](#), [526](#), [551](#), [618](#), [692](#), [737](#)
 __tag_prop_item:Nn .. [9](#), [43](#), [145](#), [150](#)
 __tag_prop_new:N [8](#),
 [9](#), [9](#), [10](#), [11](#), [12](#), [76](#), [145](#), [145](#), [156](#), [519](#)
 __tag_prop_show:N [9](#), [56](#), [145](#), [152](#), [159](#)
 __tag_ref_label:nn
 [23](#), [129](#), [129](#), [135](#), [269](#), [535](#)
 __tag_ref_value:nnn
 [33](#), [78](#), [82](#), [97](#), [98](#), [116](#), [136](#),
 [136](#), [140](#), [229](#), [240](#), [240](#), [610](#), [616](#), [619](#)
 __tag_ref_value_lastpage:nn ...
 [57](#), [71](#), [74](#), [141](#), [141](#), [208](#), [222](#)
 \c__tag_refmc_clist [91](#)
 \c__tag_refstruct_clist [91](#)
 g__tag_role/RoleMap_dict [391](#)
 __tag_role_add_tag:nn
 [392](#), [392](#), [415](#), [421](#), [492](#)
 __tag_role_add_tag:nnnn
 [425](#), [425](#), [447](#), [497](#)
 __tag_role_NS_new:nnn
 [120](#), [15](#), [15](#), [57](#), [58](#), [59](#), [61](#)
 \g__tag_role_NS_prop
 [10](#), [39](#), [189](#), [207](#), [308](#), [480](#)
 \l__tag_role_role_namespace_-
 tmpa_tl [11](#),
 [453](#), [473](#), [478](#), [480](#), [482](#), [486](#), [501](#)
 \l__tag_role_role_tmpa_tl
 [11](#), [452](#), [471](#), [477](#), [494](#), [500](#)
 \c__tag_role_sttags_mathml_clist
 [63](#), [377](#)
 \c__tag_role_sttags_only_pdf_-
 clist [63](#), [365](#)
 \c__tag_role_sttags_only_pdfII_-
 clist [63](#), [370](#)
 \c__tag_role_sttags_pdf_pdfII_-
 clist [63](#), [360](#)
 \c__tag_role_sttags_pdfII_to_-
 pdf_prop [63](#), [419](#)
 \l__tag_role_tag_namespace_tmpa_-
 tl [11](#), [451](#), [499](#)
 \l__tag_role_tag_tmpa_tl
 [11](#), [450](#), [470](#), [493](#), [498](#)
 \g__tag_role_tags_NS_prop [9](#), [175](#),
 [300](#), [363](#), [368](#), [373](#), [380](#), [405](#), [436](#), [476](#)
 \g__tag_role_tags_prop
 ... [6](#), [95](#), [130](#), [358](#), [385](#), [394](#), [401](#), [432](#)
 \g__tag_role_tags_seq
 [6](#), [357](#), [362](#), [367](#),
 [372](#), [379](#), [383](#), [387](#), [400](#), [403](#), [431](#), [434](#)
 \c__tag_role_userNS_id_str
 [120](#), [41](#), [61](#)

\g__tag_saved_in_mc_bool	\g__tag_struct_tag_NS_tl 54, 302, 308
..... 308, 317, 330, 342, 358	\g__tag_struct_tag_stack_seq ...
__tag_seq_gput_right:Nn 13, 174, 175, 320, 335, 545, 581, 595
... 9, 30, 103, 108, 118, 134, 145,	\g__tag_struct_tag_tl
148, 155, 362, 367, 372, 379, 400, 431 54, 301, 303, 307, 545, 597
__tag_seq_item:Nn ... 9, 38, 145, 149	__tag_struct_write_obj:n
__tag_seq_new:N 42, 48, 235, 235
... 7, 9, 9, 16, 78, 145, 146, 157, 521	\g__tag_tagunmarked_bool ... 104, 188
__tag_seq_show:N . 9, 49, 145, 151, 158	\l__tag_tmpa_box
__tag_show_spacemark 82, 168, 174, 175, 179, 190, 191
\l__tag_showspace_bool ... 17, 26, 67	\l__tag_tmpa_clist
__tag_space_chars_shipout 82, 670, 671, 704, 705
g__tag_struct_0_prop	\l__tag_tmpa_int
..... 76 82
__tag_struct_add_AF:nn	\l__tag_tmpa_prop .. 73, 81, 82, 94, 96
..... 397, 416, 423, 447, 471, 494	\l__tag_tmpa_seq
\g__tag_struct_cont_mc_prop 82, 158, 158, 160, 162, 163, 164,
..... 10, 92, 93, 95, 98, 111	165, 170, 300, 301, 302, 671, 672,
\l__tag_struct_elem_stash_bool ..	680, 686, 688, 690, 705, 708, 710, 734
..... 57, 297, 549	\l__tag_tmpa_str
__tag_struct_exchange_kid_- 82, 220, 225, 230, 237, 242,
command:N	249, 319, 326, 331, 338, 343, 350,
..... 143, 143, 153, 184	355, 362, 387, 394, 407, 412, 417, 422
__tag_struct_fill_kid_key:n ...	\l__tag_tmpa_tl
..... 154, 154, 239	33, 34,
__tag_struct_get_dict_content:nN	41, 77, 82, 84, 86, 88, 93, 93, 94, 95,
..... 213, 213, 240	96, 97, 100, 102, 115, 116, 145, 149,
__tag_struct_insert_annot:nn ...	150, 156, 167, 174, 179, 206, 214,
..... 252, 252, 635	240, 245, 308, 313, 373, 376, 382,
\l__tag_struct_key_label_tl	581, 582, 588, 590, 595, 597, 684, 695
..... 56, 296, 533, 535	\l__tag_tmpb_box
__tag_struct_kid_mc_gput_- 82, 169, 176, 177, 181, 183
right:nn	\l__tag_tmpb_seq
..... 91, 101, 114, 292	82, 680, 687
__tag_struct_kid_OBJR_gput_-	__tag_tree_fill_parenttree: ...
right:nn 68, 69, 138
..... 126, 126, 141, 266	__tag_tree_lua_fill_parenttree:
__tag_struct_kid_struct_gput_- 118, 118, 135
right:nn ... 116, 116, 125, 560, 614	__tag_tree_write_classmap:
g__tag_struct_kids_0_seq 154, 154, 217
..... 76	__tag_tree_write_namespaces: ...
__tag_struct_mcid_dict:n 187, 187, 218
..... 91, 95, 98, 106	__tag_tree_write_parenttree: ...
\g__tag_struct_objR_seq 131, 131, 215
..... 9	__tag_tree_write_rolemap:
__tag_struct_output_prop_aux:nn 147, 147, 216
..... 58, 58, 72	__tag_tree_write_structelements:
\g__tag_struct_stack_current_tl 44, 44, 219
..... 15, 26, 35, 66, 72, 75, 136,	__tag_tree_write_structtreeroot:
144, 150, 203, 291, 293, 297, 546, 32, 32, 220
558, 562, 563, 566, 584, 590, 615, 622	tag-namespace_(rolemap-key)
\l__tag_struct_stack_parent_- 448
tmpa_tl	tag/struct/0 internal commands:
..... 15, 260,	__tag/struct/0
268, 280, 540, 555, 559, 561, 564, 567 20
\g__tag_struct_stack_seq	tag/tree/namespaces internal commands:
. 11, 259, 539, 544, 547, 577, 582, 588	__tag/tree/namespaces
\c__tag_struct_StructElem_- 186
entries_seq	tag/tree/parenttree internal commands:
..... 17	__tag/tree/parenttree
\c__tag_struct_StructTreeRoot_- 51
entries_seq	
..... 17	

tag/tree/rolemap internal commands:	
__tag/tree/rolemap	146
tagabspage	6, 114
tagmcabs	6, 114
\tagmcbegin	29, 11
\tagmcend	29, 11
tagmcid	6, 114
\tagmcifin	29
\tagmcifinTF	29, 28
\tagmcuse	29, 11
\tagpdfifluatexT	29, 46
\tagpdfifluatexTF	29, 46
\tagpdfifpdfTeX	29, 46
\tagpdfparaOff	31, 288
\tagpdfparaOn	31, 288
\tagpdfsetup	29, 81, 119, 6
\tagpdfsuppressmarks	31, 290
tagstruct	6, 114
\tagstructbegin	30, 119, 32, 180
\tagstructend	30, 32, 181
tagstructobj	6, 114
\tagstructuse	30, 32
tagunmarked _␣ (setup-key)	6, 188
T _E X and L ^A T _E X 2 _ε commands:	
\@M	165
\@auxout	51
\@bsphack	131
\@cclv	274
\@esphack	133
\@gobble	24, 48
\@ifpackageloaded	28
\@kernel@after@foot	304
\@kernel@after@head	302
\@kernel@before@cclv	271
\@kernel@before@foot	303
\@kernel@before@footins	267, 269
\@kernel@before@head	299, 301
\@makecol	273
\@maxdepth	178
\@mult@ptagging@hook	276
\@secondoftwo	24, 48
\c@page	273
\count@	281
\mult@firstbox	279
\mult@rightbox	283
\page@sofar	278
\process@cols	279
tex commands:	
\tex_botmarks:D	88
\tex_firstmarks:D	85
\tex_kern:D	181
\tex_marks:D	22, 31, 44, 51, 62, 68
\tex_splitbotmarks:D	214
\tex_splitfirstmarks:D	194
\the	273
\tiny	232, 246
title _␣ (struct-key)	80, 294
title-o _␣ (struct-key)	80, 294
tl commands:	
\c_space_tl	64, 64, 88, 89, 94, 96, 98, 104, 143, 160, 206, 230, 273, 442, 687, 727
\tl_clear:N	156, 162, 215, 373
\tl_gput_right:Nn	62, 404
\tl_gset:Nn	75, 194, 203, 301, 302, 381, 398, 411, 546, 590, 597
\tl_if_empty:NTF	26, 34, 171, 171, 173, 312, 348, 467, 473, 532
\tl_if_empty:nTF	34, 125, 408
\tl_if_eq:NNTF	252
\tl_if_eq:NnTF	88
\tl_if_exist:NTF	75, 399
\tl_new:N	11, 12, 12, 13, 13, 14, 14, 15, 16, 17, 18, 19, 20, 27, 54, 55, 56, 59, 67, 82, 409, 651
\tl_put_left:Nn	302, 304
\tl_put_right:Nn	86, 98, 111, 140, 211, 223, 224, 225, 241, 242, 269, 271, 276, 301, 303, 376, 402, 411, 412, 421, 422, 725, 732
\tl_set:Nn	33, 77, 109, 113, 115, 117, 120, 121, 125, 129, 133, 137, 167, 193, 202, 203, 206, 238, 255, 397, 470, 471, 482, 486, 684, 706
\tl_show:N	558, 559, 730, 736
\tl_tail:n	289
\tl_to_str:n	27, 39, 71, 275, 308
\tl_use:N	76, 429, 455, 479, 500
\l_tmpa_tl	118, 130, 466, 467, 469
token commands:	
\token_to_str:N	53, 273
tree-mcid-index-wrong	18, 39
U	
\unskip	29, 19
use commands:	
\use:N	47, 291
\use_ii:nn	207
\use_none:n	43, 78
\use_none:nn	77
V	
\vbadness	165, 189
vbox commands:	
\vbox_set_split_to_ht:Nnn	191
\vbox_set_to_ht:Nnn	167
\vbox_unpack_drop:N	180
\vfuzz	166