

tagpdf – A package to experiment with pdf tagging*

Ulrike Fischer[†]

Released 2022-08-24

Contents

1	Initialization and test if pdfmanagement is active.	7
2	Package options	7
3	Packages	8
4	Temporary code	8
4.1	Faster object writing	8
4.2	a LastPage label	9
5	Variables	10
6	Variants of l3 commands	11
7	Setup label attributes	11
8	Label commands	12
9	Commands to fill seq and prop	12
10	General tagging commands	13
11	Keys for tagpdfsetup	14
12	loading of engine/more dependent code	15
I	The tagpdf-checks module	
	Messages and check code	
	Part of the tagpdf package	17
1	Commands	17

*This file describes v0.97, last revised 2022-08-24.

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

2	Description of log messages	17
2.1	\ShowTagging command	17
2.2	Messages in checks and commands	17
2.3	Messages from the ptagging code	18
2.4	Warning messages from the lua-code	18
2.5	Info messages from the lua-code	18
2.6	Debug mode messages and code	19
2.7	Messages	19
3	Messages	20
3.1	Messages related to mc-chunks	20
3.2	Messages related to structures	21
3.3	Attributes	22
3.4	Roles	22
3.5	Miscellaneous	22
4	Retrieving data	23
5	User conditionals	23
6	Internal checks	23
6.1	checks for active tagging	23
6.2	Checks related to structures	24
6.3	Checks related to roles	25
6.4	Check related to mc-chunks	26
6.5	Checks related to the state of MC on a page or in a split stream	28
 II The tagpdf-user module		
Code related to L ^A T _E X2e user commands and document commands		
Part of the tagpdf package		31
1	Setup commands	31
2	Commands related to mc-chunks	31
3	Commands related to structures	32
4	Debugging	32
5	Extension commands	32
5.1	Fake space	33
5.2	Paratagging	33
5.3	Header and footer	33
5.4	Link tagging	34
6	User commands and extensions of document commands	34
7	Setup and preamble commands	34
8	Commands for the mc-chunks	34

9	Commands for the structure	35
10	Debugging	35
11	Commands to extend document commands	38
11.1	Document structure	38
11.2	Structure destinations	39
11.3	Fake space	39
11.4	Paratagging	40
11.5	Header and footer	42
11.6	Links	44
III	The <code>tagpdf-tree</code> module	
	Commands trees and main dictionaries	
	Part of the <code>tagpdf</code> package	46
1	Trees, pdfmanagement and finalization code	46
1.1	Catalog: MarkInfo and StructTreeRoot	46
1.2	Writing structure elements	47
1.3	ParentTree	47
1.4	Rolemap dictionary	50
1.5	Classmap dictionary	50
1.6	Namespaces	51
1.7	Finishing the structure	52
1.8	StructParents entry for Page	52
IV	The <code>tagpdf-mc-shared</code> module	
	Code related to Marked Content (mc-chunks), code shared by	
	all modes	
	Part of the <code>tagpdf</code> package	53
1	Public Commands	53
2	Public keys	54
3	Marked content code – shared	54
3.1	Variables and counters	55
3.2	Functions	56
3.3	Keys	58
V	The <code>tagpdf-mc-generic</code> module	
	Code related to Marked Content (mc-chunks), generic mode	
	Part of the <code>tagpdf</code> package	60

1	Marked content code – generic mode	60
1.1	Variables	60
1.2	Functions	61
1.3	Looking at MC marks in boxes	64
1.4	Keys	71
VI The tagpdf-mc-luacode module		
Code related to Marked Content (mc-chunks), luamode-specific		
Part of the tagpdf package		73
1	Marked content code – luamode code	73
1.1	Commands	74
1.2	Key definitions	78
VII The tagpdf-struct module		
Commands to create the structure		
Part of the tagpdf package		81
1	Public Commands	81
2	Public keys	82
2.1	Keys for the structure commands	82
2.2	Setup keys	84
3	Variables	84
3.1	Variables used by the keys	86
4	Commands	86
4.1	Initialization of the StructTreeRoot	87
4.2	Handlings kids	88
4.3	Output of the object	91
5	Keys	93
6	User commands	98
7	Attributes and attribute classes	102
7.1	Variables	102
7.2	Commands and keys	102
VIII The tagpdf-luatex.def		
Driver for luatex		
Part of the tagpdf package		106
1	Loading the lua	106
2	Logging functions	110

3	Helper functions	112
3.1	Retrieve data functions	112
3.2	Functions to insert the pdf literals	114
4	Function for the real space chars	115
5	Function for the tagging	118
6	Parenttree	122
IX	The tagpdf-roles module	
	Tags, roles and namespace code	
	Part of the tagpdf package	124
1	Code related to roles and structure names	124
1.1	Variables	124
1.2	Namespaces	125
1.3	Data	127
1.4	Adding new tags and rolemapping	133
1.4.1	pdf 1.7 and earlier	133
1.4.2	The pdf 2.0 version	134
1.5	Key-val user interface	134
X	The tagpdf-space module	
	Code related to real space chars	
	Part of the tagpdf package	137
1	Code for interword spaces	137
	Index	140

<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 commands to stop tagging in some places. There simply switches the two local booleans. The grouping commands can be used to group the effect.
<code>\tag_stop_group_end:</code>	
<code>\tag_stop:</code>	
<code>\tag_start:</code>	
<code>\tag_stop:n</code>	
<code>\tag_start:n</code>	

<code>\tag_stop:n</code>	<code>\tag_stop:n{<label>}</code>
<code>\tag_start:n</code>	<code>\tag_start:n{<label>}</code>

This commands are intended as a pair. The start command will only restart tagging if the previous stop command with the same label actually stopped tagging.

<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 luamode) unmarked text should be marked up as artifact. The initial value is true.
--	--

<code>tabsorder_<setup-key></code>	This sets the tabsorder 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} {2022-08-24} {0.97}
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\DocumentMetadata{<options>}\MessageBreak
22     before~\string\documentclass
23   }
24 }
25 </package>
26
27 <*debug>
28 \ProvidesExplPackage {tagpdf-debug} {2022-08-24} {0.97}
29 { debug code for tagpdf }
30 \Atifpackageloaded{tagpdf}{\PackageWarning{tagpdf-debug}{tagpdf~not~loaded,~quitting}\endinput}
31 </debug> We map the internal module name “tag” to “tagpdf” in messages.
32 <*package>
33 \prop_gput:Nnn \g_msg_module_name_prop { tag }{ tagpdf }
34 </package>
35
36 Debug mode has its special mapping:
37 <*debug>
38 \prop_gput:Nnn \g_msg_module_type_prop { tag / debug} {}
39 \prop_gput:Nnn \g_msg_module_name_prop { tag / debug }{tagpdf~DEBUG}
40 </debug>

```

2 Package options

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

```

36 <*package>
37 \bool_new:N\g__tag_mode_lua_bool
38 \DeclareOption {luamode} { \sys_if_engine luatex:T { \bool_gset_true:N \g__tag_mode_lua_bool } }
39 \DeclareOption {genericmode}{ \bool_gset_false:N\g__tag_mode_lua_bool }
40 \ExecuteOptions{luamode}
41 \ProcessOptions

```

3 Packages

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

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

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

```
43 \RequirePackage{tagpdf-base}
```

```
44 \</package>
```

```
45 \<*base>
```

```
46 \ProvidesExplPackage {tagpdf-base} {2022-08-24} {0.97}
```

```
47 {part of tagpdf - provide base, no-op versions of the user commands }
```

```
48 \</base>
```

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

```
49 \<*base>
```

```
50 \AddToHook{begindocument}
```

```
51 {
```

```
52   \str_case:VnF \c_sys_backend_str
```

```
53   {
```

```
54     { luatex } { \cs_new_protected:Npn \__tag_whatsits: {} }
```

```
55     { dvisvgm } { \cs_new_protected:Npn \__tag_whatsits: {} }
```

```
56   }
```

```
57   {
```

```
58     \cs_new_protected:Npn \__tag_whatsits: {\tex_special:D {}} }
```

```
59   }
```

```
60 }
```

```
61 \</base>
```

4 Temporary code

This is code which will be removed when proper support exists in LaTeX. It writes only dictionaries!

4.1 Faster object writing

```
62 \<*package>
```

```
63 \cs_if_free:NT \pdf_object_write:nnn
```

```
64 {
```

```
65   \cs_new_protected:Npn \pdf_object_new:n #1
```

```
66   { \pdf_object_new:nn{#1}{dict} }
```

```
67   \cs_new_protected:Npn \pdf_object_write:nnn #1#2#3
```

```
68   {
```

```
69     \pdf_object_write:nn {#1}{#3}
```

```
70   }
```

```
71   \str_if_eq:VnT \c_sys_backend_str {pdftex}
```

```
72   {
```

```
73     \cs_set_protected:Npn \pdf_object_write:nnn #1#2#3
```

```
74     {
```

```
75       \tex_immediate:D \tex_pdfobj:D
```

```
76       useobjnum ~
```

```
77       \int_use:c
```

```
78       { c__pdf_backend_object_ \tl_to_str:n {#1} _int }
```

```

79         { << ~ \exp_not:n {#3} ~ >> }
80     }
81 }
82 \str_if_eq:VnT \c_sys_backend_str {luatex}
83 {
84     \cs_set_protected:Npn \pdf_object_write:nnn #1#2#3
85     {
86         \tex_immediate:D \tex_pdfextension:D obj ~
87         useobjnum ~
88         \int_use:c
89         { c__pdf_backend_object_ \tl_to_str:n {#1} _int }
90         { << ~ \exp_not:n {#3} ~ >> }
91     }
92 }
93 }
94 \cs_generate_variant:Nn \pdf_object_write:nnn {nnx}
95
96 \endpackage

```

4.2 a LastPage label

See also issue #2 in Accessible-xref

`__tag_lastpagelabel:`

```

97 \begin{package}
98 \cs_new_protected:Npn \__tag_lastpagelabel:
99 {
100     \legacy_if:nT { @files }
101     {
102         \exp_args:NNnx \exp_args:NNx\iow_now:Nn \@auxout
103         {
104             \token_to_str:N \newlabeldata
105             {__tag_LastPage}
106             {
107                 {abspage} { \int_use:N \g_shipout_readonly_int }
108                 {tagmcabs}{ \int_use:N \c@g__tag_MCID_abs_int }
109             }
110         }
111     }
112 }
113
114 \AddToHook{enddocument/afterlastpage}
115 {\__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.

```

116 \cs_if_exist:NF \ref_value:nnn
117 {
118     \cs_new:Npn \ref_value:nnn #1#2#3
119     {
120         \exp_args:Nee
121         \__ref_value:nnn
122         { \tl_to_str:n {#1} } { \tl_to_str:n {#2} } {#3}
123     }

```

```

124     \cs_new:Npn \__ref_value:nnn #1#2#3
125     {
126         \tl_if_exist:cTF { g__ref_label_ #1 _ #2 _tl }
127         { \tl_use:c { g__ref_label_ #1 _ #2 _tl } }
128         {
129             #3
130         }
131     }
132 }

```

(End definition for `\ref_value:nnn`. This function is documented on page 6.)

5 Variables

<pre> \l__tag_tmpa_tl \l__tag_tmpl_tl\l__tag_tmpl_str \l__tag_tmpl_prop \l__tag_tmpl_seq \l__tag_tmpl_seq \l__tag_tmpl_clist \l__tag_tmpl_int \l__tag_tmpl_box \l__tag_tmpl_box </pre>	<p>A few temporary variables</p> <pre> 133 \tl_new:N \l__tag_tmpl_tl 134 \tl_new:N \l__tag_tmpl_tl 135 \str_new:N \l__tag_tmpl_str 136 \prop_new:N \l__tag_tmpl_prop 137 \seq_new:N \l__tag_tmpl_seq 138 \seq_new:N \l__tag_tmpl_seq 139 \clist_new:N \l__tag_tmpl_clist 140 \int_new:N \l__tag_tmpl_int 141 \box_new:N \l__tag_tmpl_box 142 \box_new:N \l__tag_tmpl_box </pre>
--	--

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

Attribute lists for the label command. We have a list for mc-related labels, and one for structures.

<pre> \c__tag_refmc_clist \c__tag_refstruct_clist </pre>	<pre> 143 \clist_const:Nn \c__tag_refmc_clist {tagabspage,tagmcabs,tagmcid} 144 \clist_const:Nn \c__tag_refstruct_clist {tagstruct,tagstructobj} </pre>
--	--

(End definition for `\c__tag_refmc_clist` and `\c__tag_refstruct_clist`.)

<pre> \l__tag_loglevel_int </pre>	<p>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.</p>
-----------------------------------	---

```

145 \int_new:N    \l__tag_loglevel_int

```

(End definition for `\l__tag_loglevel_int`.)

<pre> \g__tag_active_space_bool \g__tag_active_mc_bool \g__tag_active_tree_bool \g__tag_active_struct_bool \g__tag_active_struct_dest_bool </pre>	<p>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 <code>\tag_mc_begin:n</code>, 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.</p>
---	---

```

146 \bool_new:N   \g__tag_active_space_bool

```

```

147 \bool_new:N \g__tag_active_mc_bool
148 \bool_new:N \g__tag_active_tree_bool
149 \bool_new:N \g__tag_active_struct_bool
150 \bool_new:N \g__tag_active_struct_dest_bool
151 \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
\l__tag_active_struct_bool

```

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

```

152 \bool_new:N \l__tag_active_mc_bool
153 \bool_set_true:N \l__tag_active_mc_bool
154 \bool_new:N \l__tag_active_struct_bool
155 \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 of empty artifact mc-chunks.

```
156 \bool_new:N \g__tag_tagunmarked_bool
```

(End definition for \g__tag_tagunmarked_bool.)

6 Variants of l3 commands

```

157 \prg_generate_conditional_variant:Nnn \pdf_object_if_exist:n {e}{T,F}
158 \cs_generate_variant:Nn \pdf_object_ref:n {e}
159 \cs_generate_variant:Nn \pdfannot_dict_put:nnn {nnx}
160 \cs_generate_variant:Nn \pdffile_embed_stream:nnn {nxx,oxx}
161 \cs_generate_variant:Nn \prop_gput:Nnn {Nxx,Nen}
162 \cs_generate_variant:Nn \prop_put:Nnn {Nxx}
163 \cs_generate_variant:Nn \prop_item:Nn {No}
164 \cs_generate_variant:Nn \ref_label:nn {nv}
165 \cs_generate_variant:Nn \seq_set_split:Nnn{Nne}
166 \cs_generate_variant:Nn \str_set_convert:Nnnn {Nonn, Noon, Nnon }
167 \cs_generate_variant:Nn \clist_map_inline:nn {on}

```

7 Setup label attributes

```

tagstruct
tagstructobj
tagabspage
tagmcabs
tagmcid

```

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`.

```

168 \ref_attribute_gset:nnnn { tagstruct } {0} { now }
169 { \int_use:N \c@g__tag_struct_abs_int }
170 \ref_attribute_gset:nnnn { tagstructobj } {} { now }
171 {
172   \pdf_object_if_exist:eT {__tag/struct/\int_use:N \c@g__tag_struct_abs_int}

```

```

173     {
174     \pdf_object_ref:e{__tag/struct/\int_use:N \c@g__tag_struct_abs_int}
175     }
176   }
177 \ref_attribute_gset:nnnn { tagabspage } {0} { shipout }
178   { \int_use:N \g_shipout_readonly_int }
179 \ref_attribute_gset:nnnn { tagmcabs } {0} { now }
180   { \int_use:N \c@g__tag_MCID_abs_int }
181 \ref_attribute_gset:nnnn { tagmcid } {0} { now }
182   { \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.

```

183 \cs_new_protected:Npn \__tag_ref_label:nn #1 #2 %#1 label, #2 name of list mc or struct
184   {
185     \@bsphack
186     \ref_label:nv {#1}{c__tag_ref#2_clist}
187     \@esphack
188   }
189 \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.

```

190 \cs_new:Npn \__tag_ref_value:nnn #1 #2 #3 %#1 label, #2 attribute, #3 default
191   {
192     \ref_value:nnn {#1}{#2}{#3}
193   }
194 \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.

```

195 \cs_new:Npn \__tag_ref_value_lastpage:nn #1 #2
196   {
197     \ref_value:nnn {__tag_LastPage}{#1}{#2}
198   }

```

(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
    \__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
199 \cs_set_eq:NN \__tag_prop_new:N \prop_new:N
200 \cs_set_eq:NN \__tag_seq_new:N \seq_new:N
201 \cs_set_eq:NN \__tag_prop_gput:Nnn \prop_gput:Nnn
202 \cs_set_eq:NN \__tag_seq_gput_right:Nn \seq_gput_right:Nn
203 \cs_set_eq:NN \__tag_seq_item:cn \seq_item:cn
204 \cs_set_eq:NN \__tag_prop_item:cn \prop_item:cn
205 \cs_set_eq:NN \__tag_seq_show:N \seq_show:N
206 \cs_set_eq:NN \__tag_prop_show:N \prop_show:N
207
208 \cs_generate_variant:Nn \__tag_prop_gput:Nnn { Nx , Nxx, Nnx , cnn, cxn, cnx, cno }
209 \cs_generate_variant:Nn \__tag_seq_gput_right:Nn { Nx , No, cn, cx }
210 \cs_generate_variant:Nn \__tag_prop_new:N { c }
211 \cs_generate_variant:Nn \__tag_seq_new:N { c }
212 \cs_generate_variant:Nn \__tag_seq_show:N { c }
213 \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 commands to stop tagging in some places. This simply switches the two local booleans. In some cases tagging should only restart, if it actually was stopped before.

\tag_stop_group_end: For this it is possible to label a stop.

\tag_stop:

\tag_start:

\tag_stop:n

\tag_start:n

```

214 \cs_new_protected:Npn \tag_stop_group_begin:
215 {
216   \group_begin:
217   \bool_set_false:N \l__tag_active_struct_bool
218   \bool_set_false:N \l__tag_active_mc_bool
219 }
220 \cs_set_eq:NN \tag_stop_group_end: \group_end:
221 \cs_set_protected:Npn \tag_stop:
222 {
223   \bool_set_false:N \l__tag_active_struct_bool
224   \bool_set_false:N \l__tag_active_mc_bool
225 }
226 \cs_set_protected:Npn \tag_start:
227 {
228   \bool_set_true:N \l__tag_active_struct_bool
229   \bool_set_true:N \l__tag_active_mc_bool
230 }
231 \prop_new:N\g__tag_state_prop
232 \cs_set_protected:Npn \tag_stop:n #1
233 {
234   \tag_if_active:TF
235   {
236     \bool_set_false:N \l__tag_active_struct_bool
237     \bool_set_false:N \l__tag_active_mc_bool
238     \prop_gput:Nnn \g__tag_state_prop { #1 }{ 1 }
239   }
240   {
241     \prop_gremove:Nn \g__tag_state_prop { #1 }
242   }

```

```

243 }
244 \cs_set_protected:Npn \tag_start:n #1
245 {
246   \prop_gpop:NnN \g__tag_state_prop {#1}\l__tag_tmpa_tl
247   \quark_if_no_value:NF \l__tag_tmpa_tl
248   {
249     \bool_set_true:N \l__tag_active_struct_bool
250     \bool_set_true:N \l__tag_active_mc_bool
251   }
252 }
253 </package>
254 <*base>
255 \cs_new_protected:Npn \tag_stop:{}
256 \cs_new_protected:Npn \tag_start:{}
257 \cs_new_protected:Npn \tag_stop:n{}
258 \cs_new_protected:Npn \tag_start:n{}
259 </base>

```

(End definition for \tag_stop_group_begin: and others. 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 needed for interword spaces. It is not documented, the parsing is currently implicitly activated by the known key **interwordspace**, as the code will perhaps move to some other place, now that it is better separated. **no-struct-dest** allows to suppress structure destinations.

```

260 <*package>
261 \keys_define:nn { __tag / setup }
262 {
263   activate-space .bool_gset:N = \g__tag_active_space_bool,
264   activate-mc .bool_gset:N = \g__tag_active_mc_bool,
265   activate-tree .bool_gset:N = \g__tag_active_tree_bool,
266   activate-struct .bool_gset:N = \g__tag_active_struct_bool,
267   activate-all .meta:n =
268     {activate-mc={#1},activate-tree={#1},activate-struct={#1}},
269   activate-all .default:n = true,
270   no-struct-dest .bool_gset_inverse:N = \g__tag_active_struct_dest_bool,
271

```

(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.

```

272   log .choice:,
273   log / none .code:n = {\int_set:Nn \l__tag_loglevel_int { 0 }},
274   log / v .code:n =
275     {
276       \int_set:Nn \l__tag_loglevel_int { 1 }
277       \cs_set_protected:Nn \__tag_check_typeout_v:n { \iow_term:x {##1} }

```

```

278     },
279     log / vv      .code:n = {\int_set:Nn \l__tag_loglevel_int { 2 }},
280     log / vvv     .code:n = {\int_set:Nn \l__tag_loglevel_int { 3 }},
281     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.

```

282     tagunmarked .bool_gset:N = \g__tag_tagunmarked_bool,
283     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.

```

284     tabsorder .choice:,
285     tabsorder / row      .code:n =
286       \pdfmanagement_add:nnn { Page } {Tabs}{/R},
287     tabsorder / column   .code:n =
288       \pdfmanagement_add:nnn { Page } {Tabs}{/C},
289     tabsorder / structure .code:n =
290       \pdfmanagement_add:nnn { Page } {Tabs}{/S},
291     tabsorder / none     .code:n =
292       \pdfmanagement_remove:nn {Page} {Tabs},
293     tabsorder .initial:n = structure,
294     uncompress .code:n = { \pdf_uncompress: },
295 }

```

(End definition for `tabsorder` (setup-key). This function is documented on page 6.)

12 loading of engine/more dependent code

```

296 \sys_if_engine luatex:T
297 {
298   \file_input:n {tagpdf-luatex.def}
299 }
300 </package>
301 <*mcloading>
302 \bool_if:NTF \g__tag_mode_lua_bool
303 {
304   \RequirePackage {tagpdf-mc-code-lua}
305 }
306 {
307   \RequirePackage {tagpdf-mc-code-generic} %
308 }
309 </mcloading>
310 <*debug>
311 \bool_if:NTF \g__tag_mode_lua_bool
312 {
313   \RequirePackage {tagpdf-debug-lua}
314 }

```

```
315 {  
316   \RequirePackage {tagpdf-debug-generic} %  
317 }  
318 </debug>
```

Part I

The tagpdf-checks module

Messages and check code

Part of the tagpdf package

1 Commands

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

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

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

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-interwordspaces	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} {2022-08-24} {0.97} 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 19.)

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 19.)

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

```

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

```

(End definition for mc-label-unknown. This function is documented on page 19.)

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

```

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

```

(End definition for mc-used-twice. This function is documented on page 19.)

mc-not-open This is issued if a `\tag_mc_end`: is issued wrongly, wrong coding.

```

13 \msg_new:nnn { tag } {mc-not-open} { there-is-no-mc-to-end-at-#1 }

```

(End definition for mc-not-open. This function is documented on page 19.)

mc-pushed Informational messages about mc-pushing.
mc-popped 14 \msg_new:nnn { tag } {mc-pushed} { #1~has~been~pushed~to~the~mc~stack}
15 \msg_new:nnn { tag } {mc-popped} { #1~has~been~removed~from~the~mc~stack }
(End definition for mc-pushed and mc-popped. These functions are documented on page 19.)

mc-current Informational messages about current mc state.
16 \msg_new:nnn { tag } {mc-current}
17 { current-MC:~
18 \bool_if:NTF\g__tag_in_mc_bool
19 {abscnt=__tag_get_mc_abs_cnt:,~tag=\g__tag_mc_key_tag_tl}
20 {no-MC-open,~current-abs cnt=__tag_get_mc_abs_cnt:}
21 }
(End definition for mc-current. This function is documented on page 19.)

3.2 Messages related to structures

struct-unknown if for example a parent key value points to structure that doesn't exist (yet)
22 \msg_new:nnn { tag } {struct-unknown}
23 { structure~with~number~#1~doesn't~exist\\ #2 }
(End definition for struct-unknown. This function is documented on page ??.)

struct-no-objnum Should not happen ...
24 \msg_new:nnn { tag } {struct-no-objnum} { objnum~missing~for~structure~#1 }
(End definition for struct-no-objnum. This function is documented on page 19.)

struct-faulty-nesting This indicates that there is somewhere one \tag_struct_end: too much. This should be normally an error.
25 \msg_new:nnn { tag }
26 {struct-faulty-nesting}
27 { there~is~no~open~structure~on~the~stack }
(End definition for struct-faulty-nesting. This function is documented on page 19.)

struct-missing-tag A structure must have a tag.
28 \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 19.)

struct-used-twice
29 \msg_new:nnn { tag } {struct-used-twice}
30 { structure~with~label~#1~has~already~been~used }
(End definition for struct-used-twice. This function is documented on page 19.)

struct-label-unknown label is unknown, typically needs a rerun.
31 \msg_new:nnn { tag } {struct-label-unknown}
32 { structure~with~label~#1~is~unknown~rerun }
(End definition for struct-label-unknown. This function is documented on page 19.)

struct-show-closing Informational message shown if log-mode is high enough
33 \msg_new:nnn { tag } {struct-show-closing}
34 { 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 19.)

3.3 Attributes

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

attr-unknown

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

(End definition for attr-unknown. This function is documented on page 19.)

3.4 Roles

role-missing

Warning message if either the tag or the role is missing

role-unknown

role-unknown-tag

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

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

```
38 \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 19.)

role-tag

Info messages.

new-tag

```
39 \msg_new:nnn { tag } {role-tag} { mapping~tag~#1~to~role~#2 }
```

```
40 \msg_new:nnn { tag } {new-tag} { adding~new~tag~#1 }
```

(End definition for role-tag and new-tag. These functions are documented on page 19.)

3.5 Miscellaneous

tree-mcid-index-wrong

Used in the tree code, typically indicates the document must be rerun.

```
41 \msg_new:nnn { tag } {tree-mcid-index-wrong}
```

```
42 {something-is-wrong-with-the-mcid--rerun}
```

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

sys-no-interwordspace

Currently only pdf_lat_ex and lua_lat_ex have some support for real spaces.

```
43 \msg_new:nnn { tag } {sys-no-interwordspace}
```

```
44 {engine/output~mode~#1~doesn't~support~the~interword~spaces}
```

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

_tag_check_typeout_v:n

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

```
45 \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.

```
46 \msg_new:nnnn { tag } {para-hook-count-wrong}
```

```
47 {The-number-of-automatic-begin~(#1)-and-end~(#2)-para-hooks-differ!}
```

```
48 {This-quite-probably-a-coding-error-and-the-structure-will-be-wrong!}
```

```
49 </package>
```

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

4 Retrieving data

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

```
50 <base>\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 17.)

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.

```
51 <*base>
52 \prg_new_conditional:Npnn \tag_if_active: { p , T , TF, F }
53 { \prg_return_false: }
54 </base>
55 <*package>
56 \prg_set_conditional:Npnn \tag_if_active: { p , T , TF, F }
57 {
58     \bool_lazy_all:nTF
59     {
60         {\g__tag_active_struct_bool}
61         {\g__tag_active_mc_bool}
62         {\g__tag_active_tree_bool}
63         {\l__tag_active_struct_bool}
64         {\l__tag_active_mc_bool}
65     }
66     {
67         \prg_return_true:
68     }
69     {
70         \prg_return_false:
71     }
72 }
```

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

6 Internal checks

These are checks used in various places in the code.

6.1 checks for active tagging

__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.

```
73 \prg_new_conditional:Npnn \__tag_check_if_active_mc: { T,F,TF }
74 {
75     \bool_lazy_and:nnTF { \g__tag_active_mc_bool } { \l__tag_active_mc_bool }
76     {
77         \prg_return_true:
78     }
```

```

79     {
80         \prg_return_false:
81     }
82 }
83 \prg_new_conditional:Npnn \__tag_check_if_active_struct: {T,F,TF}
84 {
85     \bool_lazy_and:nnTF { \g__tag_active_struct_bool } { \l__tag_active_struct_bool }
86     {
87         \prg_return_true:
88     }
89     {
90         \prg_return_false:
91     }
92 }

```

(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.

```

93 \cs_new_protected:Npn \__tag_check_structure_has_tag:n #1 %#1 struct num
94 {
95     \prop_if_in:cnF { g__tag_struct_#1_prop }
96     {S}
97     {
98         \msg_error:nn { tag } {struct-missing-tag}
99     }
100 }

```

(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.

```

101 \cs_new_protected:Npn \__tag_check_structure_tag:N #1
102 {
103     \prop_if_in:NoF \g__tag_role_tags_prop {#1}
104     {
105         \msg_warning:nxx { tag } {role-unknown-tag} {#1}
106     }
107 }

```

(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.

```

108 \cs_new_protected:Npn \__tag_check_info_closing_struct:n #1 %#1 struct num
109 {
110     \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
111     {
112         \msg_info:nnn { tag } {struct-show-closing} {#1}
113     }
114 }
115
116 \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.

```

117 \cs_new_protected:Npn \_tag\_check\_no\_open\_struct:
118   {
119     \msg_error:nn { tag } {struct-faulty-nesting}
120   }

```

(End definition for `_tag_check_no_open_struct:.`)

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

```

121 \cs_new_protected:Npn \_tag\_check\_struct\_used:n #1 %#1 label
122   {
123     \prop_get:cnNT
124       {g\_tag\_struct\_\_tag\_ref\_value:enn{tagpdfstruct-#1}{tagstruct}{unknown}\_prop}
125       {P}
126     \l_tmpa_tl
127     {
128       \msg_warning:nnn { tag } {struct-used-twice} {#1}
129     }
130   }

```

(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.

```

131 \cs_new_protected:Npn \_tag\_check\_add\_tag\_role:nn #1 #2 %#1 tag, #2 role
132   {
133     \tl_if_empty:nTF {#2}
134     {
135       \msg_warning:nnn { tag } {role-missing} {#1}
136     }
137     {
138       \prop_get:NnNTF \g\_tag\_role\_tags\_prop {#2} \l_tmpa_tl
139       {
140         \int_compare:nNnT {\l\_tag\_loglevel\_int} > { 0 }
141         {
142           \msg_info:nnnn { tag } {role-tag} {#1} {#2}
143         }
144       }
145       {
146         \msg_warning:nnn { tag } {role-unknown} {#2}
147       }
148     }
149   }

```

(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).
`__tag_check_mc_if_open:`

```

150 \cs_new_protected:Npn __tag_check_mc_if_nested:
151 {
152   __tag_mc_if_in:T
153   {
154     \msg_warning:nnx { tag } {mc-nested} { __tag_get_mc_abs_cnt: }
155   }
156 }
157
158 \cs_new_protected:Npn __tag_check_mc_if_open:
159 {
160   __tag_mc_if_in:F
161   {
162     \msg_warning:nnx { tag } {mc-not-open} { __tag_get_mc_abs_cnt: }
163   }
164 }

```

(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.

```

165 \cs_new_protected:Npn __tag_check_mc_pushed_popped:nn #1 #2
166 {
167   \int_compare:nNtT
168     { \l__tag_loglevel_int } = { 2 }
169     { \msg_info:nnx {tag}{mc-#1}{#2} }
170   \int_compare:nNtT
171     { \l__tag_loglevel_int } > { 2 }
172     {
173       \msg_info:nnx {tag}{mc-#1}{#2}
174       \seq_log:N \g__tag_mc_stack_seq
175     }
176 }

```

(End definition for __tag_check_mc_pushed_popped:nn.)

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

```

177 \cs_new_protected:Npn __tag_check_mc_tag:N #1  % #1 is var with a tag name in it
178 {
179   \tl_if_empty:NT #1
180   {
181     \msg_error:nnx { tag } {mc-tag-missing} { __tag_get_mc_abs_cnt: }
182   }
183   \prop_if_in:Nof \g__tag_role_tags_NS_prop {#1}
184   {
185     \msg_warning:nnx { tag } {role-unknown-tag} {#1}
186   }
187 }

```

(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. `__tag_check_init_mc_used:` TODO does this really make sense to check? When can it happen??

```

188 \cs_new_protected:Npn \__tag_check_init_mc_used:
189 {
190   \intarray_new:Nn \g_tag_check_mc_used_intarray { 65536 }
191   \cs_gset_eq:NN \__tag_check_init_mc_used: \prg_do_nothing:
192 }

```

(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.

```

193 \cs_new_protected:Npn \__tag_check_mc_used:n #1 %#1 mcid abscont
194 {
195   \int_compare:nNnT {\l__tag_loglevel_int} > { 2 }
196   {
197     \__tag_check_init_mc_used:
198     \intarray_gset:Nnn \g_tag_check_mc_used_intarray
199       {#1}
200     { \intarray_item:Nn \g_tag_check_mc_used_intarray {#1} + 1 }
201     \int_compare:nNnT
202       {
203         \intarray_item:Nn \g_tag_check_mc_used_intarray {#1}
204       }
205       >
206       { 1 }
207     {
208       \msg_warning:nnn { tag } {mc-used-twice} {#1}
209     }
210   }
211 }

```

(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.

```

212 \cs_new_protected:Npn \__tag_check_show_MCID_by_page:
213 {
214   \tl_set:Nx \l__tag_tmpa_tl
215   {
216     \__tag_ref_value_lastpage:nn
217     {abspage}
218     {-1}
219   }
220   \int_step_inline:nnnn {1}{1}
221   {
222     \l__tag_tmpa_tl
223   }
224   {
225     \seq_clear:N \l_tmpa_seq
226     \int_step_inline:nnnn

```

```

227     {1}
228     {1}
229     {
230         \__tag_ref_value_lastpage:nn
231         {tagmcabs}
232         {-1}
233     }
234     {
235         \int_compare:nT
236         {
237             \__tag_ref_value:enn
238             {mcid-###1}
239             {tagabspage}
240             {-1}
241             =
242             ##1
243         }
244         {
245             \seq_gput_right:Nx \l_tmpa_seq
246             {
247                 Page##1-###1-
248                 \__tag_ref_value:enn
249                 {mcid-###1}
250                 {tagmcid}
251                 {-1}
252             }
253         }
254     }
255     \seq_show:N \l_tmpa_seq
256 }
257 }

```

(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.

```

258 \prg_new_conditional:Npnn \__tag_check_if_mc_in_galley: { T,F,TF }
259 {
260     \tl_if_eq:NNTF \l__tag_mc_firstmarks_seq \l__tag_mc_botmarks_seq
261     { \prg_return_false: }
262     { \prg_return_true: }
263 }

```

(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
`_tag_check_if_mc_tmb_missing:TF` the case if the firstmarks start with e- or b+. Like above we assume that the marks content is already in the seq’s.

```

264 \prg_new_conditional:Npnn \_tag\_check\_if\_mc\_tmb\_missing: { T,F,TF }
265 {
266   \bool\_if:nTF
267   {
268     \str\_if\_eq\_p:ee {\seq\_item:Nn \l\_tag\_mc\_firstmarks\_seq {1}}{e-}
269     ||
270     \str\_if\_eq\_p:ee {\seq\_item:Nn \l\_tag\_mc\_firstmarks\_seq {1}}{b+}
271   }
272   { \prg\_return\_true: }
273   { \prg\_return\_false: }
274 }

```

(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.

```

275 \prg_new_conditional:Npnn \_tag\_check\_if\_mc\_tme\_missing: { T,F,TF }
276 {
277   \str\_if\_eq:eeTF {\seq\_item:Nn \l\_tag\_mc\_botmarks\_seq {1}}{b+}
278   { \prg\_return\_true: }
279   { \prg\_return\_false: }
280 }

```

(End definition for _tag_check_if_mc_tme_missing:TF.)

281 `</package>`

282 `<*debug>`

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

```

283 \msg_new:nnn { tag / debug } {mc-begin} { MC~begin~#1~with~options:~\tl\_to\_str:n{#2}~[\msg\_lin
284 \msg_new:nnn { tag / debug } {mc-end} { MC~end~#1~[\msg\_line\_context:] }
285
286 \cs_new_protected:Npn \_tag\_debug\_mc\_begin\_insert:n #1
287 {
288   \int\_compare:nNnT { \l\_tag\_loglevel\_int } > {0}
289   {
290     \msg\_note:nnnn { tag / debug } {mc-begin} {inserted} { #1 }
291   }
292 }
293 \cs_new_protected:Npn \_tag\_debug\_mc\_begin\_ignore:n #1
294 {
295   \int\_compare:nNnT { \l\_tag\_loglevel\_int } > {0}
296   {
297     \msg\_note:nnnn { tag / debug } {mc-begin } {ignored} { #1 }
298   }
299 }
300 \cs_new_protected:Npn \_tag\_debug\_mc\_end\_insert:
301 {
302   \int\_compare:nNnT { \l\_tag\_loglevel\_int } > {0}

```

```

303     {
304         \msg_note:nnn { tag / debug } {mc-end} {inserted}
305     }
306 }
307 \cs_new_protected:Npn \__tag_debug_mc_end_ignore:
308 {
309     \int_compare:nNnT { \l__tag_loglevel_int } > {0}
310     {
311         \msg_note:nnn { tag / debug } {mc-end} {ignored}
312     }
313 }

```

And now something for the structures

```

314 \msg_new:nnn { tag / debug } {struct-begin}
315 {
316     Struct~\tag_get:n{struct_num}~begin~#1~with~options::~\tl_to_str:n{#2}~[\msg_line_context:]
317 }
318 \msg_new:nnn { tag / debug } {struct-end}
319 {
320     Struct~end~#1~[\msg_line_context:]
321 }
322
323 \cs_new_protected:Npn \__tag_debug_struct_begin_insert:n #1
324 {
325     \int_compare:nNnT { \l__tag_loglevel_int } > {0}
326     {
327         \msg_note:nnnn { tag / debug } {struct-begin} {inserted} { #1 }
328         \seq_log:N \g__tag_struct_tag_stack_seq
329     }
330 }
331 \cs_new_protected:Npn \__tag_debug_struct_begin_ignore:n #1
332 {
333     \int_compare:nNnT { \l__tag_loglevel_int } > {0}
334     {
335         \msg_note:nnnn { tag / debug } {struct-begin} {ignored} { #1 }
336     }
337 }
338 \cs_new_protected:Npn \__tag_debug_struct_end_insert:
339 {
340     \int_compare:nNnT { \l__tag_loglevel_int } > {0}
341     {
342         \msg_note:nnn { tag / debug } {struct-end} {inserted}
343         \seq_log:N \g__tag_struct_tag_stack_seq
344     }
345 }
346 \cs_new_protected:Npn \__tag_debug_struct_end_ignore:
347 {
348     \int_compare:nNnT { \l__tag_loglevel_int } > {0}
349     {
350         \msg_note:nnn { tag / debug } {struct-end} {ignored}
351     }
352 }
353 </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>\tagpdfifpdfTeX</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>	

`\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}%
```

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} {2022-08-24} {0.97}
4   {tagpdf - user commands}
5 </header>
```

7 Setup and preamble commands

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

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

8 Commands for the mc-chunks

```
\tagmcbegin
\tagmcend
\tagmcuse
13 <*base>
14 \NewDocumentCommand \tagmcbegin { m }
15   {
16     \tag_mc_begin:n {#1}
17   }
18
19
20 \NewDocumentCommand \tagmcend { }
21   {
22     \tag_mc_end:
23   }
24
25 \NewDocumentCommand \tagmcuse { m }
```

```

26 {
27   \tag_mc_use:n {#1}
28 }
29 \</base>

```

(End definition for \tagmcbegin, \tagmcend, and \tagmcuse. These functions are documented on page 31.)

\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.

```

30 \<package>
31 \NewDocumentCommand \tagmcifinTF { m m }
32 {
33   \tag_mc_if_in:TF { #1 } { #2 }
34 }
35 \</package>

```

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

9 Commands for the structure

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

```

36 \<base>
37 \NewDocumentCommand \tagstructbegin { m }
38 {
39   \tag_struct_begin:n {#1}
40 }
41
42 \NewDocumentCommand \tagstructend { }
43 {
44   \tag_struct_end:
45 }
46
47 \NewDocumentCommand \tagstructuse { m }
48 {
49   \tag_struct_use:n {#1}
50 }
51 \</base>

```

(End definition for \tagstructbegin, \tagstructend, and \tagstructuse. These functions are documented on page 32.)

10 Debugging

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

```

52 \<package>
53 \NewDocumentCommand\ShowTagging { m }
54 {
55   \keys_set:nn { __tag / show }{ #1}

```

```

56
57 }

```

(End definition for \ShowTagging. This function is documented on page 32.)

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.

```

58 \keys_define:nn { __tag / show }
59 {
60   mc-data .code:n =
61   {
62     \sys_if_engine luatex:T
63     {
64       \lua_now:e{!tx.__tag.trace.show_all_mc_data(#1,\__tag_get_mc_abs_cnt:,0)}
65     }
66   }
67   ,mc-data .default:n = 1
68 }
69

```

(End definition for mc-data (show-key). This function is documented on page 32.)

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

```

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

```

```

100     {
101         \lua_now:e
102         {
103             ltx.__tag.trace.log
104             (
105                 "mc-current:~absent=~\__tag_get_mc_abs_cnt:=="
106                 ..
107                 tex.getattribute(luatexbase.attributes.g__tag_mc_cnt_attr)
108                 ..
109                 "~=>tag="
110                 ..
111                 tostring
112                 (ltx.__tag.func.get_tag_from
113                 (tex.getattribute
114                 (luatexbase.attributes.g__tag_mc_type_attr)))
115                 ..
116                 "="
117                 ..
118                 tex.getattribute
119                 (luatexbase.attributes.g__tag_mc_type_attr)
120                 ,0
121             )
122             texio.write_nl("")
123         }
124     }
125 }
126
127 {
128     \msg_note:nn{ tag }{ mc-current }
129 }
130 }
131 }

```

(End definition for `mc-current` (`show-key`). This function is documented on page 32.)

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).

```

132 \keys_define:nn { __tag / show }
133 {
134     mc-marks .choice: ,
135     mc-marks / show .code:n =
136     {
137         \__tag_mc_get_marks:
138         \__tag_check_if_mc_in_galley:TF
139         {
140             \iow_term:n {Marks~from~this~page:~}
141         }
142         {
143             \iow_term:n {Marks~from~a~previous~page:~}
144         }
145         \seq_show:N \l__tag_mc_firstmarks_seq
146         \seq_show:N \l__tag_mc_botmarks_seq
147         \__tag_check_if_mc_tmb_missing:T
148         {

```

```

149         \iow_term:n {BDC~missing-on~this~page!}
150     }
151     \__tag_check_if_mc_tme_missing:T
152     {
153         \iow_term:n {EMC~missing-on~this~page!}
154     }
155 },
156 mc-marks / use .code:n =
157 {
158     \__tag_mc_get_marks:
159     \__tag_check_if_mc_in_galley:TF
160     { Marks~from~this~page:~}
161     { Marks~from~a~previous~page:~}
162     \seq_use:Nn \l__tag_mc_firstmarks_seq {,~}\quad
163     \seq_use:Nn \l__tag_mc_botmarks_seq {,~}\quad
164     \__tag_check_if_mc_tmb_missing:T
165     {
166         BDC~missing~
167     }
168     \__tag_check_if_mc_tme_missing:T
169     {
170         EMC~missing
171     }
172 },
173 mc-marks .default:n = show
174 }

```

(End definition for `mc-marks` (`show-key`). This function is documented on page 32.)

struct-stack_␣(show-key)

```

175 \keys_define:nn { __tag / show }
176 {
177     struct-stack .choice:
178     ,struct-stack / log .code:n = \seq_log:N \g__tag_struct_tag_stack_seq
179     ,struct-stack / show .code:n = \seq_show:N \g__tag_struct_tag_stack_seq
180     ,struct-stack .default:n = show
181 }

```

(End definition for `struct-stack` (`show-key`). This function is documented on page 32.)

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) 182 \cs_new_protected:Npn \__tag_add_document_structure:n #1
183 {
184     \hook_gput_code:nnn{begindocument}{tagpdf}{\tagstructbegin{tag=#1}}

```

```

185   \hook_gput_code:nnn{tagpdf/finish/before}{tagpdf}{\tagstructend}
186 }
187 \keys_define:nn { __tag / setup}
188 {
189   activate .code:n =
190   {
191     \keys_set:nn { __tag / setup }
192     { activate-mc,activate-tree,activate-struct }
193     \__tag_add_document_structure:n {#1}
194   },
195   activate .default:n = Document
196 }

```

(End definition for `__tag_add_document_structure:n` and `activate (setup-key)`. This function is documented on page 31.)

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.

```

197 \AddToHook{begindocument/before}
198 {
199   \bool_lazy_all:nT
200   {
201     { \g__tag_active_struct_dest_bool }
202     { \g__tag_active_struct_bool }
203     { \cs_if_exist_p:N \pdf_activate_structure_destination: }
204     { ! \pdf_version_compare_p:Nn < {2.0} }
205   }
206   {
207     \tl_set:Nn \l_pdf_current_structure_destination_tl { __tag/struct/\g__tag_struct_stack
208     \pdf_activate_structure_destination:
209   }
210 }

```

11.3 Fake space

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

```

211 \sys_if_engine_luaTeX:T
212 {
213   \NewDocumentCommand\pdffakespace { }
214   {
215     \__tag_fakespace:
216   }
217 }

```

(End definition for `\pdffakespace`. This function is documented on page 33.)

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 218 \bool_new:N \l__tag_para_bool
\g__tag_para_int      219 \bool_new:N \l__tag_para_show_bool
                      220 \int_new:N \g__tag_para_begin_int
                      221 \int_new:N \g__tag_para_end_int
                      222 \tl_new:N \l__tag_para_tag_tl
                      223 \tl_set:Nn \l__tag_para_tag_tl { P }
```

(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.

```
224 \keys_define:nn { __tag / setup }
225 {
226   paratagging      .bool_set:N = \l__tag_para_bool,
227   paratagging-show .bool_set:N = \l__tag_para_show_bool,
228   paratag          .tl_set:N   = \l__tag_para_tag_tl
229 }
230
```

(End definition for `paratagging (setup-key)` and `paratagging-show (setup-key)`. These functions are documented on page 33.)

This fills the para hooks with the needed code.

```
231 \AddToHook{para/begin}
232 {
233   \bool_if:NT \l__tag_para_bool
234   {
235     \int_gincr:N \g__tag_para_begin_int
236     \tag_struct_begin:n {tag=\l__tag_para_tag_tl}
237     \bool_if:NT \l__tag_para_show_bool
238     { \tag_mc_begin:n{artifact}
239       \llap{\color_select:n{red}\tiny\int_use:N\g__tag_para_begin_int\ }
240       \tag_mc_end:
241     }
242     \tag_mc_begin:n {tag=\l__tag_para_tag_tl}
243   }
244 }
245 \AddToHook{para/end}
246 {
247   \bool_if:NT \l__tag_para_bool
248   {
249     \int_gincr:N \g__tag_para_end_int
250     \tag_mc_end:
251     \bool_if:NT \l__tag_para_show_bool
252     { \tag_mc_begin:n{artifact}
253       \rlap{\color_select:n{red}\tiny\int_use:N\g__tag_para_end_int}
254       \tag_mc_end:
255     }
256   }
257 }
```

```

256     \tag_struct_end:
257   }
258 }
259 \AddToHook{enddocument/info}
260 {
261   \int_compare:nNnF {\g__tag_para_begin_int}={\g__tag_para_end_int}
262   {
263     \msg_error:nnxx
264     {tag}
265     {para-hook-count-wrong}
266     {\int_use:N\g__tag_para_begin_int}
267     {\int_use:N\g__tag_para_end_int}
268   }
269 }

```

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

```

270 \AddToHook{begindocument/before}
271 {
272   \bool_if:NF \g__tag_mode_lua_bool
273   {
274     \cs_if_exist:NT \@kernel@before@footins
275     {
276       \tl_put_right:Nn \@kernel@before@footins
277       { \__tag_add_missing_mcs_to_stream:Nn \footins {footnote} }
278       \tl_put_right:Nn \@kernel@before@cclv
279       {
280         \__tag_check_typeout_v:n {====>-In~\token_to_str:N \@makecol\c_space_tl\the\c@p
281         \__tag_add_missing_mcs_to_stream:Nn \@cclv {main}
282       }
283       \tl_put_right:Nn \@mult@ptagging@hook
284       {
285         \__tag_check_typeout_v:n {====>-In~\string\page@sofar}
286         \process@cols\mult@firstbox
287         {
288           \__tag_add_missing_mcs_to_stream:Nn \count@ {multicol}
289         }
290         \__tag_add_missing_mcs_to_stream:Nn \mult@rightbox {multicol}
291       }
292     }
293   }
294 }
295 \end{package}

```

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

```

296 \base\newcommand\tagpdfparaOn {}
297 \base\newcommand\tagpdfparaOff{}
298 \package
299 \renewcommand\tagpdfparaOn {\bool_set_true:N \l__tag_para_bool}
300 \renewcommand\tagpdfparaOff{\bool_set_false:N \l__tag_para_bool}

```

(End definition for **\tagpdfparaOn** and **\tagpdfparaOff**. These functions are documented on page 33.)

\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}%

301 \NewDocumentCommand\tagpdfsuppressmarks{m}
302   {{\use:c{__tag_mc_disable_marks:} #1}}
```

(End definition for `\tagpdfsuppressmarks`. This function is documented on page 33.)

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 be there at the end. TODO check if Pagination should be changeable.

```
303 \cs_new_protected:Npn\__tag_hook_kernel_before_head:{}
304 \cs_new_protected:Npn\__tag_hook_kernel_after_head:{}
305 \cs_new_protected:Npn\__tag_hook_kernel_before_foot:{}
306 \cs_new_protected:Npn\__tag_hook_kernel_after_foot:{}
307
308 \AddToHook{begindocument}
309 {
310   \cs_if_exist:NT \@kernel@before@head
311   {
312     \tl_put_right:Nn \@kernel@before@head {\__tag_hook_kernel_before_head:}
313     \tl_put_left:Nn  \@kernel@after@head  {\__tag_hook_kernel_after_head:}
314     \tl_put_right:Nn \@kernel@before@foot {\__tag_hook_kernel_before_foot:}
315     \tl_put_left:Nn  \@kernel@after@foot  {\__tag_hook_kernel_after_foot:}
316   }
317 }
318
319 \bool_new:N \g__tag_saved_in_mc_bool
320 \cs_new_protected:Npn \__tag_exclude_headfoot_begin:
321 {
322   \bool_set_false:N \l__tag_para_bool
323   \bool_if:NTF \g__tag_mode_lua_bool
324   {
325     \tag_mc_end_push:
326   }
327   {
328     \bool_gset_eq:NN \g__tag_saved_in_mc_bool \g__tag_in_mc_bool
329     \bool_gset_false:N \g__tag_in_mc_bool
330   }
331   \tag_mc_begin:n {artifact}
332 }
333 \cs_new_protected:Npn \__tag_exclude_headfoot_end:
334 {
335   \tag_mc_end:
```

```

336     \bool_if:NTF \g__tag_mode_lua_bool
337     {
338         \tag_mc_begin_pop:n{ }
339     }
340     {
341         \bool_gset_eq:NN \g__tag_in_mc_bool\g__tag_saved_in_mc_bool
342     }
343 }

```

This version allows to use an Artifact structure

```

344 \__tag_attr_new_entry:nn {__tag/attr/pagination}{/0/Artifact/Type/Pagination}
345 \cs_new_protected:Npn \__tag_exclude_struct_headfoot_begin:n #1
346 {
347     \bool_set_false:N \l__tag_para_bool
348     \bool_if:NTF \g__tag_mode_lua_bool
349     {
350         \tag_mc_end_push:
351     }
352     {
353         \bool_gset_eq:NN \g__tag_saved_in_mc_bool \g__tag_in_mc_bool
354         \bool_gset_false:N \g__tag_in_mc_bool
355     }
356     \tag_struct_begin:n{tag=Artifact,attribute-class=__tag/attr/#1}
357     \tag_mc_begin:n {artifact=#1}
358 }
359
360 \cs_new_protected:Npn \__tag_exclude_struct_headfoot_end:
361 {
362     \tag_mc_end:
363     \tag_struct_end:
364     \bool_if:NTF \g__tag_mode_lua_bool
365     {
366         \tag_mc_begin_pop:n{ }
367     }
368     {
369         \bool_gset_eq:NN \g__tag_in_mc_bool\g__tag_saved_in_mc_bool
370     }
371 }

```

And now the keys

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

```

372 \keys_define:nn { __tag / setup }
373 {
374     exclude-header-footer .choice:,
375     exclude-header-footer / true .code:n =
376     {
377         \cs_set_eq:NN \__tag_hook_kernel_before_head: \__tag_exclude_headfoot_begin:
378         \cs_set_eq:NN \__tag_hook_kernel_before_foot: \__tag_exclude_headfoot_begin:
379         \cs_set_eq:NN \__tag_hook_kernel_after_head: \__tag_exclude_headfoot_end:
380         \cs_set_eq:NN \__tag_hook_kernel_after_foot: \__tag_exclude_headfoot_end:
381     },
382     exclude-header-footer / pagination .code:n =
383     {
384         \cs_set:Nn \__tag_hook_kernel_before_head: { \__tag_exclude_struct_headfoot_begin:n {p

```

```

385     \cs_set:Nn \__tag_hook_kernel_before_foot: { \__tag_exclude_struct_headfoot_begin:n {p
386     \cs_set_eq:NN \__tag_hook_kernel_after_head: \__tag_exclude_struct_headfoot_end:
387     \cs_set_eq:NN \__tag_hook_kernel_after_foot: \__tag_exclude_struct_headfoot_end:
388   },
389   exclude-header-footer / false .code:n =
390   {
391     \cs_set_eq:NN \__tag_hook_kernel_before_head: \prg_do_nothing:
392     \cs_set_eq:NN \__tag_hook_kernel_before_foot: \prg_do_nothing:
393     \cs_set_eq:NN \__tag_hook_kernel_after_head: \prg_do_nothing:
394     \cs_set_eq:NN \__tag_hook_kernel_after_foot: \prg_do_nothing:
395   },
396   exclude-header-footer .default:n = true,
397   exclude-header-footer .initial:n = true
398 }

```

(End definition for `exclude-header-footer` (setup-key). This function is documented on page 33.)

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.

```

399 \hook_gput_code:nnn
400 {pdfannot/link/URI/before}
401 {tagpdf}
402 {
403   \tag_mc_end_push:
404   \tag_struct_begin:n { tag=Link }
405   \tag_mc_begin:n { tag=Link }
406   \pdfannot_dict_put:nnx
407     { link/URI }
408     { StructParent }
409     { \tag_struct_parent_int: }
410 }
411
412 \hook_gput_code:nnn
413 {pdfannot/link/URI/after}
414 {tagpdf}
415 {
416   \tag_struct_insert_annot:xx {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
417   \tag_mc_end:
418   \tag_struct_end:
419   \tag_mc_begin_pop:n{ }
420 }
421
422 \hook_gput_code:nnn
423 {pdfannot/link/GoTo/before}
424 {tagpdf}
425 {
426   \tag_mc_end_push:
427   \tag_struct_begin:n{tag=Link}
428   \tag_mc_begin:n{tag=Link}
429   \pdfannot_dict_put:nnx
430     { link/GoTo }

```

```

431         { StructParent }
432         { \tag_struct_parent_int: }
433     }
434
435     \hook_gput_code:nnn
436     {pdfannot/link/GoTo/after}
437     {tagpdf}
438     {
439         \tag_struct_insert_annot:xx {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
440         \tag_mc_end:
441         \tag_struct_end:
442         \tag_mc_begin_pop:n{ }
443     }
444 }
445
446 % "alternative descriptions " for PAX3. How to get better text here??
447 \pdfannot_dict_put:nnn
448 { link/URI }
449 { Contents }
450 { (url) }
451
452 \pdfannot_dict_put:nnn
453 { link/GoTo }
454 { Contents }
455 { (ref) }
456
</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} {2022-08-24} {0.97}
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:n { __tag/struct/0 }
(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:n { __tag/tree/parenttree }

```

(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:nxx { __tag/tree/parenttree }{dict}
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:n { __tag/tree/rolemap }

```

(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:nxx { __tag/tree/rolemap }{dict}
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:n { __tag/tree/classmap }
177         \pdf_object_write:nxx
178         { __tag/tree/classmap }
179         {dict}
180         { \l__tag_tmpa_tl }
181         \__tag_prop_gput:cnx
182         { g__tag_struct_0_prop }
183         { ClassMap }
184         { \pdf_object_ref:n { __tag/tree/classmap } }
185     }
186 }

```

(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

```

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

```

(End definition for __tag/tree/namespaces.)

__tag_tree_write_namespaces:

```

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

```

```

207     {
208       \prop_map_tokens:Nn \g__tag_role_NS_prop{\use_i:nn}
209     }
210   }

```

(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:

```

211 \hook_new:n {tagpdf/finish/before}
212 \cs_new_protected:Npn \_tag_finish_structure:
213   {
214     \bool_if:NT\g__tag_active_tree_bool
215     {
216       \hook_use:n {tagpdf/finish/before}
217       \_tag_tree_write_parenttree:
218       \_tag_tree_write_rolemap:
219       \_tag_tree_write_classmap:
220       \_tag_tree_write_namespaces:
221       \_tag_tree_write_structelements: %this is rather slow!!
222       \_tag_tree_write_structtreeroot:
223     }
224   }

```

(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.

```

225 \hook_gput_code:nnn{begindocument}{tagpdf}
226   {
227     \bool_if:NT\g__tag_active_tree_bool
228     {
229       \hook_gput_code:nnn{shipout/before} { tagpdf/structparents }
230       {
231         \pdfmanagement_add:nnx
232         { Page }
233         { StructParents }
234         { \int_eval:n { \g_shipout_readonly_int } }
235       }
236     }
237   }
238 </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 alt text for now.

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

2 Public keys

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

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

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

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

`alt_␣(mc-key)` This key inserts an `/Alt` value in the property dictionary of the BDC operator. The value is handled as verbatim string, commands are not expanded. The value will be expanded first once.

`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} {2022-08-24} {0.97}
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 `}`
29 `\shared`
(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?

30 `\base\cs_new_protected:Npn \tag_mc_use:n #1 { __tag_whatsits: }`
31 `\shared`
32 `\cs_set_protected:Npn \tag_mc_use:n #1 %#1: label name`
33 `{`
34 `__tag_check_if_active_struct:T`
35 `{`
36 `\tl_set:Nx \l__tag_tmpa_tl { __tag_ref_value:nnn{tagpdf-#1}{tagmcabs}{ } }`
37 `\tl_if_empty:NTF\l__tag_tmpa_tl`
38 `{`
39 `\msg_warning:nnn {tag} {mc-label-unknown} {#1}`
40 `}`
41 `}`

```

42         \cs_if_free:cTF { g__tag_mc_label_\tl_to_str:n{#1}_used_tl }
43         {
44             \__tag_mc_handle_stash:x { \l__tag_tmpa_tl }
45             \__tag_mc_set_label_used:n {#1}
46         }
47         {
48             \msg_warning:nnn {tag}{mc-used-twice}{#1}
49         }
50     }
51 }
52 }
53 </shared>

```

(End definition for \tag_mc_use:n. This function is documented on page 53.)

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

\tag_mc_artifact_group_end:

```

54 <base>\cs_new_protected:Npn \tag_mc_artifact_group_begin:n #1 {}
55 <base>\cs_new_protected:Npn \tag_mc_artifact_group_end: {}
56 <*shared>
57 \cs_set_protected:Npn \tag_mc_artifact_group_begin:n #1
58 {
59     \tag_mc_end_push:
60     \tag_mc_begin:n {artifact=#1}
61     \tag_stop_group_begin:
62 }
63
64 \cs_set_protected:Npn \tag_mc_artifact_group_end:
65 {
66     \tag_stop_group_end:
67     \tag_mc_end:
68     \tag_mc_begin_pop:n {}
69 }
70 </shared>

```

(End definition for \tag_mc_artifact_group_begin:n and \tag_mc_artifact_group_end:. These functions are documented on page 53.)

\tag_mc_end_push:
\tag_mc_begin_pop:n

```

71 <base>\cs_new_protected:Npn \tag_mc_end_push: {}
72 <base>\cs_new_protected:Npn \tag_mc_begin_pop:n #1 {}
73 <*shared>
74 \cs_set_protected:Npn \tag_mc_end_push:
75 {
76     \__tag_check_if_active_mc:T
77     {
78         \__tag_mc_if_in:TF
79         {
80             \seq_gpush:Nx \g__tag_mc_stack_seq { \tag_get:n {mc_tag} }
81             \__tag_check_mc_pushed_popped:nn
82             { pushed }
83             { \tag_get:n {mc_tag} }
84             \tag_mc_end:
85         }
86     }

```

```

87         \seq_gpush:Nn \g__tag_mc_stack_seq {-1}
88         \__tag_check_mc_pushed_popped:nn { pushed }{-1}
89     }
90 }
91 }
92
93 \cs_set_protected:Npn \tag_mc_begin_pop:n #1
94 {
95     \__tag_check_if_active_mc:T
96     {
97         \seq_gpop:NNTF \g__tag_mc_stack_seq \l__tag_tmpa_tl
98         {
99             \tl_if_eq:NnTF \l__tag_tmpa_tl {-1}
100             {
101                 \__tag_check_mc_pushed_popped:nn {popped}{-1}
102             }
103             {
104                 \__tag_check_mc_pushed_popped:nn {popped}{\l__tag_tmpa_tl}
105                 \tag_mc_begin:n {tag=\l__tag_tmpa_tl,#1}
106             }
107         }
108         {
109             \__tag_check_mc_pushed_popped:nn {popped}{empty~stack,~nothing}
110         }
111     }
112 }

```

(End definition for \tag_mc_end_push: and \tag_mc_begin_pop:n. These functions are documented on page 53.)

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.

```

113 \keys_define:nn { __tag / mc }
114 {
115     stash .bool_set:N = \l__tag_mc_key_stash_bool,
116     __artifact-bool .bool_set:N = \l__tag_mc_artifact_bool,
117     __artifact-type .choice:,
118     __artifact-type / pagination .code:n =
119     {
120         \tl_set:Nn \l__tag_mc_artifact_type_tl { Pagination }
121     },
122     __artifact-type / pagination/header .code:n =
123     {
124         \tl_set:Nn \l__tag_mc_artifact_type_tl { Pagination/Subtype/Header }
125     },
126     __artifact-type / pagination/footer .code:n =
127     {
128         \tl_set:Nn \l__tag_mc_artifact_type_tl { Pagination/Subtype/Footer }

```

```

129     },
130     __artifact-type / layout      .code:n    =
131     {
132         \tl_set:Nn \l__tag_mc_artifact_type_tl { Layout }
133     },
134     __artifact-type / page        .code:n    =
135     {
136         \tl_set:Nn \l__tag_mc_artifact_type_tl { Page }
137     },
138     __artifact-type / background .code:n    =
139     {
140         \tl_set:Nn \l__tag_mc_artifact_type_tl { Background }
141     },
142     __artifact-type / notype      .code:n    =
143     {
144         \tl_set:Nn \l__tag_mc_artifact_type_tl {}
145     },
146     __artifact-type /            .code:n    =
147     {
148         \tl_set:Nn \l__tag_mc_artifact_type_tl {}
149     },
150 }

```

(End definition for stash (mc-key), __artifact-bool, and __artifact-type. This function is documented on page 54.)

```

151 </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} {2022-08-24} {0.97}
4 {part of tagpdf - code related to marking chunks - generic mode}
5 </generic>
6 <*debug>
7 \ProvidesExplPackage {tagpdf-debug-generic} {2022-08-24} {0.97}
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         \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 note text, or `multicol` for boxes produced for columns in that environment. Other streams may follow over time.

```

164 \cs_new_protected: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:nNt {\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 {

```

First set up a temp box for trial splitting.

```

189   \vbadness\maxdimen
190   \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 53.)

`_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 <base>\cs_new_protected:Npn \tag_mc_begin:n #1 { \__tag_whatsits: }
319 <base>\cs_new_protected:Nn \tag_mc_end:{ \__tag_whatsits: }
320 <*generic | debug>
321 <*generic>
322 \cs_set_protected:Npn \tag_mc_begin:n #1 %#1 keyval
323   {
324     \__tag_check_if_active_mc:T
325     {
326       </generic>
327       <*debug>
328     }
329   }
330 \cs_set_protected:Npn \tag_mc_begin:n #1 %#1 keyval
331   {
332     \__tag_check_if_active_mc:TF
333     {
334       \__tag_debug_mc_begin_insert:n { #1 }

```

```

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

```

```

387 <*debug>
388     {
389         \__tag_debug_mc_end_ignore:
390     }
391 </debug>
392 }
393 </generic | debug>

```

(End definition for \tag_mc_begin:n and \tag_mc_end:. These functions are documented on page 53.)

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

```

```

432         { __tag / mc }
433         { __artifact-bool, __artifact-type=#1 }
434     },
435     artifact .default:n    = {notype}
436 }
437 </generic>

```

(End definition for tag (mc-key) and others. These functions are documented on page 54.)

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 ... \tagmcend` 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} {2022-08-24} {0.97}
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 53.)

_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_set_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 53.)

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

```

185 \cs_set_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 53.)

__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 }
alt_(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   },
alt .code:n      216   = % Alt property
                  217   {
                  218     \str_set_convert:Noon
                  219     \l__tag_tmpa_str
                  220     { #1 }
                  221     { default }
                  222     { utf16/hex }
                  223     \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
                  224     \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
                  225     \lua_now:e
                  226     {
                  227       ltx.__tag.func.store_mc_data
                  228       (
                  229         \__tag_get_mc_abs_cnt:,"alt","/Alt~<\str_use:N \l__tag_tmpa_str>"
                  230       )
                  231     }
                  232   },
alttext .meta:n  233   = {alt=#1},
                  234

```

```

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

```

289 `</luamode>`

(End definition for tag `(mc-key)` and others. These functions are documented on page 54.)

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.

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

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

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

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

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

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

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

2 Public keys

2.1 Keys for the structure commands

<u>tag_□(struct-key)</u>	This is required. The value of the key is normally one of the standard types listed in the main tagpdf documentation. It is possible to setup new tags/types. The value can also be of the form type/NS , where NS is the shorthand of a declared name space. Currently the names spaces pdf , pdf2 , mathml and user are defined. This allows to use a different name space than the one connected by default to the tag. But normally this should not be needed.
<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 refer to the structure. It is e.g. used by <code>\tag_struct_use:n</code> (where a real label is actually not needed as you can only use structures already defined), and by the ref key (which can refer to future structures). Internally the label name will start with tagpdfstruct- and it stores the two attributes tagstruct (the structure number) and tagstructobj (the object reference).
<u>parent_□(struct-key)</u>	By default a structure is added as kid to the currently active structure. With the parent key one can choose another parent. The value is a structure number which must refer to an already existing, previously created structure. Such a structure number can for example be have been stored with <code>\tag_get:n</code> , but one can also use a label on the parent structure and then use <code>\ref_value:nn{tagpdfstruct-label}{tagstruct}</code> to retrieve it.
<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>alt_□(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 .

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

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

AF_□(struct-key) `AF = <object name>`

AFinline_□(struct-key) `AF-inline = <text content>`

AFinline-o_□(struct-key) These keys allows to reference an associated file in the structure element. The value `<object name>` should be the name of an object pointing to the `/Filespec` dictionary as expected by `\pdf_object_ref:n` from a current `l3kernel`.

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} {2022-08-24} {0.97}
4 {part of tagpdf - code related to storing structure}
5 </header>
```

3 Variables

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

```
6 <base>\newcounter { g__tag_struct_abs_int }
7 <base>\int_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.

```
8 <*package>
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. It is already defined in `tagpdf-base`. The local temporary variable will hold the parent when we fetch it from the stack.

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

```
20 \seq_const_from_clist:Nn \c__tag_struct_StructTreeRoot_entries_seq
21   {%p. 857/858
22     Type,                % always /StructTreeRoot
23     K,                   % kid, dictionary or array of dictionaries
24     IDTree,              % currently unused
25     ParentTree,          % required,obj ref to the parent tree
26     ParentTreeNextKey,   % optional
27     RoleMap,
28     ClassMap,
29     Namespaces,
30     AF                   %pdf 2.0
31   }
32
```

```

33 \seq_const_from_clist:Nn \c__tag_struct_StructElem_entries_seq
34   {%p 858 f
35     Type,           %always /StructElem
36     S,             %tag/type
37     P,             %parent
38     ID,            %optional
39     Ref,           %optional, pdf 2.0 Use?
40     Pg,            %obj num of starting page, optional
41     K,             %kids
42     A,             %attributes, probably unused
43     C,             %class ""
44     %R,            %attribute revision number, irrelevant for us as we
45                   % don't update/change existing PDF and (probably)
46                   % deprecated in PDF 2.0
47     T,             %title, value in () or <>
48     Lang,          %language
49     Alt,           % value in () or <>
50     E,             % abbreviation
51     ActualText,
52     AF,            %pdf 2.0, array of dict, associated files
53     NS,            %pdf 2.0, dict, namespace
54     PhoneticAlphabet, %pdf 2.0
55     Phoneme        %pdf 2.0
56   }

```

(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.

```

57 \tl_new:N \g__tag_struct_tag_tl
58 \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.

```

59 \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

```

60 \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
61 \cs_new:Npn \__tag_struct_output_prop_aux:nn #1 #2 %#1 num, #2 key
62 {
63   \prop_if_in:cnT
64     { g__tag_struct_#1_prop }
65     { #2 }
66     {
67       \c_space_tl/#2~ \prop_item:cn{ g__tag_struct_#1_prop } { #2 }
68     }
69   }
70
71 \cs_new_protected:Npn \__tag_new_output_prop_handler:n #1
72 {
73   \cs_new:cn { __tag_struct_output_prop_#1:n }
74   {
75     \__tag_struct_output_prop_aux:nn {#1}{##1}
76   }
77 }

(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.

```

78 \tl_gset:Nn \g__tag_struct_stack_current_tl {0}

g__tag_struct_0_prop
g__tag_struct_kids_0_seq
79 \__tag_prop_new:c { g__tag_struct_0_prop }
80 \__tag_new_output_prop_handler:n {0}
81 \__tag_seq_new:c { g__tag_struct_kids_0_seq }
82
83 \__tag_prop_gput:cnn
84 { g__tag_struct_0_prop }
85 { Type }
86 { /StructTreeRoot }
87
88
89

```

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.

```

90 \__tag_prop_gput:cnx
91 { g__tag_struct_0_prop }
92 { Namespaces }
93 { \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:xx

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 to 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!

```

94 \cs_new:Npn \__tag_struct_mcid_dict:n #1 %#1 MCID absnum
95 {
96   <<
97   /Type \c_space_tl /MCR \c_space_tl
98   /Pg
99   \c_space_tl
100   \pdf_pageobject_ref:n { \__tag_ref_value:enn{mcid-#1}{tagabspage}{1} }
101   /MCID \c_space_tl \__tag_ref_value:enn{mcid-#1}{tagmcid}{1}
102   >>
103 }
104 \cs_new_protected:Npn \__tag_struct_kid_mc_gput_right:nn #1 #2 %#1 structure num, #2 MCID absnum
105 {
106   \__tag_seq_gput_right:cx
107   { g__tag_struct_kids_#1_seq }
108   {
109     \__tag_struct_mcid_dict:n {#2}
110   }
111   \__tag_seq_gput_right:cn
112   { g__tag_struct_kids_#1_seq }
113   {
114     \prop_item:Nn \g__tag_struct_cont_mc_prop {#2}
115   }
116 }
117 \cs_generate_variant:Nn \__tag_struct_kid_mc_gput_right:nn {nx}
118

```

(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.

```

119 \cs_new_protected:Npn \__tag_struct_kid_struct_gput_right:nn #1 #2 %#1 num of parent struct, #2
120 {
121   \__tag_seq_gput_right:cx
122   { g__tag_struct_kids_#1_seq }
123   {
124     \pdf_object_ref:n { __tag/struct/#2 }
125   }
126 }
127
128 \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:nnn
_tag_struct_kid_OBJR_gput_right:xxx

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

```

129 \cs_new_protected:Npn\_tag_struct_kid_OBJR_gput_right:nnn #1 #2 #3 % #1 num of parent struct,
130                                     % #2 obj reference
131                                     % #3 page object reference
132 {
133   \pdf_object_unnamed_write:nn
134     { dict }
135     {
136       /Type/OBJR/Obj~#2/Pg~#3
137     }
138   \_tag_seq_gput_right:cx
139     { g__tag_struct_kids_#1_seq }
140     {
141       \pdf_object_ref_last:
142     }
143 }
144
145 \cs_generate_variant:Nn\_tag_struct_kid_OBJR_gput_right:nnn { xxx }
146

```

(End definition for _tag_struct_kid_OBJR_gput_right:nnn.)

_tag_struct_exchange_kid_command:N
_tag_struct_exchange_kid_command:c

In luamode it can happen that a single kid in a structure is split at a page break into two or more mcid. In this case the lua code has to convert put the dictionary of the kid into an array. See issue 13 at tagpdf repo. We exchange the dummy command for the kids to mark this case.

```

147 \cs_new_protected:Npn\_tag_struct_exchange_kid_command:N #1 % #1 = seq var
148 {
149   \seq_gpop_left:NN #1 \l__tag_tmpa_tl
150   \regex_replace_once:nnN
151     { \c{\_tag_mc_insert_mcid_kids:n} }
152     { \c{\_tag_mc_insert_mcid_single_kids:n} }
153     \l__tag_tmpa_tl
154   \seq_gput_left:NV #1 \l__tag_tmpa_tl
155 }
156
157 \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.

```

158 \cs_new_protected:Npn\_tag_struct_fill_kid_key:n #1 % #1 is the struct num
159 {
160   \bool_if:NF\g__tag_mode_lua_bool
161   {
162     \seq_clear:N \l__tag_tmpa_seq
163     \seq_map_inline:cn { g__tag_struct_kids_#1_seq }
164       { \seq_put_right:Nx \l__tag_tmpa_seq { ##1 } }

```

```

165     \%seq_show:c { g__tag_struct_kids_#1_seq }
166     \%seq_show:N \l__tag_tmpa_seq
167     \seq_remove_all:Nn \l__tag_tmpa_seq {}
168     \%seq_show:N \l__tag_tmpa_seq
169     \seq_gset_eq:cN { g__tag_struct_kids_#1_seq } \l__tag_tmpa_seq
170 }
171
172 \int_case:nnF
173 {
174     \seq_count:c
175     {
176         g__tag_struct_kids_#1_seq
177     }
178 }
179 {
180     { 0 }
181     { } %no kids, do nothing
182     { 1 } % 1 kid, insert
183     {
184         % in this case we need a special command in
185         % luamode to get the array right. See issue #13
186         \bool_if:NT\g__tag_mode_lua_bool
187         {
188             \__tag_struct_exchange_kid_command:c
189             {g__tag_struct_kids_#1_seq}
190         }
191         \__tag_prop_gput:cnx { g__tag_struct_#1_prop } {K}
192         {
193             \seq_item:cn
194             {
195                 g__tag_struct_kids_#1_seq
196             }
197             {1}
198         }
199     } %
200 }
201 { %many kids, use an array
202     \__tag_prop_gput:cnx { g__tag_struct_#1_prop } {K}
203     {
204         [
205             \seq_use:cn
206             {
207                 g__tag_struct_kids_#1_seq
208             }
209             {
210                 \c_space_tl
211             }
212         ]
213     }
214 }
215 }
216

```

(End definition for __tag_struct_fill_kid_key:n.)

4.3 Output of the object

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

```

217 \cs_new_protected:Npn \__tag_struct_get_dict_content:nN #1 #2 %#1: structure num
218 {
219   \tl_clear:N #2
220   \seq_map_inline:cn
221   {
222     c__tag_struct_
223     \int_compare:nNnTF{#1}={0}{StructTreeRoot}{StructElem}
224     _entries_seq
225   }
226   {
227     \tl_put_right:Nx
228     #2
229     {
230       \prop_if_in:cnT
231       { g__tag_struct_#1_prop }
232       { ##1 }
233       {
234         \c_space_tl/##1~
235         \cs_if_exist_use:cTF {__tag_struct_format_##1:e}
236         {
237           { \prop_item:cn{ g__tag_struct_#1_prop } { ##1 } }
238         }
239         {
240           \prop_item:cn{ g__tag_struct_#1_prop } { ##1 }
241         }
242       }
243     }
244   }
245 }

```

(End definition for `__tag_struct_get_dict_content:nN`.)

`__tag_struct_format_Ref:n` Ref is an array, we store only the content to be able to extend it so the formatting command adds the brackets:

```

246 \cs_new:Nn\__tag_struct_format_Ref:n{[#1]}
247 \cs_generate_variant:Nn\__tag_struct_format_Ref:n{e}

```

(End definition for `__tag_struct_format_Ref:n`.)

`__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.

```

248 \cs_new_protected:Npn \__tag_struct_write_obj:n #1 % #1 is the struct num
249 {
250   \pdf_object_if_exist:nTF { __tag/struct/#1 }
251   {
252     \__tag_struct_fill_kid_key:n { #1 }
253     \__tag_struct_get_dict_content:nN { #1 } \l__tag_tmpa_tl
254     \exp_args:Nx

```

```

255         \pdf_object_write:nnx
256         { __tag/struct/#1 }
257         {dict}
258         {
259             \l__tag_tmpa_tl
260         }
261     }
262     {
263         \msg_error:nnn { tag } { struct-no-objnum } { #1}
264     }
265 }

```

(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:

```

```

266 \cs_new_protected:Npn \__tag_struct_insert_annot:nn #1 #2 %#1 object reference to the annotat
267                                     %#2 structparent number
268 {
269     \bool_if:NT \g__tag_active_struct_bool
270     {
271         %get the number of the parent structure:
272         \seq_get:NMF
273         \g__tag_struct_stack_seq
274         \l__tag_struct_stack_parent_tmpa_tl
275         {
276             \msg_error:nn { tag } { struct-faulty-nesting }
277         }
278         %put the obj number of the annot in the kid entry, this also creates
279         %the OBJR object
280         \ref_label:nn {__tag_objr_page_#2 }{ tagabspage }
281         \__tag_struct_kid_OBJR_gput_right:xxx
282         {
283             \l__tag_struct_stack_parent_tmpa_tl

```

```

284     }
285     {
286         #1 %
287     }
288     {
289         \pdf_pageobject_ref:n { \__tag_ref_value:nnn {__tag_objr_page_#2 }{ tagabspace }{
290     }
291     % add the parent obj number to the parent tree:
292     \exp_args:Nnx
293     \__tag_parenttree_add_objr:nn
294     {
295         #2
296     }
297     {
298         \pdf_object_ref:e { __tag/struct/\l__tag_struct_stack_parent_tmpa_tl }
299     }
300     % increase the int:
301     \stepcounter{ g__tag_parenttree_obj_int }
302 }
303 }

```

(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

```

304 \cs_new:Npn \__tag_get_data_struct_tag:
305 {
306     \exp_args:Ne
307     \tl_tail:n
308     {
309         \prop_item:cn {g__tag_struct\_g__tag_struct_stack_current_tl _prop}{S}
310     }
311 }
312 \</package>

```

(End definition for __tag_get_data_struct_tag:.)

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

```

313 \< *base>
314 \cs_new:Npn \__tag_get_data_struct_num:
315 {
316     \g__tag_struct_stack_current_tl
317 }
318 \</base>

```

(End definition for __tag_get_data_struct_num:.)

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)
parent_(struct-key)
tag_(struct-key)
title_(struct-key)
title-o_(struct-key)
alt_(struct-key)
actualtext_(struct-key)
lang_(struct-key)
ref_(struct-key)
E_(struct-key)

319 (*package)
320 \keys_define:nn { __tag / struct }
321 {
322   label .tl_set:N      = \l__tag_struct_key_label_tl,
323   stash .bool_set:N    = \l__tag_struct_elem_stash_bool,
324   parent .code:n      =
325   {
326     \bool_lazy_and:nnTF
327     {
328       \prop_if_exist_p:c { g__tag_struct_\int_eval:n {#1}_prop }
329     }
330     {
331       \int_compare_p:nNn {#1}<{\c@g__tag_struct_abs_int}
332     }
333     { \tl_set:Nx \l__tag_struct_stack_parent_tmpa_tl { \int_eval:n {#1} } }
334     {
335       \msg_warning:nnxx { tag } { struct-unknown }
336       { \int_eval:n {#1} }
337       { parent-key-ignored }
338     }
339   },
340   parent .default:n    = {-1},
341   tag .code:n          = % S property
342   {
343     \seq_set_split:Nne \l__tag_tmpa_seq { / } {#1/\prop_item:No\g__tag_role_tags_NS_prop{
344     \tl_gset:Nx \g__tag_struct_tag_tl { \seq_item:Nn\l__tag_tmpa_seq {1} }
345     \tl_gset:Nx \g__tag_struct_tag_NS_tl { \seq_item:Nn\l__tag_tmpa_seq {2} }
346     \__tag_check_structure_tag:N \g__tag_struct_tag_tl
347     \__tag_prop_gput:cnx
348     { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
349     { S }
350     { \pdf_name_from_unicode_e:n{ \g__tag_struct_tag_tl } } %
351     \prop_get:NVNT \g__tag_role_NS_prop\g__tag_struct_tag_NS_tl\l__tag_tmpa_tl
352     {
353       \__tag_prop_gput:cnx
354       { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
355       { NS }
356       { \l__tag_tmpa_tl } %
357     }
358   },
359   title .code:n        = % T property
360   {
361     \str_set_convert:Nnnn
362     \l__tag_tmpa_str
363     { #1 }
364     { default }
365     { utf16/hex }
366     \__tag_prop_gput:cnx
367     { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
368     { T }
369     { <\l__tag_tmpa_str> }
370   },
371   title-o .code:n      = % T property

```

```

372     {
373         \str_set_convert:Nonn
374         \l__tag_tmpa_str
375         { #1 }
376         { default }
377         { utf16/hex }
378         \__tag_prop_gput:cnx
379         { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
380         { T }
381         { <\l__tag_tmpa_str> }
382     },
383     alt .code:n      = % Alt property
384     {
385         \str_set_convert:Noon
386         \l__tag_tmpa_str
387         { #1 }
388         { default }
389         { utf16/hex }
390         \__tag_prop_gput:cnx
391         { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
392         { Alt }
393         { <\l__tag_tmpa_str> }
394     },
395     alttext .meta:n = {alt=#1},
396     actualtext .code:n = % ActualText property
397     {
398         \str_set_convert:Noon
399         \l__tag_tmpa_str
400         { #1 }
401         { default }
402         { utf16/hex }
403         \__tag_prop_gput:cnx
404         { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
405         { ActualText }
406         { <\l__tag_tmpa_str>}
407     },
408     lang .code:n      = % Lang property
409     {
410         \__tag_prop_gput:cnx
411         { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
412         { Lang }
413         { (#1) }
414     },

```

Ref is an array, the brackets are added through the formatting command.

```

415     ref .code:n      = % ref property
416     {
417         \tl_clear:N\l__tag_tmpa_tl
418         \clist_map_inline:on {#1}
419         {
420             \tl_put_right:Nx \l__tag_tmpa_tl
421             {~\ref_value:nn{tagpdfstruct-##1}{tagstructobj} }
422         }
423         \__tag_struct_gput_data_ref:ee { \int_eval:n {\c@g__tag_struct_abs_int} } {\l__tag_tmpa_tl}
424     },

```

```

425     E .code:n          = % E property
426     {
427         \str_set_convert:Nnon
428         \l__tag_tmpa_str
429         { #1 }
430         { default }
431         { utf16/hex }
432         \__tag_prop_gput:cnx
433         { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
434         { E }
435         { <\l__tag_tmpa_str> }
436     },
437 }

```

(End definition for label (struct-key) and others. These functions are documented on page 82.)

AF_□(struct-key)
AFinline_□(struct-key)
AFinline-o_□(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).

```

438 \cs_new_protected:Npn \__tag_struct_add_AF:nn #1 #2 % #1 struct num #2 object name
439 {
440     \tl_if_exist:cTF
441     {
442         g__tag_struct_#1_AF_tl
443     }
444     {
445         \tl_gput_right:cx
446         { g__tag_struct_#1_AF_tl }
447         { ~ \pdf_object_ref:n {#2} }
448     }
449     {
450         \tl_new:c
451         { g__tag_struct_#1_AF_tl }
452         \tl_gset:cx
453         { g__tag_struct_#1_AF_tl }
454         { \pdf_object_ref:n {#2} }
455     }
456 }
457 \cs_generate_variant:Nn \__tag_struct_add_AF:nn {en,ee}
458 \keys_define:nn { __tag / struct }
459 {
460     AF .code:n          = % AF property
461     {
462         \pdf_object_if_exist:nTF {#1}
463         {
464             \__tag_struct_add_AF:en { \int_eval:n {\c@g__tag_struct_abs_int} }{#1}
465             \__tag_prop_gput:cnx
466             { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
467             { AF }
468             {
469                 [
470                     \tl_use:c

```

```

471         { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
472     ]
473 }
474 }
475 {
476
477 }
478 },
479 ,AFinline .code:n =
480 {
481     \group_begin:
482     \pdf_object_if_exist:eF {__tag/fileobj\int_use:N\c@g__tag_struct_abs_int}
483     {
484         \pdffile_embed_stream:nxx
485         {#1}
486         {tag-AFfile\int_use:N\c@g__tag_struct_abs_int.txt}
487         {__tag/fileobj\int_use:N\c@g__tag_struct_abs_int}
488         \__tag_struct_add_AF:ee
489         { \int_eval:n {\c@g__tag_struct_abs_int} }
490         { __tag/fileobj\int_use:N\c@g__tag_struct_abs_int }
491         \__tag_prop_gput:cnx
492         { g__tag_struct_\int_use:N\c@g__tag_struct_abs_int _prop }
493         { AF }
494         {
495             [
496                 \tl_use:c
497                 { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
498             ]
499         }
500     }
501     \group_end:
502 }
503 ,AFinline-o .code:n =
504 {
505     \group_begin:
506     \pdf_object_if_exist:eF {__tag/fileobj\int_use:N\c@g__tag_struct_abs_int}
507     {
508         \pdffile_embed_stream:oxx
509         {#1}
510         {tag-AFfile\int_use:N\c@g__tag_struct_abs_int.txt}
511         {__tag/fileobj\int_use:N\c@g__tag_struct_abs_int}
512         \__tag_struct_add_AF:ee
513         { \int_eval:n {\c@g__tag_struct_abs_int} }
514         { __tag/fileobj\int_use:N\c@g__tag_struct_abs_int }
515         \__tag_prop_gput:cnx
516         { g__tag_struct_\int_use:N\c@g__tag_struct_abs_int _prop }
517         { AF }
518         {
519             [
520                 \tl_use:c
521                 { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
522             ]
523         }
524     }

```

```

525     \group_end:
526   }
527 }

```

(End definition for `AF (struct-key)`, `AFinline (struct-key)`, and `AFinline-o (struct-key)`. These functions are documented on page 83.)

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!

```

528 \keys_define:nn { __tag / setup }
529 {
530   root-AF .code:n =
531   {
532     \pdf_object_if_exist:nTF {#1}
533     {
534       \__tag_struct_add_AF:en { 0 }{#1}
535       \__tag_prop_gput:cnx
536       { g__tag_struct_0_prop }
537       { AF }
538       {
539         [
540           \tl_use:c
541           { g__tag_struct_0_AF_tl }
542         ]
543       }
544     }
545     {
546     }
547   },
548 }
549 }
550 \</package>

```

(End definition for `root-AF (setup-key)`. This function is documented on page 84.)

6 User commands

```

\tag_struct_begin:n
\tag_struct_end:
551 <base>\cs_new_protected:Npn \tag_struct_begin:n #1 {\int_gincr:N \c@g__tag_struct_abs_int}
552 <base>\cs_new_protected:Npn \tag_struct_end: {}
553 <*package|debug>
554 <package>\cs_set_protected:Npn \tag_struct_begin:n #1 % #1 key-val
555 <debug>\cs_set_protected:Npn \tag_struct_begin:n #1 % #1 key-val
556 {
557 <package>\__tag_check_if_active_struct:T
558 <debug>\__tag_check_if_active_struct:TF
559 {
560   \group_begin:
561   \int_gincr:N \c@g__tag_struct_abs_int
562   \__tag_prop_new:c { g__tag_struct\_int_eval:n { \c@g__tag_struct_abs_int }_prop }
563   \__tag_new_output_prop_handler:n {\int_eval:n { \c@g__tag_struct_abs_int }}
564   \__tag_seq_new:c { g__tag_struct_kids\_int_eval:n { \c@g__tag_struct_abs_int }_seq}
565   \exp_args:Ne

```

```

566     \pdf_object_new:n
567     { __tag/struct/\int_eval:n { \c@g__tag_struct_abs_int } }
568   \__tag_prop_gput:cno
569   { g__tag_struct_\int_eval:n { \c@g__tag_struct_abs_int }_prop }
570   { Type }
571   { /StructElem }
572   \tl_set:Nn \l__tag_struct_stack_parent_tmpa_tl {-1}
573   \keys_set:nn { __tag / struct } { #1 }
574   \__tag_check_structure_has_tag:n { \int_eval:n { \c@g__tag_struct_abs_int } }
575   \tl_if_empty:NF
576     \l__tag_struct_key_label_tl
577     {
578       \__tag_ref_label:en{tagpdfstruct-\l__tag_struct_key_label_tl}{struct}
579     }

```

The structure number of the parent is either taken from the stack or has been set with the parent key.

```

580   \int_compare:nNnT { \l__tag_struct_stack_parent_tmpa_tl } = { -1 }
581   {
582     \seq_get:NNF
583     \g__tag_struct_stack_seq
584     \l__tag_struct_stack_parent_tmpa_tl
585     {
586       \msg_error:nn { tag } { struct-faulty-nesting }
587     }
588   }
589   \seq_gpush:NV \g__tag_struct_stack_seq \c@g__tag_struct_abs_int
590   \seq_gpush:NV \g__tag_struct_tag_stack_seq \g__tag_struct_tag_tl
591   \tl_gset:NV \g__tag_struct_stack_current_tl \c@g__tag_struct_abs_int
592   %\seq_show:N \g__tag_struct_stack_seq
593   \bool_if:NF
594     \l__tag_struct_elem_stash_bool
595     {%set the parent
596       \__tag_prop_gput:cnx
597       { g__tag_struct_\int_eval:n { \c@g__tag_struct_abs_int }_prop }
598       { P }
599       {
600         \pdf_object_ref:e { __tag/struct/\l__tag_struct_stack_parent_tmpa_tl }
601       }
602       %record this structure as kid:
603       %\tl_show:N \g__tag_struct_stack_current_tl
604       %\tl_show:N \l__tag_struct_stack_parent_tmpa_tl
605       \__tag_struct_kid_struct_gput_right:xx
606       { \l__tag_struct_stack_parent_tmpa_tl }
607       { \g__tag_struct_stack_current_tl }
608       %\prop_show:c { g__tag_struct_\g__tag_struct_stack_current_tl _prop }
609       %\seq_show:c { g__tag_struct_kids_\l__tag_struct_stack_parent_tmpa_tl _seq }
610     }
611     %\prop_show:c { g__tag_struct_\g__tag_struct_stack_current_tl _prop }
612     %\seq_show:c { g__tag_struct_kids_\l__tag_struct_stack_parent_tmpa_tl _seq }
613   <debug> \__tag_debug_struct_begin_insert:n { #1 }
614   \group_end:
615   }
616   <debug> { \__tag_debug_struct_begin_ignore:n { #1 } }

```

```

617 }
618 <package>\cs_set_protected:Nn \tag_struct_end:
619 <debug>\cs_set_protected:Nn \tag_struct_end:
620 { %take the current structure num from the stack:
621   %the objects are written later, lua mode hasn't all needed info yet
622   %\seq_show:N \g__tag_struct_stack_seq
623 <package>\__tag_check_if_active_struct:T
624 <debug>\__tag_check_if_active_struct:TF
625 {
626   \seq_gpop:NN \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
627   \seq_gpop:NNTF \g__tag_struct_stack_seq \l__tag_tmpa_tl
628   {
629     \__tag_check_info_closing_struct:o { \g__tag_struct_stack_current_tl }
630   }
631   { \__tag_check_no_open_struct: }
632   % get the previous one, shouldn't be empty as the root should be there
633   \seq_get:NNTF \g__tag_struct_stack_seq \l__tag_tmpa_tl
634   {
635     \tl_gset:NV \g__tag_struct_stack_current_tl \l__tag_tmpa_tl
636   }
637   {
638     \__tag_check_no_open_struct:
639   }
640   \seq_get:NNT \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
641   {
642     \tl_gset:NV \g__tag_struct_tag_tl \l__tag_tmpa_tl
643   }
644 <debug>\__tag_debug_struct_end_insert:
645 }
646 <debug>{\__tag_debug_struct_end_ignore:}
647 }
648 </package|debug>

```

(End definition for \tag_struct_begin:n and \tag_struct_end:. These functions are documented on page 81.)

\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.

```

649 <base>\cs_new_protected:Npn \tag_struct_use:n #1 {}
650 (*package)
651 \cs_set_protected:Npn \tag_struct_use:n #1 %#1 is the label
652 {
653   \__tag_check_if_active_struct:T
654   {
655     \prop_if_exist:cTF
656     { g__tag_struct\___tag_ref_value:enn{tagpdfstruct-#1}{tagstruct}{unknown}_prop } %
657     {
658       \__tag_check_struct_used:n {#1}
659       %add the label structure as kid to the current structure (can be the root)
660       \__tag_struct_kid_struct_gput_right:xx
661       { \g__tag_struct_stack_current_tl }
662       { \__tag_ref_value:enn{tagpdfstruct-#1}{tagstruct}{0} }
663       %add the current structure to the labeled one as parents
664       \__tag_prop_gput:cnx

```

```

665         { g__tag_struct_\__tag_ref_value:enn{tagpdfstruct-#1}{tagstruct}{0}_prop }
666         { P }
667         {
668             \pdf_object_ref:e { __tag/struct/\g__tag_struct_stack_current_t1 }
669         }
670     }
671     {
672         \msg_warning:nnn{ tag }{struct-label-unknown}{#1}
673     }
674 }
675 }
676 \endpackage

```

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

`\tag_struct_object_ref:n` This is a command that allows to reference a structure. The argument is the number which can be get for the current structure with `\tag_get:n{struct_num}` TODO check if it should be in base too.

```

677 \begin{package}
678 \cs_new:Npn \tag_struct_object_ref:n #1
679 {
680     \pdf_object_ref:n {__tag/struct/#1}
681 }
682 \cs_generate_variant:Nn \tag_struct_object_ref:n {e}

```

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

`\tag_struct_gput:nnn` This is a command that allows to update the data of a structure. The first argument is the number of the structure, the second a keyword referring to a function, the third the value. Currently the only keyword is `ref`

```

683 \cs_new_protected:Npn \tag_struct_gput:nnn #1 #2 #3
684 {
685     \cs_if_exist_use:cF {__tag_struct_gput_data_#2:nn}
686     { %warning??
687         \use_none:nn
688     }
689     {#1}{#3}
690 }
691 \cs_generate_variant:Nn \tag_struct_gput:nnn {ene,nne}

```

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

`__tag_struct_gput_data_ref:nn`

```

692 \cs_new_protected:Npn \__tag_struct_gput_data_ref:nn #1 #2
693     % #1 receiving struct num, #2 list of object ref
694     {
695         \prop_get:cnN
696         { g__tag_struct_#1_prop }
697         {Ref}
698         \l__tag_tmpb_t1
699         \__tag_prop_gput:cnx
700         { g__tag_struct_#1_prop }
701         { Ref }
702         { \quark_if_no_value:NF\l__tag_tmpb_t1 { \l__tag_tmpb_t1\c_space_tl }#2 }
703     }
704 \cs_generate_variant:Nn \__tag_struct_gput_data_ref:nn {ee}

```

(End definition for `_tag_struct_gput_data_ref:nn`.)

`\tag_struct_insert_annot:nn` This are the user command to insert annotations. They must be used together to get the numbers right. They use a counter to the `StructParent` and `\tag_struct_insert_annot:nn` increases the counter given back by `\tag_struct_parent_int:`.

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.

```

705 \cs_new_protected:Npn \tag_struct_insert_annot:nn #1 #2 %#1 should be an object reference
706                                     %#2 struct parent num
707 {
708   \__tag_check_if_active_struct:T
709   {
710     \__tag_struct_insert_annot:nn {#1}{#2}
711   }
712 }
713
714 \cs_generate_variant:Nn \tag_struct_insert_annot:nn {xx}
715 \cs_new:Npn \tag_struct_parent_int: {\int_use:c {c@g__tag_parenttree_obj_int }}
716
717 </package>
718

```

(End definition for `\tag_struct_insert_annot:nn` and `\tag_struct_parent_int:`. These functions are documented on page 81.)

7 Attributes and attribute classes

```

719 <*header>
720 \ProvidesExplPackage {tagpdf-attr-code} {2022-08-24} {0.97}
721 {part of tagpdf - code related to attributes and attribute classes}
722 </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 name. `\l__tag_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__tag_attr_objref_prop`

```

723 <*package>
724 \prop_new:N \g__tag_attr_entries_prop
725 \seq_new:N \g__tag_attr_class_used_seq
726 \tl_new:N \l__tag_attr_value_tl
727 \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` expects two brace group, the name and the content. The content typically needs an `/0` key for the owner. An example look like this.

`newattribute{setup-key}`

```

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

728 \cs_new_protected:Npn \__tag_attr_new_entry:nn #1 #2 %#1:name, #2: content
729 {
730   \prop_gput:Nen \g__tag_attr_entries_prop
731   {\pdf_name_from_unicode_e:n{#1}}{#2}
732 }
733
734 \keys_define:nn { __tag / setup }
735 {
736   newattribute .code:n =
737   {
738     \__tag_attr_new_entry:nn #1
739   }
740 }

```

(End definition for `__tag_attr_new_entry:nn` and `newattribute` (setup-key). This function is documented on page 84.)

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

```

741 \keys_define:nn { __tag / struct }
742 {
743   attribute-class .code:n =
744   {
745     \clist_set:No \l__tag_tmpa_clist { #1 }
746     \seq_set_from_clist:NN \l__tag_tmpb_seq \l__tag_tmpa_clist
747     \seq_set_map_x:NNN \l__tag_tmpa_seq \l__tag_tmpb_seq
748     {
749       \pdf_name_from_unicode_e:n {##1}
750     }
751     \seq_map_inline:Nn \l__tag_tmpa_seq
752     {
753       \prop_if_in:NnF \g__tag_attr_entries_prop {##1}
754       {
755         \msg_error:nnn { tag } { attr-unknown } { ##1 }
756       }
757       \seq_gput_left:Nn \g__tag_attr_class_used_seq { ##1 }
758     }
759     \tl_set:Nx \l__tag_tmpa_tl
760     {
761       \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}
762       \seq_use:Nn \l__tag_tmpa_seq { \c_space_tl }
763       \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}
764     }
765     \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 0 }
766     {

```

```

767         \_tag_prop_gput:cnx
768         { g__tag_struct\_int_eval:n {\c@g__tag_struct_abs_int}_prop }
769         { C }
770         { \l__tag_tmpa_tl }
771         %\prop_show:c { g__tag_struct\_int_eval:n {\c@g__tag_struct_abs_int}_prop }
772     }
773 }
774 }

```

(End definition for attribute-class (struct-key). This function is documented on page 83.)

attribute_ (struct-key)

```

775 \keys_define:nn { __tag / struct }
776 {
777     attribute .code:n = % A property (attribute, value currently a dictionary)
778     {
779         \clist_set:No          \l__tag_tmpa_clist { #1 }
780         \seq_set_from_clist:NN \l__tag_tmpb_seq \l__tag_tmpa_clist
781
782         we convert the names into pdf names with slash
783
784         \seq_set_map_x:NnNn \l__tag_tmpa_seq \l__tag_tmpb_seq
785         {
786             \pdf_name_from_unicode_e:n {##1}
787         }
788         \tl_set:Nx \l__tag_attr_value_tl
789         {
790             \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}%
791         }
792         \seq_map_inline:Nn \l__tag_tmpa_seq
793         {
794             \prop_if_in:NnF \g__tag_attr_entries_prop {##1}
795             {
796                 \msg_error:nnn { tag } { attr-unknown } { ##1 }
797             }
798             \prop_if_in:NnF \g__tag_attr_objref_prop {##1}
799             {%\prop_show:N \g__tag_attr_entries_prop
800              \pdf_object_unnamed_write:nx
801              { dict }
802              {
803                  \prop_item:Nn\g__tag_attr_entries_prop {##1}
804              }
805              \prop_gput:Nnx \g__tag_attr_objref_prop {##1} {\pdf_object_ref_last:}
806          }
807         \tl_put_right:Nx \l__tag_attr_value_tl
808         {
809             \c_space_tl
810             \prop_item:Nn \g__tag_attr_objref_prop {##1}
811         }
812         % \tl_show:N \l__tag_attr_value_tl
813         }
814         \tl_put_right:Nx \l__tag_attr_value_tl
815         { %[
816             \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}%
817         }
818         % \tl_show:N \l__tag_attr_value_tl

```

```

816         \_tag_prop_gput:cnx
817         { g__tag_struct\_int_eval:n {\c@g__tag_struct_abs_int}_prop }
818         { A }
819         { \l__tag_attr_value_tl }
820     },
821 }
822 \end{package}

```

(End definition for attribute (struct-key). This function is documented on page [83](#).)

Part VIII

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

```
1 <@@=tag>
2 <*luatex>
3 \ProvidesExplFile {tagpdf-luatex.def} {2022-08-24} {0.97}
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
19 \cs_set_protected:Npn \__tag_prop_new:N #1
20 {
21   \prop_new:N #1
22   \lua_now:e { ltx.__tag.tables.\cs_to_str:N#1 = {} }
23 }
24
25 \cs_set_protected:Npn \__tag_seq_new:N #1
26 {
27   \seq_new:N #1
28   \lua_now:e { ltx.__tag.tables.\cs_to_str:N#1 = {} }
29 }
30
31 \cs_set_protected:Npn \__tag_prop_gput:Nnn #1 #2 #3
32 {
33   \prop_gput:Nnn #1 { #2 } { #3 }
34   \lua_now:e { ltx.__tag.tables.\cs_to_str:N#1 ["#2"] = "#3" }
35 }
36
37
38
39
```

```

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 \langle /luatex \rangle

The module declaration

```

63  $\langle$ *lua $\rangle$ 
64 -- tagpdf.lua
65 -- Ulrike Fischer
66
67 local ProvidesLuaModule = {
68   name      = "tagpdf",
69   version   = "0.97",      --TAGVERSION
70   date      = "2022-08-24", --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 leve
120 ltx.__tag.trace.show_all_mc_data (max,loglevel): shows a maximum about mc's if the current l
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 tabl
168 ltx.__tag.page = ltx.__tag.page or { } -- page data, currently only i->{0->mcnum,1->mcr
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> 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>(End definition for <code>__tag_get_tag_from</code>, <code>ltx.__tag.func.get_tag_from</code>, and <code>ltx.__tag.func.output_tag_from</code>.)</p>
<pre> ltx.__tag.func.store_mc_data </pre>	<p>This function stores for <code>key=data</code> for mc-chunk <code>num</code>. It is used in the <code>tagpdf-mc</code> 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>(End definition for <code>ltx.__tag.func.store_mc_data</code>.)</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>(End definition for <code>ltx.__tag.func.store_mc_label</code>.)</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>(End definition for <code>ltx.__tag.func.store_mc_kid</code>.)</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 `\pdfsfakespace`

```

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 "..mcpntprev ..
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             mcopen,mcpagencnt,mccntprev,mctypeprev=
572             ltx.__tag.func.mark_page_elements (n,mcpagencnt,mccntprev,mcopen,"INTERNAL HLIST",mctypeprev)
573         elseif n.id == VLIST then -- enter the vlist
574             mcopen,mcpagencnt,mccntprev,mctypeprev=
575             ltx.__tag.func.mark_page_elements (n,mcpagencnt,mccntprev,mcopen,"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 mcopen~=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                     mcopen = mcopen - 1
602                     ltx.__tag.trace.log ("INFO TAG-INSERT-EMC: " ..
603                                         mcpagencnt .. " MCOPEN = " .. mcopen,3)
604                     if mcopen ~=0 then
605                         ltx.__tag.trace.log ("WARN TAG-OPEN-MC: " .. mcopen,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     mccntprev = mccnt
654 end
655 end -- end if
656 end -- end for
657 if head then
658     mccnthead, 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(mccnthead)..

```

```

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-
ltx.__tag.func.output_parenttree 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} {2022-08-24} {0.97}
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:n {tag/NS/#1}
18   \pdfdict_new:n {g__tag_role/Namespace_#1_dict}
19   \pdf_object_new:n {_tag/RoleMapNS/#1}
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 %\_tag_role_NS_new:nnn {latex} {https://www.latex-project.org/ns/2022}{ }
61 \exp_args:Nnx
62 \_tag_role_NS_new:nnn {user}{\_c\_tag_role_userNS_id_str}{ }
63 }

```

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

```

```

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

```

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

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

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

```

318     ,sec
319     ,sech
320     ,selector
321     ,semantics
322     ,sep
323     ,set
324     ,setdiff
325     ,share
326     ,sin
327     ,sinh
328     ,subset
329     ,sum
330     ,tan
331     ,tanh
332     ,tendsto
333     ,times
334     ,transpose
335     ,true
336     ,union
337     ,uplimit
338     ,variance
339     ,vector
340     ,vectorproduct
341     ,xor
342 }
343
344 \prop_const_from_keyval:Nn \c__tag_role_sttags_pdfII_to_pdf_prop
345 {
346     DocumentFragment = Art,
347     Aside = Note,
348     Title = H1,
349     Sub    = Span,
350     H7     = H6 ,
351     H8     = H6 ,
352     H9     = H6 ,
353     H10    = H6 ,
354     FENote= Note,
355     Em     = Span,
356     Strong= Span,
357 }

```

(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.

```

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

```

```

368     \_tag_seq_gput_right:Nn \g__tag_role_tags_seq { #1 }
369     \prop_gput:Nnn \g__tag_role_tags_NS_prop { #1 }{ pdf }
370 }
371 \clist_map_inline:Nn \c__tag_role_sttags_only_pdfII_clist
372 {
373     \_tag_seq_gput_right:Nn \g__tag_role_tags_seq { #1 }
374     \prop_gput:Nnn \g__tag_role_tags_NS_prop { #1 }{ pdf2 }
375 }
376 \pdf_version_compare:NnT > {1.9}
377 {
378     \clist_map_inline:Nn \c__tag_role_sttags_mathml_clist
379     {
380         \_tag_seq_gput_right:Nn \g__tag_role_tags_seq { #1 }
381         \prop_gput:Nnn \g__tag_role_tags_NS_prop { #1 }{ mathml }
382     }
383 }

```

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

```

384 \int_step_inline:nnnn { 1 }{ 1 }{ \seq_count:N \g__tag_role_tags_seq }
385 {
386     \_tag_prop_gput:Nxn \g__tag_role_tags_prop
387     {
388         \seq_item:Nn \g__tag_role_tags_seq { #1 }
389     }
390     { #1 }
391 }

```

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`

```

392 \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.

```

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

```

```

408     \__tag_check_add_tag_role:nn {#1}{#2}
409     \tl_if_empty:nF { #2 }
410     {
411         \pdfdict_gput:nnx {g__tag_role/RoleMap_dict}
412         {#1}
413         {\pdf_name_from_unicode_e:n{#2}}
414     }
415 }
416 \cs_generate_variant:Nn \__tag_role_add_tag:nn {VV}
417
418 \pdf_version_compare:NnT < {2.0}
419 {
420     \prop_map_inline:Nn \c__tag_role_sttags_pdfII_to_pdf_prop
421     {
422         \__tag_role_add_tag:nn {#1}{#2}
423     }
424 }
425

```

(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

```

426 \cs_new_protected:Nn \__tag_role_add_tag:nnnn %tag/namespace/role/namespace
427 {
428     \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
429     {
430         \msg_info:nnn { tag }{new-tag}{#1}
431     }
432     \__tag_seq_gput_right:Nn \g__tag_role_tags_seq { #1 }
433     \__tag_prop_gput:Nnx \g__tag_role_tags_prop { #1 }
434     {
435         \seq_count:N \g__tag_role_tags_seq
436     }
437     \prop_gput:Nnn \g__tag_role_tags_NS_prop { #1 }{ #2 }
438     \__tag_check_add_tag_role:nn {#1}{#3}
439     \pdfdict_gput:nnx {g__tag_role/RoleMapNS_#2_dict}{#1}
440     {
441         [
442             \pdf_name_from_unicode_e:n{#3}
443             \c_space_tl
444             \pdf_object_ref:n {tag/NS/#4}
445         ]
446     }
447 }
448 \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) 449 \keys_define:nn { __tag / tag-role }
role_(rolemap-key) 450 {
role-namespace_(rolemap-key) 451 ,tag .tl_set:N = \l__tag_role_tag_tmpa_tl
add-new-tag_(setup-key) 452 ,tag-namespace .tl_set:N = \l__tag_role_tag_namespace_tmpa_tl
453 ,role .tl_set:N = \l__tag_role_role_tmpa_tl
454 ,role-namespace .tl_set:N = \l__tag_role_role_namespace_tmpa_tl
455 }
456
457 \keys_define:nn { __tag / setup }
458 {
459 add-new-tag .code:n =
460 {
461 \keys_set_known:nnnN
462 {__tag/tag-role}
463 {
464 tag-namespace=user,
465 role-namespace=, %so that we can test for it.
466 #1
467 }{__tag/tag-role}\l_tmpa_tl
468 \tl_if_empty:NF \l_tmpa_tl
469 {
470 \exp_args:NNno \seq_set_split:Nnn \l_tmpa_seq { / } {\l_tmpa_tl/}
471 \tl_set:Nx \l__tag_role_tag_tmpa_tl { \seq_item:Nn \l_tmpa_seq {1} }
472 \tl_set:Nx \l__tag_role_role_tmpa_tl { \seq_item:Nn \l_tmpa_seq {2} }
473 }
474 \tl_if_empty:NT \l__tag_role_role_namespace_tmpa_tl
475 {
476 \prop_get:NVNTF
477 \g__tag_role_tags_NS_prop
478 \l__tag_role_role_tmpa_tl
479 \l__tag_role_role_namespace_tmpa_tl
480 {
481 \prop_if_in:NVF\g__tag_role_NS_prop \l__tag_role_role_namespace_tmpa_tl
482 {
483 \tl_set:Nn \l__tag_role_role_namespace_tmpa_tl {user}
484 }
485 }
486 {
487 \tl_set:Nn \l__tag_role_role_namespace_tmpa_tl {user}
488 }
489 }
490 \pdf_version_compare:NnTF < {2.0}
491 {
492 %TODO add check for emptyness?
493 \__tag_role_add_tag:VV
494 \l__tag_role_tag_tmpa_tl
495 \l__tag_role_role_tmpa_tl
496 }
497 {
498 \__tag_role_add_tag:VVVV
499 \l__tag_role_tag_tmpa_tl
500 \l__tag_role_tag_namespace_tmpa_tl
501 \l__tag_role_role_tmpa_tl

```

```

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

```

(End definition for tag (rolemap-key) and others. These functions are documented on page 124.)

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} {2022-08-24} {0.97}
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 \keys_define:nn { __tag / setup }
8 {
9     interwordspace .choices:nn = { true, on }
10     { \msg_warning:nnx {tag}{sys-no-interwordspace}{\c_sys_engine_str} },
11     interwordspace .choices:nn = { false, off }
12     { \msg_warning:nnx {tag}{sys-no-interwordspace}{\c_sys_engine_str} },
13     interwordspace .default:n = true,
14     show-spaces .bool_set:N = \l__tag_showspaces_bool
15 }
16 \sys_if_engine_pdftex:T
17 {
18     \sys_if_output_pdf:TF
19     {
20         \pdfglyptounicode{space}{0020}
21         \keys_define:nn { __tag / setup }
22         {
23             interwordspace .choices:nn = { true, on } { \pdfinterwordspaceon },
24             interwordspace .choices:nn = { false, off } { \pdfinterwordspaceon },
25             interwordspace .default:n = true,
```

```

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

```

(End definition for `interwordspace` (setup-key) and `show-spaces` (setup-key). These functions are documented on page 137.)

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

```

66 \sys_if_engine luatex:T
67 {
68   \cs_new_protected:Nn \__tag_fakespace:
69   {
70     \group_begin:
71     \lua_now:e{ltx.__tag.func.fakespace()}
72     \skip_horizontal:n{\c_zero_skip}
73     \group_end:
74   }

```

```

75 }
76 \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, 23
<code>_</code>	239, 253
A	
<code>activate_ (setup-key)</code>	31, 182
<code>activate-all_ (setup-key)</code>	6, 260
<code>activate-mc_ (setup-key)</code>	6, 260
<code>activate-space_ (setup-key)</code>	6, 260
<code>activate-struct_ (setup-key)</code>	6, 260
<code>activate-tree_ (setup-key)</code>	6, 260
<code>actualtext_ (mc-key)</code>	54, 198, 394
<code>actualtext_ (struct-key)</code>	82, 319
<code>add-new-tag_ (setup-key)</code>	124, 449
<code>\AddToHook</code>	13,
	16, 50, 114, 197, 231, 245, 259, 270, 308
<code>AF_ (struct-key)</code>	83, 438
<code>AFinline_ (struct-key)</code>	83, 438
<code>AFinline-o_ (struct-key)</code>	83, 438
<code>alt_ (mc-key)</code>	54, 198, 394
<code>alt_ (struct-key)</code>	82, 319
<code>artifact_ (mc-key)</code>	54, 198, 394
artifact-bool internal commands:	
<code>__artifact-bool</code>	113
artifact-type internal commands:	
<code>__artifact-type</code>	113
<code>attr-unknown</code>	19, 35
<code>attribute_ (struct-key)</code>	83, 775
<code>attribute-class_ (struct-key)</code>	83, 741
B	
bool commands:	
<code>\bool_gset_eq:NN</code>	328, 341, 353, 369
<code>\bool_gset_false:N</code>	39, 54, 190, 329, 354, 382
<code>\bool_gset_true:N</code>	38, 48, 151, 160, 336
<code>\bool_if:NTF</code>	9, 9, 18, 23, 28, 33, 37, 73, 133,
	160, 169, 177, 186, 214, 224, 227,
	233, 237, 247, 251, 269, 272, 302,
	311, 323, 336, 338, 348, 355, 364, 593
<code>\bool_if:nTF</code>	6, 266
<code>\bool_lazy_all:nTF</code>	58, 199
<code>\bool_lazy_and:nnTF</code>	75, 85, 326
<code>\bool_lazy_and_p:nn</code>	8
<code>\bool_new:N</code>	11, 15, 16, 37, 60, 146, 147, 148,
	149, 150, 152, 154, 156, 218, 219, 319
<code>\bool_set_false:N</code>	161, 191, 217,
	218, 223, 224, 236, 237, 300, 322, 347
<code>\bool_set_true:N</code>	153, 155, 228, 229, 249, 250, 299
box commands:	
<code>\box_dp:N</code>	177, 181
<code>\box_ht:N</code>	167
<code>\box_new:N</code>	141, 142
<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>	65, 178
C	
<code>\c</code>	151, 152
c@g internal commands:	
<code>\c@g__tag_MCID_abs_int</code>	9, 25, 34, 47, 54, 65, 71, 108,
	135, 163, 180, 237, 242, 271, 311, 357
<code>\c@g__tag_parenttree_obj_int</code>	52
<code>\c@g__tag_struct_abs_int</code>	6, 17, 46, 169,
	172, 174, 331, 348, 354, 367, 379,
	391, 404, 411, 423, 433, 464, 466,
	471, 482, 486, 487, 489, 490, 492,
	497, 506, 510, 511, 513, 514, 516,
	521, 551, 561, 562, 563, 564, 567,
	569, 574, 589, 591, 597, 768, 771, 817
clist commands:	
<code>\clist_const:Nn</code>	65, 111, 133, 143, 144, 149
<code>\clist_map_inline:Nn</code>	361, 366, 371, 378
<code>\clist_map_inline:nn</code>	167, 418
<code>\clist_new:N</code>	139
<code>\clist_set:Nn</code>	745, 779
color commands:	
<code>\color_select:n</code>	239, 253
cs commands:	
<code>\cs_generate_variant:Nn</code>	41, 94, 98,
	104, 116, 117, 128, 145, 153, 157,
	158, 159, 160, 161, 162, 163, 164,
	165, 166, 167, 189, 194, 208, 209,
	210, 211, 212, 213, 234, 247, 288,
	299, 416, 448, 457, 682, 691, 704, 714
<code>\cs_gset_eq:NN</code>	191
<code>\cs_if_exist:NTF</code>	116, 274, 310
<code>\cs_if_exist_p:N</code>	9, 203

`\int_step_inline:nnnn`
 46, 71, 74, 91, 220, 226, 384
`\int_to_Hex:n` 43, 44, 46, 48, 50, 52, 53
`\int_use:N` . 9, 17, 25, 34, 47, 54, 65,
 71, 77, 88, 107, 108, 126, 169, 172,
 174, 178, 180, 182, 239, 242, 253,
 266, 267, 271, 357, 482, 486, 487,
 490, 492, 506, 510, 511, 514, 516, 715
 intarray commands:
`\intarray_gset:Nnn` 198
`\intarray_item:Nn` 200, 203
`\intarray_new:Nn` 190
`interwordspace_(setup-key)` 137, 6
 iow commands:
`\iow_newline:` 171, 202
`\iow_now:Nn` 102
`\iow_term:n` 140, 143, 149, 153, 195, 277

K

keys commands:
`\keys_define:nn`
 7, 21, 30, 43, 58, 70, 113, 132,
 175, 187, 198, 224, 261, 320, 372,
 395, 449, 457, 458, 528, 734, 741, 775
`\keys_set:nn` 10,
 55, 164, 191, 278, 282, 337, 431, 573
`\keys_set_known:nnnN` 461

L

`label_(mc-key)` 54, 198, 394
`label_(struct-key)` 82, 319
`lang_(struct-key)` 82, 319
 legacy commands:
`\legacy_if:nTF` 100
`\llap` 239
`log_(setup-key)` 6, 272
 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, 43, 45, 49, 52, 52, 53,
 55, 59, 60, 60, 62, 64, 64, 71, 77, 78,
 81, 87, 89, 101, 102, 111, 122, 124,
 129, 140, 204, 212, 226, 244, 257, 267

M

`\maxdimen` 189
`mc-current` 19, 16
`mc-current_(show-key)` 32, 70
`mc-data_(show-key)` 32, 58
`mc-label-unknown` 19, 9
`mc-marks_(show-key)` 32, 132
`mc-nested` 19, 6
`mc-not-open` 19, 13
`mc-popped` 19, 14
`mc-pushed` 19, 14
`mc-tag-missing` 19, 8
`mc-used-twice` 19, 12
`\MessageBreak` 15, 19, 20, 21
 msg commands:
`\msg_error:nn` 98, 119, 276, 586
`\msg_error:nnn` 181, 263, 755, 793
`\msg_error:nnnn` 263
`\msg_info:nnn` . 112, 169, 173, 399, 430
`\msg_info:nnnn` 142
`\msg_line_context:` . 283, 284, 316, 320
`\g_msg_module_name_prop` 30, 34
`\g_msg_module_type_prop` 33
`\msg_new:nnn` 7, 8, 9, 12, 13, 14, 15,
 16, 22, 24, 25, 28, 29, 31, 33, 35, 36,
 37, 38, 39, 40, 41, 43, 283, 284, 314, 318
`\msg_new:nnnn` 46
`\msg_note:nn` 128
`\msg_note:nnn` 304, 311, 342, 350
`\msg_note:nnnn` 290, 297, 327, 335
`\msg_warning:nn` 108

\msg_warning:nnn		\pdf_version_compare_p:Nn	204
..... 10, 12, 33, 39, 48, 105,		pdfannot commands:	
128, 135, 146, 154, 162, 185, 208, 672		\pdfannot_dict_put:nnn	
\msg_warning:nnnn	335 159, 406, 429, 447, 452	
		\pdfannot_link_ref_last: ...	416, 439
		pdfdict commands:	
N		\pdfdict_gput:nnn	
namespace_(rolemap-key)	124 21, 28, 36, 198, 411, 439	
new-tag	19, 39	\pdfdict_if_empty:nTF	192
newattribute_(setup-key)	84, 728	\pdfdict_new:n	18, 20, 392
\newcommand	296, 297	\pdfdict_use:n	151, 196, 203
\newcounter	6, 8, 52	\pdffakespace	33, 211
\NewDocumentCommand	6,	pdffile commands:	
14, 20, 25, 31, 37, 42, 47, 53, 213, 301		\pdffile_embed_stream:nnn	
\newlabeldata	104 160, 484, 508	
\newmarks	14	\pdfglyphtounicode	20
no-struct-dest_(setup-key)	6, 260	\pdfinterwordspacelon	23, 24
\nointerlineskip	182	pdfmanagement commands:	
		\pdfmanagement_add:nnn	
P	 25, 26, 231, 286, 288, 290	
\PackageError	13	\pdfmanagement_if_active_p: ..	9, 10
\PackageWarning	28	\pdfmanagement_remove:nn	292
para-hook-count-wrong	20, 46	prg commands:	
paratagging_(setup-key)	33, 224	\prg_do_nothing:	
paratagging-show_(setup-key) ...	33, 224 79, 191, 391, 392, 393, 394	
parent_(struct-key)	82, 319	\prg_generate_conditional_-	
pdf commands:		variant:Nnn	157
\pdf_activate_structure_destination:		\prg_new_conditional:Nnn	59, 222
..... 203, 208		\prg_new_conditional:Npnn	
\pdf_bdc:nn	233 52, 73, 83, 258, 264, 275	
\pdf_bmc:n	231	\prg_new_eq_conditional:NnN .	73, 229
\l_pdf_current_structure_-		\prg_return_false:	
destination_tl	207	. 53, 69, 70, 80, 90, 226, 261, 273, 279	
\pdf_emc:	232	\prg_return_true:	
\pdf_name_from_unicode_e:n 67, 70, 77, 87, 225, 262, 272, 278	
..... 350, 413, 442, 731, 749, 783		\prg_set_conditional:Npnn	56
\pdf_object_if_exist:n	157	\ProcessOptions	41
\pdf_object_if_exist:nTF		prop commands:	
..... 100, 172, 250, 462, 482, 506, 532		\prop_clear:N	73
\pdf_object_new:n		\prop_const_from_keyval:Nn	344
..... 17, 19, 20, 51, 65, 146, 176, 566		\prop_count:N	94
\pdf_object_new:nn	66, 187	\prop_get:NnN	695
\pdf_object_ref:n	29, 37,	\prop_get:NnNTF	
39, 41, 93, 102, 124, 158, 174, 184,	 93, 96, 123, 138, 247, 351, 476	
199, 298, 444, 447, 454, 600, 668, 680		\prop_gpop:NnN	246
\pdf_object_ref_last:	141, 802	\prop_gput:Nnn . 25, 30, 33, 34, 39,	
\pdf_object_unnamed_write:nn	133, 797	95, 98, 100, 147, 161, 201, 238, 295,	
\pdf_object_write:nn	69, 206	364, 369, 374, 381, 406, 437, 730, 802	
\pdf_object_write:nnn ...	63, 67,	\prop_gremove:Nn	241
73, 84, 94, 141, 149, 177, 194, 201, 255		\prop_if_exist:NnTF	655
\pdf_pageobject_ref:n	100, 289	\prop_if_exist_p:N	328
\pdf_string_from_unicode:nnN ...	25	\prop_if_in:NnTF	63, 95,
\pdf_uncompress:	294	103, 183, 230, 395, 481, 753, 791, 795	
\pdf_version_compare:NnTF			
..... 55, 376, 418, 490			

`\prop_item:Nn` 34, 67, 83, 114, 162, 163,
204, 237, 240, 309, 343, 359, 800, 807
`\prop_map_inline:Nn` 190, 420
`\prop_map_tokens:Nn` 208
`\prop_new:N`
.... 9, 10, 11, 136, 199, 231, 724, 727
`\prop_put:Nnn` 80, 162
`\prop_show:N`
.... 58, 92, 206, 608, 611, 771, 796
`\ProvidesExplFile` 3
`\ProvidesExplPackage` 3,
3, 3, 3, 3, 3, 3, 3, 3, 7, 26, 46, 720

Q

`\quad` 162, 163
quark commands:
`\quark_if_no_value:NTF` 247, 702

R

`raw_␣(mc-key)` 54, 198, 394
`ref_␣(struct-key)` 83, 319
ref commands:
`\ref_attribute_gset:nnnn`
..... 168, 170, 177, 179, 181
`\ref_label:nn` 164, 186, 280
`\ref_value:nn` 82, 421
`\ref_value:nnn` 6, 116, 116, 118, 192, 197
ref internal commands:
`__ref_value:nnn` 121, 124
regex commands:
`\regex_replace_once:nnN` 150
`\renewcommand` 299, 300
`\RenewDocumentCommand` 8
`\RequirePackage` 20, 42, 43, 304, 307, 313, 316
`\rlap` 253
`role_␣(rolemap-key)` 124, 449
`role-missing` 19, 36
`role-namespace_␣(rolemap-key)` .. 124, 449
`role-tag` 19, 39
`role-unknown` 19, 36
`role-unknown-tag` 19, 36
`root-AF_␣(setup-key)` 84, 528

S

`\selectfont` 6
seq commands:
`\seq_clear:N` 162, 225
`\seq_const_from_clist:Nn` 20, 33
`\seq_count:N` 174,
384, 404, 435, 761, 763, 765, 787, 813
`\seq_get:NNTF` 272, 582, 633, 640
`\seq_gpop:NN` 626
`\seq_gpop:NNTF` 97, 627
`\seq_gpop_left:NN` 149
`\seq_gpush:Nn` . 12, 14, 80, 87, 589, 590
`\seq_gput_left:Nn` 154, 757
`\seq_gput_right:Nn` .. 32, 134, 202, 245
`\seq_gremove_duplicates:N` 157
`\seq_gset_eq:NN` 156, 169, 218
`\seq_if_empty:NTF` 197
`\seq_item:Nn` 113, 115,
122, 126, 133, 137, 193, 203, 268,
270, 277, 344, 345, 358, 388, 471, 472
`\seq_log:N` . 172, 174, 178, 196, 328, 343
`\seq_map_inline:Nn` . 163, 220, 751, 789
`\seq_new:N` 11, 13, 13,
15, 16, 17, 18, 18, 19, 137, 138, 200, 725
`\seq_put_right:Nn` 164
`\seq_remove_all:Nn` 167
`\seq_set_eq:NN` 204, 205
`\seq_set_from_clist:NN` 746, 780
`\seq_set_from_clist:Nn` 84, 87, 193, 213
`\seq_set_map:NNn` 158
`\seq_set_map_x:NNn` 747, 781
`\seq_set_split:Nnn` 165, 343, 470
`\seq_show:N` . 51, 145, 146, 165, 166,
168, 179, 205, 255, 592, 609, 612, 622
`\seq_use:Nn`
107, 108, 162, 163, 169, 202, 205, 762
`\l_tmpa_seq` 225, 245, 255, 470, 471, 472
shipout commands:
`\g_shipout_readonly_int`
..... 107, 126, 178, 234
`show-spaces_␣(setup-key)` 137, 6
`\ShowTagging` 17, 32, 52
skip commands:
`\skip_horizontal:n` 72
`\c_zero_skip` 72
`stash_␣(mc-key)` 54, 113
`stash_␣(struct-key)` 82, 319
`\stepcounter` 301
str commands:
`\str_case:nnTF` 52
`\str_const:Nn` 41
`\str_if_eq:nnTF` 71, 82, 124, 277
`\str_if_eq_p:nn` 268, 270
`\str_new:N` 135
`\str_set_convert:Nnnn` .. 166, 219,
237, 361, 373, 385, 398, 408, 419, 427
`\str_use:N` 230, 250
`\l_tmpa_str` 25, 26, 31
`\string` 20, 21, 22, 285
`struct-faulty-nesting` 19, 25
`struct-label-unknown` 19, 31
`struct-missing-tag` 19, 28
`struct-no-objnum` 19, 24
`struct-show-closing` 19, 33

struct-stack_␣(show-key) 32, [175](#)
 struct-unknown [22](#)
 struct-used-twice 19, [29](#)
 sys commands:
 \c_sys_backend_str 52, 71, 82
 \c_sys_engine_str 10, 12
 \sys_if_engine luatex:TF
 38, 41, 62, 66, 75, 211, 296
 \sys_if_engine pdftex:TF 16
 \sys_if_output_pdf:TF 11, 18
 sys-no-interwordspace 20, [43](#)

T

tabsorder_␣(setup-key) 6, [284](#)
 tag_␣(mc-key) 54, [198](#), [394](#)
 tag_␣(rolemap-key) 124, [449](#)
 tag_␣(struct-key) 82, [319](#)
 tag commands:
 \tag_get:n
 . . . 17, 81, 82, 93, [50](#), 50, 80, 83, 316
 \tag_if_active: 52, 56
 \tag_if_active:TF 17, [51](#), 234
 \tag_if_active_p: 17, [51](#)
 \tag_mc_artifact_group_begin:n . .
 53, [54](#), 54, 57
 \tag_mc_artifact_group_end:
 53, [54](#), 55, 64
 \tag_mc_begin:n 10, [53](#),
 16, 60, 105, [154](#), 154, 238, 242, 252,
 [318](#), 318, 322, 328, 331, 357, 405, 428
 \tag_mc_begin_pop:n
 . 53, 68, [71](#), 72, 93, 338, 366, 419, 442
 \tag_mc_end: 53,
 22, 67, 84, [185](#), 185, 240, 250, 254,
 [318](#), 319, 335, 362, 369, 375, 417, 440
 \tag_mc_end_push:
 . 53, 59, [71](#), 71, 74, 325, 350, 403, 426
 \tag_mc_if_in: 73, 229
 \tag_mc_if_in:TF 53, 33, [59](#), [222](#)
 \tag_mc_if_in_p: 53, [59](#), [222](#)
 \tag_mc_use:n 53, 27, 30, 30, 32
 \tag_start: 6, [214](#), 226, 256
 \tag_start:n 6, [214](#), 244, 258
 \tag_stop: 6, [214](#), 221, 255
 \tag_stop:n 6, [214](#), 232, 257
 \tag_stop_group_begin: 6, 61, [214](#), 214
 \tag_stop_group_end: . 6, 66, [214](#), 220
 \tag_struct_begin:n 81, 39,
 236, 356, 404, 427, [551](#), 551, 554, 555
 \tag_struct_end: 81, 44,
 256, 363, 418, 441, [551](#), 552, 618, 619
 \tag_struct_gput:nnn . . 683, 683, 691
 \tag_struct_insert_annot:nn
 81, 102, 416, 439, [705](#), 705, 714

 \tag_struct_object_ref:n
 81, [677](#), 678, 682
 \tag_struct_parent_int:
 . 81, 102, 409, 416, 432, 439, [705](#), 715
 \tag_struct_use:n
 81, 82, 49, [649](#), 649, 651
 tag internal commands:
 __tag_activate_mark_space [415](#)
 \g__tag_active_mc_bool
 33, 61, 75, [146](#), 264
 \l__tag_active_mc_bool
 . . 64, 75, [152](#), 218, 224, 229, 237, 250
 \g__tag_active_space_bool
 9, 48, 54, [146](#), 263
 \g__tag_active_struct_bool
 60, 85, [146](#), 202, 266, 269
 \l__tag_active_struct_bool
 . . 63, 85, [152](#), 217, 223, 228, 236, 249
 \g__tag_active_struct_dest_bool . .
 [146](#), 201, 270
 \g__tag_active_tree_bool
 9, 23, 62, [146](#), 214, 227, 265
 __tag_add_document_structure:n . .
 [182](#), 182, 193
 __tag_add_missing_mcs:Nn
 66, [164](#), 164, 216
 __tag_add_missing_mcs_to_-
 stream:Nn [58](#),
 58, [186](#), 186, 277, 281, 288, 290
 \g__tag_attr_class_used_seq
 157, 158, [723](#), 757
 \g__tag_attr_entries_prop
 163, [723](#), 730, 753, 791, 796, 800
 __tag_attr_new_entry:nn
 344, [728](#), 728, 738
 \g__tag_attr_objref_prop
 [723](#), 795, 802, 807
 \l__tag_attr_value_tl
 723, 785, 804, 809, 811, 815, 819
 __tag_check_add_tag_role:nn
 [131](#), 131, 408, 438
 __tag_check_if_active_mc: 73
 __tag_check_if_active_mc:TF [73](#),
 76, 95, 156, 187, 188, 324, 330, 371, 377
 __tag_check_if_active_struct: . . 83
 __tag_check_if_active_struct:TF . .
 . . 34, [73](#), 557, 558, 623, 624, 653, 708
 __tag_check_if_mc_in_galley: . . 258
 __tag_check_if_mc_in_galley:TF . .
 138, 159
 __tag_check_if_mc_tmb_missing: 264
 __tag_check_if_mc_tmb_missing:TF .
 109, 147, 164, [264](#)

__tag_check_if_mc_tmb_missing_-		__tag_exclude_headfoot_end: ...
p:	264 333 , 379 , 380
__tag_check_if_mc_tme_missing:	275	__tag_exclude_struct_headfoot_-
__tag_check_if_mc_tme_missing:TF		begin:n
.....	151 , 152 , 168 , 275 345 , 384 , 385
__tag_check_if_mc_tme_missing_-		__tag_exclude_struct_headfoot_-
p:	275	end:
__tag_check_info_closing_-	 360 , 386 , 387
struct:n	108 , 108 , 116 , 629	__tag_fakespace
__tag_check_init_mc_used: 66 , 68 , 215
.....	188 , 188 , 191 , 197	__tag_finish_structure:
__tag_check_mc_if_nested: 13 , 16 , 211 , 212
.....	150 , 150 , 159 , 335	__tag_get_data_mc_tag:
__tag_check_mc_if_open: 197 , 197 , 316 , 316
.....	150 , 158 , 189 , 381	__tag_get_data_struct_num:
__tag_check_mc_in_galley:TF ..	258 313 , 314
__tag_check_mc_in_galley_p: ..	258	__tag_get_data_struct_tag:
__tag_check_mc_pushed_popped:nn	 304 , 304
.....	81 , 88 , 101 , 104 , 109 , 165 , 165	__tag_get_mathsubtype
__tag_check_mc_tag:N 250
.....	166 , 177 , 177 , 345	__tag_get_mc_abs_cnt:
__tag_check_mc_used:n 9 ,
.....	133 , 193 , 193 , 291 9 , 19 , 20 , 64 , 93 , 94 , 105 , 154 , 162 ,
\g_tag_check_mc_used_intarray 179 , 181 , 206 , 214 , 230 , 248 , 261 , 271
.....	188 , 198 , 200 , 203	__tag_get_mc_cnt_type_tag
__tag_check_no_open_struct: 244
.....	117 , 117 , 631 , 638	__tag_get_num_from
__tag_check_show_MCID_by_page: 258
.....	212 , 212	__tag_get_tag_from
__tag_check_struct_used:n 277
.....	121 , 121 , 658	__tag_hook_kernel_after_foot: ..
__tag_check_structure_has_tag:n	 306 , 315 , 380 , 387 , 394
.....	93 , 93 , 574	__tag_hook_kernel_after_head: ..
__tag_check_structure_tag:N 304 , 313 , 379 , 386 , 393
.....	101 , 101 , 346	__tag_hook_kernel_before_foot: ..
__tag_check_typeout_v:n 305 , 314 , 378 , 385 , 392
.....	45 , 45 , 107 , 108 , 111 ,	__tag_hook_kernel_before_head: ..
.....	146 , 154 , 161 , 199 , 208 , 277 , 280 , 285 303 , 312 , 377 , 384 , 391
__tag_debug_mc_begin_ignore:n ..		\g_tag_in_mc_bool
.....	293 , 364 11 , 18 , 160 , 190 , 224 ,
__tag_debug_mc_begin_insert:n 328 , 329 , 336 , 341 , 353 , 354 , 369 , 382
.....	286 , 332	__tag_insert_bdc_node
__tag_debug_mc_end_ignore:	307 , 389 329
__tag_debug_mc_end_insert:	300 , 379	__tag_insert_bmc_node
__tag_debug_struct_begin_-	 315
ignore:n	331 , 616	__tag_lastpagelabel: ...
__tag_debug_struct_begin_-	 97 , 98 , 115
insert:n	323 , 613	__tag_log
__tag_debug_struct_end_ignore: 172
.....	346 , 646	\l_tag_loglevel_int
__tag_debug_struct_end_insert: 110 , 140 , 145 , 168 , 170 , 171 , 195 ,
.....	338 , 644 273 , 276 , 279 , 280 , 281 , 288 , 295 ,
__tag_exclude_headfoot_begin: 302 , 309 , 325 , 333 , 340 , 348 , 397 , 428
.....	320 , 377 , 378	__tag_mark_spaces
	 356
		__tag_mc_artifact_begin_marks:n
	 20 , 42 , 78 , 342
		\l_tag_mc_artifact_bool
	 15 , 116 , 161 , 169 , 191 , 338
		\l_tag_mc_artifact_type_tl
	 14 , 120 , 124 , 128 ,
	 132 , 136 , 140 , 144 , 148 , 284 , 340 , 342
		__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	\g__tag_mc_parenttree_prop
..... 20, 20, 41, 77, 349 12, 13, 83, 100, 148, 295
__tag_mc_bmc:n	\l__tag_mc_ref_abbrev_t1
..... 230, 231, 302 12, 238, 250, 258, 266
__tag_mc_bmc_artifact: 300, 300, 313	__tag_mc_set_label_used:n 25, 25, 45
__tag_mc_bmc_artifact:n 300, 304, 314	\g__tag_mc_stack_seq 13, 80, 87, 97, 174
\l__tag_mc_botmarks_seq	__tag_mc_store:nnn . 90, 90, 104, 131
..... 66, 18, 87, 108,	\l__tag_mc_tmpa_t1 .. 13, 252, 255, 259
146, 158, 163, 205, 213, 218, 260, 277	g__tag_MCID_abs_int
__tag_mc_disable_marks: 75, 75	7
__tag_mc_emc: 155, 230, 232, 384	\g__tag_MCID_byabbrev_prop
__tag_mc_end_marks: . 20, 60, 79, 385 10, 248, 257, 265
\l__tag_mc_firstmarks_seq	\g__tag_MCID_tmp_bypage_int
..... 66, 18, 84, 107, 145, 162, 10, 182, 255, 263, 276
193, 196, 197, 204, 205, 260, 268, 270	\g__tag_mode_lua_bool
\g__tag_mc_footnote_marks_seq ... 15 37, 38, 39, 73, 133, 160,
__tag_mc_get_marks: 81, 81, 137, 158	186, 272, 302, 311, 323, 336, 348, 364
__tag_mc_handle_artifact:N	__tag_new_output_prop_handler:n
..... 116, 300, 308, 340 61, 71, 80, 563
__tag_mc_handle_mc_label:n	__tag_pairs_prop
..... 21, 21, 174, 353	189
__tag_mc_handle_mcid:nn	\g__tag_para_begin_int
..... 235, 283, 288, 346 220, 235, 239, 261, 266
__tag_mc_handle_stash:n 44,	\l__tag_para_bool
131, 131, 153, 179, 289, 289, 299, 357	218, 226, 233, 247, 299, 300, 322, 347
__tag_mc_if_in: 59, 73, 222, 229	\g__tag_para_end_int
__tag_mc_if_in:TF 59, 78, 152, 160, 222 221, 249, 253, 261, 267
__tag_mc_if_in_p:	\g__tag_para_int
59, 222	218
__tag_mc_insert_extra_tmb:n ...	\l__tag_para_show_bool
..... 105, 105, 168 218, 227, 237, 251
__tag_mc_insert_extra_tme:n ...	\l__tag_para_tag_t1
..... 105, 150, 169 222, 223, 228, 236, 242
__tag_mc_insert_mcid_kids:n ...	__tag_parenttree_add_objr:nn ...
..... 122, 122, 138, 151 60, 60, 293
__tag_mc_insert_mcid_single_kids:n	\l__tag_parenttree_content_t1 ...
..... 122, 127, 152 67, 86, 98, 112, 120, 140, 143
\l__tag_mc_key_label_t1	\g__tag_parenttree_objr_t1 59, 62, 140
..... 17, 171, 174, 256, 349, 350, 353, 427	__tag_pdf_object_ref
\l__tag_mc_key_properties_t1 ...	336
..... 17, 162, 211, 224, 225,	__tag_prop_gput:Nnn
242, 243, 348, 404, 413, 414, 424, 425 9, 23, 34, 38, 83, 90,
\l__tag_mc_key_stash_bool	181, 191, 199, 201, 202, 208, 256,
..... 15, 28, 37, 115, 177, 355	264, 347, 353, 366, 378, 386, 390,
\g__tag_mc_key_tag_t1	402, 403, 410, 432, 433, 465, 491,
..... 17, 19, 194, 197, 203, 316, 383, 400	515, 535, 568, 596, 664, 699, 767, 816
\l__tag_mc_key_tag_t1	__tag_prop_item:Nn .. 9, 43, 199, 204
166, 168, 193, 202, 345, 347, 349, 399	__tag_prop_new:N
__tag_mc_lua_set_mc_type_attr:n	8,
..... 74, 74, 98, 168	9, 9, 10, 11, 12, 79, 199, 199, 210, 562
__tag_mc_lua_unset_mc_type_attr:	__tag_prop_show:N 9, 56, 199, 206, 213
74, 100, 192	__tag_ref_label:nn
\g__tag_mc_main_marks_seq 23, 183, 183, 189, 269, 578
15	__tag_ref_value:nnn
\g__tag_mc_marks	36,
22, 31, 44, 51, 62, 68, 85, 88, 194, 214	78, 82, 100, 101, 124, 190, 190,
\g__tag_mc_multicol_marks_seq ... 15	194, 237, 240, 248, 289, 656, 662, 665
	__tag_ref_value_lastpage:nn ...
 57, 71, 74, 195, 195, 216, 230
	\c__tag_refmc_clist
	143

\c__tag_refstruct_clist	143	\g__tag_struct_cont_mc_prop	10 , 92 , 93 , 95 , 98 , 114
g__tag_role/RoleMap_dict	392	\l__tag_struct_elem_stash_bool ..	60 , 323 , 594
__tag_role_add_tag:nn	393 , 393 , 416 , 422 , 493	__tag_struct_exchange_kid_-	
__tag_role_add_tag:nnnn	426 , 426 , 448 , 498	command:N	147 , 147 , 157 , 188
__tag_role_NS_new:nnn	125 , 15 , 15 , 57 , 58 , 59 , 60 , 62	__tag_struct_fill_kid_key:n ...	158 , 158 , 252
\g__tag_role_NS_prop	10 , 39 , 190 , 208 , 351 , 481	__tag_struct_format_Ref:n	246 , 246 , 247
\l__tag_role_role_namespace_-		__tag_struct_get_dict_content:nN	217 , 217 , 253
tmpa_tl	11 , 454 , 474 , 479 , 481 , 483 , 487 , 502	__tag_struct_gput_data_ref:nn ..	423 , 692 , 692 , 704
\l__tag_role_role_tmpa_tl	11 , 453 , 472 , 478 , 495 , 501	__tag_struct_insert_annot:nn ...	266 , 266 , 710
\c__tag_role_sttags_mathml_clist	64 , 378	\l__tag_struct_key_label_tl	59 , 322 , 576 , 578
\c__tag_role_sttags_only_pdf_-		__tag_struct_kid_mc_gput_-	
clist	64 , 366	right:nn	94 , 104 , 117 , 292
\c__tag_role_sttags_only_pdfII_-		__tag_struct_kid_OBJR_gput_-	
clist	64 , 371	right:nnn	129 , 129 , 145 , 281
\c__tag_role_sttags_pdf_pdfII_-		__tag_struct_kid_struct_gput_-	
clist	64 , 361	right:nn ...	119 , 119 , 128 , 605 , 660
\c__tag_role_sttags_pdfII_to_-		g__tag_struct_kids_0_seq	79
pdf_prop	64 , 420	__tag_struct_mcid_dict:n	94 , 95 , 98 , 109
\l__tag_role_tag_namespace_tmpa_-		\g__tag_struct_objR_seq	8
tl	11 , 452 , 500	__tag_struct_output_prop_aux:nn	61 , 61 , 75
\l__tag_role_tag_tmpa_tl	11 , 451 , 471 , 494 , 499	\g__tag_struct_stack_current_tl .	15 , 26 , 35 , 66 , 72 , 78 , 136 , 144 , 150 , 207 , 293 , 297 , 309 , 316 , 591 , 603 , 607 , 608 , 611 , 629 , 635 , 661 , 668
\g__tag_role_tags_NS_prop	9 , 183 , 343 , 364 , 369 , 374 , 381 , 406 , 437 , 477	\l__tag_struct_stack_parent_-	
\g__tag_role_tags_prop	6 , 103 , 138 , 359 , 386 , 395 , 402 , 433	tmpa_tl .	15 , 274 , 283 , 298 , 333 , 572 , 580 , 584 , 600 , 604 , 606 , 609 , 612
\g__tag_role_tags_seq	6 , 358 , 363 , 368 , 373 , 380 , 384 , 388 , 401 , 404 , 432 , 435	\g__tag_struct_stack_seq	11 , 273 , 583 , 589 , 592 , 622 , 627 , 633
\c__tag_role_userNS_id_str	125 , 41 , 62	\c__tag_struct_StructElem_-	
\g__tag_saved_in_mc_bool	319 , 328 , 341 , 353 , 369	entries_seq	20
__tag_seq_gput_right:Nn	9 , 30 , 106 , 111 , 121 , 138 , 199 , 202 , 209 , 363 , 368 , 373 , 380 , 401 , 432	\c__tag_struct_StructTreeRoot_-	
__tag_seq_item:Nn ...	9 , 38 , 199 , 203	entries_seq	20
__tag_seq_new:N	7 , 9 , 9 , 16 , 81 , 199 , 200 , 211 , 564	\g__tag_struct_tag_NS_tl	57 , 345 , 351
__tag_seq_show:N .	9 , 49 , 199 , 205 , 212	\g__tag_struct_tag_stack_seq ...	13 , 178 , 179 , 328 , 343 , 590 , 626 , 640
__tag_show_spacemark	342	\g__tag_struct_tag_tl	57 , 344 , 346 , 350 , 590 , 642
\l__tag_showspaces_bool ...	14 , 26 , 35	__tag_struct_write_obj:n	42 , 48 , 248 , 248
__tag_space_chars_shipout	436	\g__tag_tagunmarked_bool ...	156 , 282
\g__tag_state_prop .	231 , 238 , 241 , 246	\l__tag_tmpa_box	133 , 168 , 174 , 175 , 179 , 190 , 191
g__tag_struct_0_prop	79		
__tag_struct_add_AF:nn	438 , 457 , 464 , 488 , 512 , 534		

<code>\l__tag_tmpa_clist</code>	<code>\tagmcend</code>	31, 13
..... 133, 745, 746, 779, 780	<code>tagmcid</code>	6, 168
<code>\l__tag_tmpa_int</code>	<code>\tagmcifin</code>	31
..... 133	<code>\tagmcifinTF</code>	31, 30
<code>\l__tag_tmpa_prop</code> .	<code>\tagmcuse</code>	31, 13
..... 73, 81, 94, 96, 133	<code>\tagpdfifluatexT</code>	31
<code>\l__tag_tmpa_seq</code>	<code>\tagpdfifluatexTF</code>	31
..... 133, 158, 162, 164, 166, 167, 168,	<code>\tagpdfifpdfTeX</code>	31
169, 170, 343, 344, 345, 747, 751,	<code>\tagpdfparaOff</code>	33, 296
761, 762, 763, 765, 781, 787, 789, 813	<code>\tagpdfparaOn</code>	33, 296
<code>\l__tag_tmpa_str</code>	<code>\tagpdfsetup</code>	31, 83, 84, 124, 6
..... 135, 220, 225, 230, 238, 243,	<code>\tagpdfsuppressmarks</code>	33, 301
250, 362, 369, 374, 381, 386, 393,	<code>tagstruct</code>	6, 168
399, 406, 409, 414, 420, 425, 428, 435	<code>\tagstructbegin</code>	32, 124, 36, 184
<code>\l__tag_tmpa_tl</code>	<code>\tagstructend</code>	32, 36, 185
..... 36, 37, 44, 77, 84, 93, 95,	<code>tagstructobj</code>	6, 168
96, 97, 99, 100, 102, 104, 105,	<code>\tagstructuse</code>	32, 36
115, 116, 133, 149, 153, 154, 156,	<code>tagunmarked_␣(setup-key)</code>	6, 282
167, 174, 180, 214, 222, 246, 247,	TeX and L ^A T _E X 2 _ε commands:	
253, 259, 351, 356, 417, 420, 423,	<code>\@M</code>	165
626, 627, 633, 635, 640, 642, 759, 770	<code>\@auxout</code>	102
<code>\l__tag_tmpb_box</code>	<code>\@bsphack</code>	185
..... 133, 169, 176, 177, 181, 183	<code>\@cclv</code>	281
<code>\l__tag_tmpb_seq</code> 133, 746, 747, 780, 781	<code>\@esphack</code>	187
<code>\l__tag_tmpb_tl</code>	<code>\@gobble</code>	24, 48
..... 134, 698, 702	<code>\@ifpackageloaded</code>	28
<code>\l__tag_tmpb_tl_{␣␣␣␣}\l__tag_tmpa_</code>	<code>\@kernel@after@foot</code>	315
<code>str</code>	<code>\@kernel@after@head</code>	313
..... 133	<code>\@kernel@before@cclv</code>	278
<code>__tag_tree_fill_parenttree:</code> ...	<code>\@kernel@before@foot</code>	314
..... 68, 69, 138	<code>\@kernel@before@footins</code>	274, 276
<code>__tag_tree_lua_fill_parenttree:</code>	<code>\@kernel@before@head</code>	310, 312
..... 118, 118, 135	<code>\@makecol</code>	280
<code>__tag_tree_write_classmap:</code>	<code>\@maxdepth</code>	178
..... 154, 154, 219	<code>\@mult@ptagging@hook</code>	283
<code>__tag_tree_write_namespaces:</code> ...	<code>\@secondoftwo</code>	24, 48
..... 188, 188, 220	<code>\c@page</code>	280
<code>__tag_tree_write_parenttree:</code> ...	<code>\count@</code>	288
..... 131, 131, 217	<code>\mult@firstbox</code>	286
<code>__tag_tree_write_rolemap:</code>	<code>\mult@rightbox</code>	290
..... 147, 147, 218	<code>\page@sofar</code>	285
<code>__tag_tree_write_structelements:</code>	<code>\process@cols</code>	286
..... 44, 44, 221	tex commands:	
<code>__tag_tree_write_structtreeroot:</code>	<code>\tex_botmarks:D</code>	88
..... 32, 32, 222	<code>\tex_firstmarks:D</code>	85
<code>__tag_whatsits:</code> 30, 54, 55, 58, 318, 319	<code>\tex_immediate:D</code>	75, 86
<code>tag-namespaces_␣(rolemap-key)</code>	<code>\tex_kern:D</code>	181
..... 449	<code>\tex_marks:D</code>	22, 31, 44, 51, 62, 68
<code>tag/struct/0</code> internal commands:	<code>\tex_pdfextension:D</code>	86
<code>__tag/struct/0</code>	<code>\tex_pdfobj:D</code>	75
..... 20	<code>\tex_special:D</code>	58
<code>tag/tree/namespaces</code> internal commands:	<code>\tex_splitbotmarks:D</code>	214
<code>__tag/tree/namespaces</code>	<code>\tex_splitfirstmarks:D</code>	194
..... 187	<code>\the</code>	280
<code>tag/tree/parenttree</code> internal commands:		
<code>__tag/tree/parenttree</code>		
..... 51		
<code>tag/tree/rolemap</code> internal commands:		
<code>__tag/tree/rolemap</code>		
..... 146		
<code>tagabspace</code>		
..... 6, 168		
<code>tagmcabs</code>		
..... 6, 168		
<code>\tagmcbegin</code>		
..... 31, 13		

<code>\tiny</code>	239, 253	132, 136, 140, 144, 148, 167, 193,
<code>title□(struct-key)</code>	82, <u>319</u>	202, 207, 214, 223, 238, 256, 333,
<code>title-o□(struct-key)</code>	82, <u>319</u>	399, 471, 472, 483, 487, 572, 759, 785
tl commands:		<code>\tl_show:N</code>
<code>\c_space_tl</code>	64,	603, 604, 809, 815
67, 88, 89, 97, 99, 101, 104, 143,		<code>\tl_tail:n</code>
160, 210, 234, 280, 443, 702, 762, 806		307
<code>\tl_clear:N</code>	156, 162, 219, 417	<code>\tl_to_str:n</code> 27, 42, 78, 89, 122, 283, 316
<code>\tl_gput_right:Nn</code>	62, 445	<code>\tl_use:N</code>
<code>\tl_gset:Nn</code>	17, 78, 194, 203,	127, 470, 496, 520, 540
344, 345, 383, 400, 452, 591, 635, 642		<code>\l_tmpa_tl</code>
<code>\tl_if_empty:NTF</code>	26, 37,	126, 138, 467, 468, 470
171, 173, 179, 312, 350, 468, 474, 575		token commands:
<code>\tl_if_empty:nTF</code>	34, 133, 409	<code>\token_to_str:N</code>
<code>\tl_if_eq:NNTF</code>	260	104, 280
<code>\tl_if_eq:NnTF</code>	99	tree-mcid-index-wrong
<code>\tl_if_exist:NTF</code>	126, 440	20, <u>41</u>
<code>\tl_new:N</code>	11, 12, 12, 13, 13,	
14, 14, 16, 17, 18, 19, 19, 20, 27, 57,		
58, 59, 59, 67, 133, 134, 222, 450, 726		
<code>\tl_put_left:Nn</code>	313, 315	
<code>\tl_put_right:Nn</code>	86,	
98, 111, 140, 211, 224, 225, 227,		
242, 243, 276, 278, 283, 312, 314,		
404, 413, 414, 420, 424, 425, 804, 811		
<code>\tl_set:Nn</code>		
.. 36, 77, 115, 120, 120, 124, 128,		

U

<code>\unskip</code>	31
use commands:	
<code>\use:N</code>	50, 302
<code>\use_ii:nn</code>	208
<code>\use_none:n</code>	45, 78
<code>\use_none:nn</code>	77, 687

V

<code>\vbadness</code>	165, 189
vbox commands:	
<code>\vbox_set_split_to_ht:Nn</code>	191
<code>\vbox_set_to_ht:Nnn</code>	167
<code>\vbox_unpack_drop:N</code>	180
<code>\vfuzz</code>	166