

tagpdf – A package to experiment with pdf tagging^{*}

Ulrike Fischer[†]

Released 2023-06-14

Contents

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

^{*}This file describes v0.98i, last revised 2023-06-14.

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

2	Description of log messages	16
2.1	\ShowTagging command	16
2.2	Messages in checks and commands	17
2.3	Messages from the ptagging code	17
2.4	Warning messages from the lua-code	17
2.5	Info messages from the lua-code	17
2.6	Debug mode messages and code	18
2.7	Messages	18
3	Messages	19
3.1	Messages related to mc-chunks	19
3.2	Messages related to structures	20
3.3	Attributes	21
3.4	Roles	22
3.5	Miscellaneous	22
4	Retrieving data	23
5	User conditionals	23
6	Internal checks	24
6.1	checks for active tagging	24
6.2	Checks related to structures	25
6.3	Checks related to roles	26
6.4	Check related to mc-chunks	27
6.5	Checks related to the state of MC on a page or in a split stream	29
II The tagpdf-user module		
Code related to L^AT_EX2e user commands and document commands		
Part of the tagpdf package		33
1	Setup commands	33
2	Commands related to mc-chunks	33
3	Commands related to structures	34
4	Debugging	34
5	Extension commands	34
5.1	Fake space	35
5.2	Paratagging	35
5.3	Header and footer	35
5.4	Link tagging	36
6	User commands and extensions of document commands	36
7	Setup and preamble commands	36
8	Commands for the mc-chunks	37

9	Commands for the structure	37
10	Debugging	38
11	Commands to extend document commands	41
11.1	new ref system	41
11.2	Document structure	41
11.3	Structure destinations	41
11.4	Fake space	42
11.5	Paratagging	42
11.6	Header and footer	46
11.7	Links	48
III	The <code>tagpdf-tree</code> module	
	Commands trees and main dictionaries	
	Part of the <code>tagpdf</code> package	50
1	Trees, pdfmanagement and finalization code	50
1.1	Check structure	50
1.2	Catalog: MarkInfo and StructTreeRoot	51
1.3	Writing the IDtree	51
1.4	Writing structure elements	52
1.5	ParentTree	53
1.6	Rolemap dictionary	56
1.7	Classmap dictionary	57
1.8	Namespaces	57
1.9	Finishing the structure	58
1.10	StructParents entry for Page	59
IV	The <code>tagpdf-mc-shared</code> module	
	Code related to Marked Content (<code>mc-chunks</code>), code shared by	
	all modes	
	Part of the <code>tagpdf</code> package	60
1	Public Commands	60
2	Public keys	61
3	Marked content code – shared	62
3.1	Variables and counters	62
3.2	Functions	63
3.3	Keys	66
V	The <code>tagpdf-mc-generic</code> module	
	Code related to Marked Content (<code>mc-chunks</code>), generic mode	
	Part of the <code>tagpdf</code> package	68

1	Marked content code – generic mode	68
1.1	Variables	68
1.2	Functions	69
1.3	Looking at MC marks in boxes	72
1.4	Keys	79
VI	The <code>tagpdf-mc-luacode</code> module	
	Code related to Marked Content (mc-chunks), luamode-specific	
	Part of the tagpdf package	81
1	Marked content code – luamode code	81
1.1	Commands	82
1.2	Key definitions	87
VII	The <code>tagpdf-struct</code> module	
	Commands to create the structure	
	Part of the tagpdf package	89
1	Public Commands	89
2	Public keys	90
2.1	Keys for the structure commands	90
2.2	Setup keys	92
3	Variables	92
3.1	Variables used by the keys	94
3.2	Variables used by tagging code of basic elements	95
4	Commands	95
4.1	Initialization of the StructTreeRoot	96
4.2	Adding the /ID key	97
4.3	Filling in the tag info	97
4.4	Handlings kids	98
4.5	Output of the object	101
5	Keys	105
6	User commands	110
7	Attributes and attribute classes	117
7.1	Variables	117
7.2	Commands and keys	118
VIII	The <code>tagpdf-luatex.def</code>	
	Driver for luatex	
	Part of the tagpdf package	121
1	Loading the lua	121

2	Logging functions	125
3	Helper functions	127
3.1	Retrieve data functions	127
3.2	Functions to insert the pdf literals	129
4	Function for the real space chars	130
5	Function for the tagging	133
6	Parenttree	138
IX	The tagpdf-roles module	
	Tags, roles and namespace code	
	Part of the tagpdf package	140
1	Code related to roles and structure names	140
1.1	Variables	141
1.2	Namespaces	143
1.3	Adding a new tag	144
1.3.1	pdf 1.7 and earlier	145
1.3.2	The pdf 2.0 version	147
1.4	Helper command to read the data from files	148
1.5	Reading the default data	150
1.6	Parent-child rules	150
1.6.1	Reading in the csv-files	151
1.6.2	Retrieving the parent-child rule	153
1.7	Remapping of tags	157
1.8	Key-val user interface	159
X	The tagpdf-space module	
	Code related to real space chars	
	Part of the tagpdf package	161
1	Code for interword spaces	161
	Index	164

<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. They 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_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 <code>luamode</code>) unmarked text should be marked up as artifact. The initial value is <code>true</code> .
--------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------

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

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

1 Initialization and test if pdfmanagement is active.

```

1 <@@=tag>
2 <*package>
3 \ProvidesExplPackage {tagpdf} {2023-06-14} {0.98i}
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} {2023-06-14} {0.98i}
29 { debug code for tagpdf }
30 \@ifpackageloaded{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 base package

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

```

36 <*base>
37 \ProvidesExplPackage {tagpdf-base} {2023-06-14} {0.98i}
38 {part of tagpdf - provide base, no-op versions of the user commands }
39 </base>

```

3 Package options

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

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

4 Packages

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

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

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

```
47 \RequirePackage{tagpdf-base}
48 </package>
```

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

```
49 <*base>
50 \AddToHook{begindocument}
51 {
52   \str_case:N\F \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>
```

4.1 a LastPage label

See also issue #2 in Accessible-xref

__tag_lastpagelabel:

```
62 <*package>
63 \cs_new_protected:Npn \__tag_lastpagelabel:
64 {
65   \legacy_if:nT { @files }
66   {
67     \exp_args:NNnx \exp_args:NNx\iow_now:Nn \@auxout
68     {
69       \token_to_str:N \newlabeldata
70       {__tag_LastPage}
71       {
72         {abspage} { \int_use:N \g_shipout_readonly_int}
73         {tagmcabs}{ \int_use:N \c@g__tag_MCID_abs_int }
```



```

\c__tag_refmc_clist
\c__tag_refstruct_clist 112 \clist_const:Nn \c__tag_refmc_clist {tagabspage,tagmcabs,tagmcid}
113 \clist_const:Nn \c__tag_refstruct_clist {tagstruct,tagstructobj}

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

\l__tag_loglevel_int This integer hold the log-level and so allows to control the messages. TODO: a list which
log-level shows what is needed. The current behaviour is quite ad-hoc.
114 \int_new:N \l__tag_loglevel_int

(End of definition for \l__tag_loglevel_int.)

\g__tag_active_space_bool These booleans should help to control the global behaviour of tagpdf. Ideally it should
\g__tag_active_mc_bool more or less do nothing if all are false. The space-boolean controles the interword space
\g__tag_active_tree_bool code, the mc-boolean activates \tag_mc_begin:n, the tree-boolean activates writing the
\g__tag_active_struct_bool finish code and the pdfmanagement related commands, the struct-boolean activates the
\g__tag_active_struct_dest_bool 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.

115 \bool_new:N \g__tag_active_space_bool
116 \bool_new:N \g__tag_active_mc_bool
117 \bool_new:N \g__tag_active_tree_bool
118 \bool_new:N \g__tag_active_struct_bool
119 \bool_new:N \g__tag_active_struct_dest_bool
120 \bool_gset_true:N \g__tag_active_struct_dest_bool

(End of definition for \g__tag_active_space_bool and others.)

\l__tag_active_mc_bool These booleans should help to control the local behaviour of tagpdf. In some cases it
\l__tag_active_struct_bool 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.

121 \bool_new:N \l__tag_active_mc_bool
122 \bool_set_true:N \l__tag_active_mc_bool
123 \bool_new:N \l__tag_active_struct_bool
124 \bool_set_true:N \l__tag_active_struct_bool

(End of 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 used it in generic mode, but
this would create quite a lot empty artifact mc-chunks.

125 \bool_new:N \g__tag_tagunmarked_bool

(End of definition for \g__tag_tagunmarked_bool.)

```

6 Variants of l3 commands

```

126 \prg_generate_conditional_variant:Nnn \pdf_object_if_exist:n {e}{T,F}
127 \cs_generate_variant:Nn \pdf_object_ref:n {e}
128 \cs_generate_variant:Nn \pdfannot_dict_put:nnn {nnx}
129 \cs_generate_variant:Nn \pdffile_embed_stream:nnn {nxx,oxx}
130 \cs_generate_variant:Nn \prop_gput:Nnn {Nxx,Nen}
131 \cs_generate_variant:Nn \prop_put:Nnn {Nxx}
132 \cs_generate_variant:Nn \prop_item:Nn {No,Ne}
133 \cs_generate_variant:Nn \ref_label:nn { nv }
134 \cs_generate_variant:Nn \seq_set_split:Nnn{Nne}
135 \cs_generate_variant:Nn \str_set_convert:Nnnn {Nonn, Noon, Nnon }
136 \cs_generate_variant:Nn \clist_map_inline:nn {on}

```

7 Setup label attributes

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

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

```

137 \ref_attribute_gset:nnnn { tagstruct } {0} { now }
138 { \int_use:N \c@g__tag_struct_abs_int }
139 \ref_attribute_gset:nnnn { tagstructobj } {} { now }
140 {
141   \pdf_object_if_exist:eT {__tag/struct/\int_use:N \c@g__tag_struct_abs_int}
142   {
143     \pdf_object_ref:e{__tag/struct/\int_use:N \c@g__tag_struct_abs_int}
144   }
145 }
146 \ref_attribute_gset:nnnn { tagabspage } {0} { shipout }
147 { \int_use:N \g_shipout_readonly_int }
148 \ref_attribute_gset:nnnn { tagmcabs } {0} { now }
149 { \int_use:N \c@g__tag_MCID_abs_int }
150 \ref_attribute_gset:nnnn {tagmcid } {0} { now }
151 { \int_use:N \g__tag_MCID_tmp_bypage_int }

```

(End of 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.
152 \cs_new_protected:Npn \__tag_ref_label:nn #1 #2 %#1 label, #2 name of list mc or struct
153 {
154   \@bsphack
155   \ref_label:nv {#1}{c__tag_ref#2_clist}
156   \@esphack
157 }
158 \cs_generate_variant:Nn \__tag_ref_label:nn {en}

```

(End of 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.

```

159 \cs_new:Npn \__tag_ref_value:nnn #1 #2 #3 %#1 label, #2 attribute, #3 default
160 {
161     \ref_value:nnn {#1}{#2}{#3}
162 }
163 \cs_generate_variant:Nn \__tag_ref_value:nnn {enn}

```

(End of 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.

```

164 \cs_new:Npn \__tag_ref_value_lastpage:nn #1 #2
165 {
166     \ref_value:nnn {\__tag_LastPage}{#1}{#2}
167 }

```

(End of 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
168 \cs_set_eq:NN \__tag_prop_new:N \prop_new:N
169 \cs_set_eq:NN \__tag_seq_new:N \seq_new:N
170 \cs_set_eq:NN \__tag_prop_gput:Nnn \prop_gput:Nnn
171 \cs_set_eq:NN \__tag_seq_gput_right:Nn \seq_gput_right:Nn
172 \cs_set_eq:NN \__tag_seq_item:cn \seq_item:cn
173 \cs_set_eq:NN \__tag_prop_item:cn \prop_item:cn
174 \cs_set_eq:NN \__tag_seq_show:N \seq_show:N
175 \cs_set_eq:NN \__tag_prop_show:N \prop_show:N
176
177 \cs_generate_variant:Nn \__tag_prop_gput:Nnn { NxN , Nxx , Nnx , cnn , cxn , cnx , cno }
178 \cs_generate_variant:Nn \__tag_seq_gput_right:Nn { Nx , No , cn , cx }
179 \cs_generate_variant:Nn \__tag_prop_new:N { c }
180 \cs_generate_variant:Nn \__tag_seq_new:N { c }
181 \cs_generate_variant:Nn \__tag_seq_show:N { c }
182 \cs_generate_variant:Nn \__tag_prop_show:N { c }

```

(End of 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
`\tag_stop_group_end:` booleans. In some cases tagging should only restart, if it actually was stopped before.
`\tag_stop:` For this it is possible to label a stop.
`\tag_start:`
`\tag_stop:n`
`\tag_start:n`

```

185     \group_begin:
186     \bool_set_false:N \l__tag_active_struct_bool
187     \bool_set_false:N \l__tag_active_mc_bool
188   }
189 \cs_set_eq:NN \tag_stop_group_end: \group_end:
190 \cs_set_protected:Npn \tag_stop:
191 {
192   \bool_set_false:N \l__tag_active_struct_bool
193   \bool_set_false:N \l__tag_active_mc_bool
194 }
195 \cs_set_protected:Npn \tag_start:
196 {
197   \bool_set_true:N \l__tag_active_struct_bool
198   \bool_set_true:N \l__tag_active_mc_bool
199 }
200 \prop_new:N\g__tag_state_prop
201 \cs_set_protected:Npn \tag_stop:n #1
202 {
203   \tag_if_active:TF
204   {
205     \bool_set_false:N \l__tag_active_struct_bool
206     \bool_set_false:N \l__tag_active_mc_bool
207     \prop_gput:Nnn \g__tag_state_prop { #1 }{ 1 }
208   }
209   {
210     \prop_gremove:Nn \g__tag_state_prop { #1 }
211   }
212 }
213 \cs_set_protected:Npn \tag_start:n #1
214 {
215   \prop_gpop:NnN \g__tag_state_prop {#1}\l__tag_tmpa_tl
216   \quark_if_no_value:NF \l__tag_tmpa_tl
217   {
218     \bool_set_true:N \l__tag_active_struct_bool
219     \bool_set_true:N \l__tag_active_mc_bool
220   }
221 }
222 </package>
223 <*base>
224 \cs_new_protected:Npn \tag_stop:{}
225 \cs_new_protected:Npn \tag_start:{}
226 \cs_new_protected:Npn \tag_stop:n #1 {}
227 \cs_new_protected:Npn \tag_start:n #1 {}
228 </base>

```

(End of 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)
 activate-mc_␣(setup-key)
 activate-tree_␣(setup-key)
 activate-struct_␣(setup-key)
 activate-all_␣(setup-key)
 no-struct-dest_␣(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.

```

229 (*package)
230 \keys_define:nn { __tag / setup }
231 {
232   activate-space .bool_gset:N = \g__tag_active_space_bool,
233   activate-mc    .bool_gset:N = \g__tag_active_mc_bool,
234   activate-tree  .bool_gset:N = \g__tag_active_tree_bool,
235   activate-struct .bool_gset:N = \g__tag_active_struct_bool,
236   activate-all  .meta:n =
237     {activate-mc={#1},activate-tree={#1},activate-struct={#1}},
238   activate-all  .default:n = true,
239   no-struct-dest .bool_gset_inverse:N = \g__tag_active_struct_dest_bool,
240

```

(End of 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`.

```

241   log .choice:,
242   log / none .code:n = {\int_set:Nn \l__tag_loglevel_int { 0 }},
243   log / v .code:n =
244     {
245       \int_set:Nn \l__tag_loglevel_int { 1 }
246       \cs_set_protected:Nn \__tag_check_typeof_v:n { \iow_term:x {##1} }
247     },
248   log / vv .code:n = {\int_set:Nn \l__tag_loglevel_int { 2 }},
249   log / vvv .code:n = {\int_set:Nn \l__tag_loglevel_int { 3 }},
250   log / all .code:n = {\int_set:Nn \l__tag_loglevel_int { 10 }},

```

(End of 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`.

```

251   tagunmarked .bool_gset:N = \g__tag_tagunmarked_bool,
252   tagunmarked .initial:n = true,

```

(End of 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.

```

253   tabsorder .choice:,
254   tabsorder / row .code:n =
255     \pdfmanagement_add:nnn { Page } {Tabs}{/R},
256   tabsorder / column .code:n =
257     \pdfmanagement_add:nnn { Page } {Tabs}{/C},
258   tabsorder / structure .code:n =
259     \pdfmanagement_add:nnn { Page } {Tabs}{/S},
260   tabsorder / none .code:n =
261     \pdfmanagement_remove:nn {Page} {Tabs},
262   tabsorder .initial:n = structure,

```

```

263     uncompress      .code:n = { \pdf_uncompress: },
264   }

```

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

12 loading of engine/more dependent code

```

265 \sys_if_engine luatex:T
266 {
267   \file_input:n {tagpdf-luatex.def}
268 }
269 </package>

270 <*mcloading>
271 \bool_if:NTF \g__tag_mode_lua_bool
272 {
273   \RequirePackage {tagpdf-mc-code-lua}
274 }
275 {
276   \RequirePackage {tagpdf-mc-code-generic} %
277 }
278 </mcloading>

279 <*debug>
280 \bool_if:NTF \g__tag_mode_lua_bool
281 {
282   \RequirePackage {tagpdf-debug-lua}
283 }
284 {
285   \RequirePackage {tagpdf-debug-generic} %
286 }
287 </debug>

```

Part I

The **tagpdf-checks** module

Messages and check code

Part of the tagpdf package

1 Commands

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

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

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

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

`\tag_if_box_tagged:` *NTF* * This tests if a box contains tagging commands. It relies currently on that the code that saved the box correctly set the command `\l_tag_box_\int_use:N #1_tl` to a positive value. The LaTeX commands will do that automatically at some time but it is in the responsibility of the user to ensure that when using low-level code. If the internal command doesn't exist the box is assumed to be tagged.

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

2.3 Messages from the ptagging code

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

2.4 Warning messages from the lua-code

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

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

2.5 Info messages from the lua-code

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

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

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

2.6 Debug mode messages and code

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

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

2.7 Messages

mc-nested
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.
------------------------------------------------------------------------------------------------------------	-----------------------------------

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

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

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

```

1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-checks-code} {2023-06-14} {0.98i}
4 {part of tagpdf - code related to checks, conditionals, debugging and messages}
5 </header>

```

3 Messages

3.1 Messages related to mc-chunks

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

```

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

```

(End of definition for mc-nested. This function is documented on page 18.)

mc-tag-missing If the tag is missing

```

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

```

(End of definition for mc-tag-missing. This function is documented on page 18.)

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

```

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

```

(End of definition for mc-label-unknown. This function is documented on page 18.)

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

```

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

```

(End of definition for mc-used-twice. This function is documented on page 18.)

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

```

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

```

(End of definition for mc-not-open. This function is documented on page 18.)

mc-pushed Informational messages about mc-pushing.

mc-popped

```

14 \msg_new:nnn { tag } {mc-pushed} { #1~has~been~pushed~to~the~mc~stack}
15 \msg_new:nnn { tag } {mc-popped} { #1~has~been~removed~from~the~mc~stack }

```

(End of definition for mc-pushed and mc-popped. These functions are documented on page 18.)

mc-current Informational messages about current mc state.

```

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

```

(End of definition for mc-current. This function is documented on page 18.)

3.2 Messages related to structures

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

```

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

```

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

struct-no-objnum Should not happen ...

```

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

```

(End of definition for struct-no-objnum. This function is documented on page 19.)

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

```

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

```

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

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

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

(End of definition for struct-faulty-nesting. This function is documented on page 19.)

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

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

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

struct-used-twice

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

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

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

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

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

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

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

(End of definition for struct-show-closing. This function is documented on page 19.)

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

```
42 \msg_new:nnn { tag } {tree-struct-still-open}  
43   {  
44     There-are-still-open-structures-on-the-stack!\\  
45     The-stack-contains-\seq_use:Nn\g__tag_struct_tag_stack_seq{,}.\\  
46     The-structures-are-automatically-closed,\\  
47     but-their-nesting-can-be-wrong.  
48   }
```

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

3.3 Attributes

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

attr-unknown

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

(End of 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 50 \msg_new:nnn { tag } {role-missing} { tag~#1~has~no~role~assigned }

role-unknown-tag 51 \msg_new:nnn { tag } {role-unknown} { role~#1~is~not~known }

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

(End of definition for role-missing, role-unknown, and role-unknown-tag. These functions are documented on page 19.)

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

53 \msg_new:nnn { tag } {role-parent-child}

54 { Rule~'~#1'~-->~'~#2'~is~#3~\msg_line_context:}

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

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

55 \msg_new:nnn { tag } {role-remapping}

56 { remapping~tag~to~#1 }

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

role-tag Info messages.

new-tag 57 \msg_new:nnn { tag } {role-tag} { mapping~tag~#1~to~role~#2 }

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

59 \msg_new:nnn { tag } {read-namespace} { reading~namespace~definitions~tagpdf~ns~#1.def }

60 \msg_new:nnn { tag } {namespace-missing}{ namespace~definitions~tagpdf~ns~#1.def~not~found }

61 \msg_new:nnn { tag } {namespace-unknown}{ namespace~#1~is~not~declared }

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

3.5 Miscellaneous

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

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

63 {something~is~wrong~with~the~mcid~--rerun}

(End of definition for tree-mcid-index-wrong. This function is documented on page 19.)

sys-no-interwordspace Currently only pdf_latex and lua_latex have some support for real spaces.

64 \msg_new:nnn { tag } {sys-no-interwordspace}

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

(End of definition for sys-no-interwordspace. This function is documented on page 19.)

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

66 \cs_set_eq:NN __tag_check_typeout_v:n \use_none:n

(End of definition for __tag_check_typeout_v:n.)

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

```

67 \msg_new:nnnn { tag } {para-hook-count-wrong}
68   {The-number-of-automatic-begin-(#1)-and-end-(#2)-#3-para-hooks-differ!}
69   {This-quite-probably-a-coding-error-and-the-structure-will-be-wrong!}
70 \package

```

(End of definition for para-hook-count-wrong. This function is documented on page 19.)

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.

```

71 <base>\cs_new:Npn \tag_get:n #1 { \use:c {__tag_get_data_#1: } }

```

(End of definition for \tag_get:n. This function is documented on page 16.)

5 User conditionals

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

```

72 <*base>
73 \prg_new_conditional:Npnn \tag_if_active: { p , T , TF, F }
74   { \prg_return_false: }
75 </base>
76 <*package>
77 \prg_set_conditional:Npnn \tag_if_active: { p , T , TF, F }
78   {
79     \bool_lazy_all:nTF
80     {
81       {\g__tag_active_struct_bool}
82       {\g__tag_active_mc_bool}
83       {\g__tag_active_tree_bool}
84       {\l__tag_active_struct_bool}
85       {\l__tag_active_mc_bool}
86     }
87     {
88       \prg_return_true:
89     }
90     {
91       \prg_return_false:
92     }
93   }
94 </package>

```

(End of definition for \tag_if_active:TF. This function is documented on page 16.)

\tag_if_box_tagged_p:N This tests if a box contains tagging commands. It relies on that the code that saved the box correctly set \l_tag_box_<box number>_tl to a positive value. The LaTeX commands will do that automatically at some time but it is in the responsibility of the user to ensure that when using low-level code. If the internal command doesn't exist the box is assumed to be tagged.
\tag_if_box_tagged:NTF

```

95 <*base>
96 \prg_new_conditional:Npnn \tag_if_box_tagged:N #1 {p,T,F,TF}
97 {
98   \tl_if_exist:cTF {l_tag_box_\int_use:N #1_tl}
99   {
100     \int_compare:nNnTF {\tl_use:c{l_tag_box_\int_use:N #1_tl}}>{0}
101     { \prg_return_true: }
102     { \prg_return_false: }
103   }
104   {
105     \prg_return_true:
106     % warning??
107   }
108 }
109 </base>

```

(End of definition for \tag_if_box_tagged:N. This function is documented on page 16.)

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 This checks if mc are active.
\__tag_check_if_active_struct:TF
110 <*package>
111 \prg_new_conditional:Npnn \__tag_check_if_active_mc: {T,F,TF}
112 {
113   \bool_lazy_and:nnTF { \g__tag_active_mc_bool } { \l__tag_active_mc_bool }
114   {
115     \prg_return_true:
116   }
117   {
118     \prg_return_false:
119   }
120 }
121 \prg_new_conditional:Npnn \__tag_check_if_active_struct: {T,F,TF}
122 {
123   \bool_lazy_and:nnTF { \g__tag_active_struct_bool } { \l__tag_active_struct_bool }
124   {
125     \prg_return_true:
126   }
127   {
128     \prg_return_false:
129   }
130 }

```

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

6.2 Checks related to structures

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

```

131 \cs_new_protected:Npn \__tag_check_structure_has_tag:n #1 %#1 struct num
132 {
133   \prop_if_in:cnF { g__tag_struct_#1_prop }
134     {S}
135     {
136       \msg_error:nn { tag } {struct-missing-tag}
137     }
138 }
```

(End of definition for __tag_check_structure_has_tag:n.)

`__tag_check_structure_tag:N` This checks if the name of the tag is known, either because it is a standard type or has been rolemapped.

```

139 \cs_new_protected:Npn \__tag_check_structure_tag:N #1
140 {
141   \prop_if_in:NoF \g__tag_role_tags_NS_prop {#1}
142   {
143     \msg_warning:nnx { tag } {role-unknown-tag} {#1}
144   }
145 }
```

(End of definition for __tag_check_structure_tag:N.)

`__tag_check_info_closing_struct:n` This info message is issued at a closing structure, the use should be guarded by log-level.

```

146 \cs_new_protected:Npn \__tag_check_info_closing_struct:n #1 %#1 struct num
147 {
148   \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
149   {
150     \msg_info:nnn { tag } {struct-show-closing} {#1}
151   }
152 }
153
154 \cs_generate_variant:Nn \__tag_check_info_closing_struct:n {o,x}
```

(End of definition for __tag_check_info_closing_struct:n.)

`__tag_check_no_open_struct:` This checks if there is an open structure. It should be used when trying to close a structure. It errors if false.

```

155 \cs_new_protected:Npn \__tag_check_no_open_struct:
156 {
157   \msg_error:nn { tag } {struct-faulty-nesting}
158 }
```

(End of definition for __tag_check_no_open_struct:.)

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

```

159 \cs_new_protected:Npn \__tag_check_struct_used:n #1 %#1 label
160 {
161   \prop_get:cnNT
162     {g__tag_struct_\__tag_ref_value:enn{tagpdfstruct-#1}{tagstruct}{unknown}_prop}
```

```

163     {P}
164     \l__tag_tmpa_tl
165     {
166         \msg_warning:nnn { tag } {struct-used-twice} {#1}
167     }
168 }

```

(End of definition for __tag_check_struct_used:n.)

6.3 Checks related to roles

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

```

169 \cs_new_protected:Npn \__tag_check_add_tag_role:nn #1 #2 %#1 tag, #2 role
170 {
171     \tl_if_empty:nTF {#2}
172     {
173         \msg_error:nnn { tag } {role-missing} {#1}
174     }
175     {
176         \prop_get:NnNTF \g__tag_role_tags_NS_prop {#2} \l_tmpa_tl
177         {
178             \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
179             {
180                 \msg_info:nnnn { tag } {role-tag} {#1} {#2}
181             }
182         }
183         {
184             \msg_error:nnn { tag } {role-unknown} {#2}
185         }
186     }
187 }

```

Similar with a namespace

```

188 \cs_new_protected:Npn \__tag_check_add_tag_role:nnn #1 #2 #3 %#1 tag/NS, #2 role #3 namespace
189 {
190     \tl_if_empty:nTF {#2}
191     {
192         \msg_error:nnn { tag } {role-missing} {#1}
193     }
194     {
195         \prop_get:cnNTF { g__tag_role_NS_#3_prop } {#2} \l_tmpa_tl
196         {
197             \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
198             {
199                 \msg_info:nnnn { tag } {role-tag} {#1} {#2/#3}
200             }
201         }
202         {
203             \msg_error:nnn { tag } {role-unknown} {#2/#3}
204         }
205     }
206 }

```

(End of definition for __tag_check_add_tag_role:nn.)

6.4 Check related to mc-chunks

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

```

207 \cs_new_protected:Npn __tag_check_mc_if_nested:
208 {
209   __tag_mc_if_in:T
210   {
211     \msg_warning:nnx { tag } {mc-nested} { __tag_get_mc_abs_cnt: }
212   }
213 }
214
215 \cs_new_protected:Npn __tag_check_mc_if_open:
216 {
217   __tag_mc_if_in:F
218   {
219     \msg_warning:nnx { tag } {mc-not-open} { __tag_get_mc_abs_cnt: }
220   }
221 }

```

(End of definition for __tag_check_mc_if_nested: and __tag_check_mc_if_open:.)

`__tag_check_mc_pushed_popped:nn` This creates an information message if mc's are pushed or popped. The first argument is a word (pushed or popped), the second the tag name. With larger log-level the stack is shown too.

```

222 \cs_new_protected:Npn __tag_check_mc_pushed_popped:nn #1 #2
223 {
224   \int_compare:nNtT
225     { \l__tag_loglevel_int } = { 2 }
226     { \msg_info:nnx {tag}{mc-#1}{#2} }
227   \int_compare:nNtT
228     { \l__tag_loglevel_int } > { 2 }
229     {
230       \msg_info:nnx {tag}{mc-#1}{#2}
231       \seq_log:N \g__tag_mc_stack_seq
232     }
233 }

```

(End of definition for __tag_check_mc_pushed_popped:nn.)

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

```

234 \cs_new_protected:Npn __tag_check_mc_tag:N #1  % #1 is var with a tag name in it
235 {
236   \tl_if_empty:NT #1
237   {
238     \msg_error:nnx { tag } {mc-tag-missing} { __tag_get_mc_abs_cnt: }
239   }
240   \prop_if_in:Nof \g__tag_role_tags_NS_prop {#1}
241   {
242     \msg_warning:nnx { tag } {role-unknown-tag} {#1}
243   }
244 }

```

(End of 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??

```

245 \cs_new_protected:Npn \__tag_check_init_mc_used:
246 {
247   \intarray_new:Nn \g_tag_check_mc_used_intarray { 65536 }
248   \cs_gset_eq:NN \__tag_check_init_mc_used: \prg_do_nothing:
249 }

```

(End of definition for `\g_tag_check_mc_used_intarray` and `__tag_check_init_mc_used:`)

`__tag_check_mc_used:n` This checks if a mc is used twice.

```

250 \cs_new_protected:Npn \__tag_check_mc_used:n #1 %#1 mcid abscent
251 {
252   \int_compare:nNnT {\l__tag_loglevel_int} > { 2 }
253   {
254     \__tag_check_init_mc_used:
255     \intarray_gset:Nnn \g_tag_check_mc_used_intarray
256       {#1}
257       { \intarray_item:Nn \g_tag_check_mc_used_intarray {#1} + 1 }
258     \int_compare:nNnT
259       {
260         \intarray_item:Nn \g_tag_check_mc_used_intarray {#1}
261       }
262       >
263       { 1 }
264       {
265         \msg_warning:nnn { tag } {mc-used-twice} {#1}
266       }
267   }
268 }

```

(End of definition for `__tag_check_mc_used:n`)

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

```

269 \cs_new_protected:Npn \__tag_check_show_MCID_by_page:
270 {
271   \tl_set:Nx \l__tag_tmpa_tl
272   {
273     \__tag_ref_value_lastpage:nn
274     {abspage}
275     {-1}
276   }
277   \int_step_inline:nnnn {1}{1}
278   {
279     \l__tag_tmpa_tl
280   }
281   {
282     \seq_clear:N \l_tmpa_seq
283     \int_step_inline:nnnn

```

```

284     {1}
285     {1}
286     {
287         \__tag_ref_value_lastpage:nn
288         {tagmcabs}
289         {-1}
290     }
291     {
292         \int_compare:nT
293         {
294             \__tag_ref_value:enn
295             {mcid-###1}
296             {tagabspage}
297             {-1}
298             =
299             ##1
300         }
301         {
302             \seq_gput_right:Nx \l_tmpa_seq
303             {
304                 Page##1-###1-
305                 \__tag_ref_value:enn
306                 {mcid-###1}
307                 {tagmcid}
308                 {-1}
309             }
310         }
311     }
312     \seq_show:N \l_tmpa_seq
313 }
314 }

```

(End of definition for __tag_check_show_MCID_by_page:.)

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

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

__tag_check_mc_in_galley_p: At first we need a test to decide if \tag_mc_begin:n (tmb) and \tag_mc_end: (tme) has been used at all on the current galley. As each command issues two slightly different marks we can do it by comparing firstmarks and botmarks. The test assumes that the marks have been already mapped into the sequence with \@@_mc_get_marks:. As \seq_if_eq:NNTF doesn't exist we use the tl-test.

```

315 \prg_new_conditional:Npnn \__tag_check_if_mc_in_galley: { T,F,TF }
316 {
317     \tl_if_eq:NNTF \l__tag_mc_firstmarks_seq \l__tag_mc_botmarks_seq
318     { \prg_return_false: }
319     { \prg_return_true: }
320 }

```

(End of definition for __tag_check_mc_in_galley:TF.)

`_tag_check_if_mc_tmb_missing_p:` This checks if a extra top mark (“extra-tmb”) is needed. According to the analysis this
`_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.

```

321 \prg\_new\_conditional:Npnn \_tag\_check\_if\_mc\_tmb\_missing: { T,F,TF }
322 {
323   \bool\_if:nTF
324   {
325     \str\_if\_eq\_p:ee {\seq\_item:Nn \l\_tag\_mc\_firstmarks\_seq {1}}{e-}
326     ||
327     \str\_if\_eq\_p:ee {\seq\_item:Nn \l\_tag\_mc\_firstmarks\_seq {1}}{b+}
328   }
329   { \prg\_return\_true: }
330   { \prg\_return\_false: }
331 }

```

(End of definition for _tag_check_if_mc_tmb_missing:TF.)

`_tag_check_if_mc_tme_missing_p:` This checks if a extra bottom mark (“extra-tme”) is needed. According to the analysis
`_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.

```

332 \prg\_new\_conditional:Npnn \_tag\_check\_if\_mc\_tme\_missing: { T,F,TF }
333 {
334   \str\_if\_eq:eeTF {\seq\_item:Nn \l\_tag\_mc\_botmarks\_seq {1}}{b+}
335   { \prg\_return\_true: }
336   { \prg\_return\_false: }
337 }

```

(End of definition for _tag_check_if_mc_tme_missing:TF.)

338 `</package>`

339 `<*debug>`

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

```

340 \msg\_new:nnn { tag / debug } {mc-begin} { MC~begin~#1~with~options:~\tl\_to\_str:n{#2}~[\msg\_lin
341 \msg\_new:nnn { tag / debug } {mc-end} { MC~end~#1~[\msg\_line\_context:] }
342
343 \cs\_new\_protected:Npn \_tag\_debug\_mc\_begin\_insert:n #1
344 {
345   \int\_compare:nNnT { \l\_tag\_loglevel\_int } > {0}
346   {
347     \msg\_note:nnnn { tag / debug } {mc-begin} {inserted} { #1 }
348   }
349 }
350 \cs\_new\_protected:Npn \_tag\_debug\_mc\_begin\_ignore:n #1
351 {
352   \int\_compare:nNnT { \l\_tag\_loglevel\_int } > {0}
353   {
354     \msg\_note:nnnn { tag / debug } {mc-begin} {ignored} { #1 }
355   }
356 }
357 \cs\_new\_protected:Npn \_tag\_debug\_mc\_end\_insert:
358 {
359   \int\_compare:nNnT { \l\_tag\_loglevel\_int } > {0}

```

```

360     {
361         \msg_note:nnn { tag / debug } {mc-end} {inserted}
362     }
363 }
364 \cs_new_protected:Npn \__tag_debug_mc_end_ignore:
365 {
366     \int_compare:nNnT { \l__tag_loglevel_int } > {0}
367     {
368         \msg_note:nnn { tag / debug } {mc-end} {ignored}
369     }
370 }

```

And now something for the structures

```

371 \msg_new:nnn { tag / debug } {struct-begin}
372 {
373     Struct~\tag_get:n{struct_num}~begin~#1~with~options::~~\tl_to_str:n{#2}~[\msg_line_context:]
374 }
375 \msg_new:nnn { tag / debug } {struct-end}
376 {
377     Struct~end~#1~[\msg_line_context:]
378 }
379 \msg_new:nnn { tag / debug } {struct-end-wrong}
380 {
381     Struct~end~'#1'~doesn't~fit~start~'#2'~[\msg_line_context:]
382 }
383
384 \cs_new_protected:Npn \__tag_debug_struct_begin_insert:n #1
385 {
386     \int_compare:nNnT { \l__tag_loglevel_int } > {0}
387     {
388         \msg_note:nnnn { tag / debug } {struct-begin} {inserted} { #1 }
389         \seq_log:N \g__tag_struct_tag_stack_seq
390     }
391 }
392 \cs_new_protected:Npn \__tag_debug_struct_begin_ignore:n #1
393 {
394     \int_compare:nNnT { \l__tag_loglevel_int } > {0}
395     {
396         \msg_note:nnnn { tag / debug } {struct-begin} {ignored} { #1 }
397     }
398 }
399 \cs_new_protected:Npn \__tag_debug_struct_end_insert:
400 {
401     \int_compare:nNnT { \l__tag_loglevel_int } > {0}
402     {
403         \msg_note:nnn { tag / debug } {struct-end} {inserted}
404         \seq_log:N \g__tag_struct_tag_stack_seq
405     }
406 }
407 \cs_new_protected:Npn \__tag_debug_struct_end_ignore:
408 {
409     \int_compare:nNnT { \l__tag_loglevel_int } > {0}
410     {
411         \msg_note:nnn { tag / debug } {struct-end} {ignored}
412     }

```

```

413 }
414 \cs_new_protected:Npn \__tag_debug_struct_end_check:n #1
415 {
416   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
417   {
418     \seq_get:NNT \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
419     {
420       \str_if_eq:eeF
421       {#1}
422       {\exp_last_unbraced:NV\use_i:nn \l__tag_tmpa_tl}
423       {
424         \msg_warning:nnxx { tag/debug }{ struct-end-wrong }
425         {#1}
426         {\exp_last_unbraced:NV\use_i:nn \l__tag_tmpa_tl}
427       }
428     }
429   }
430 }
431
432 </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 <code>activate-mc</code> , <code>activate-tree</code> , <code>activate-struct</code> and additionally add a document structure.
------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

<code>\tag_tool:n</code>	<code>\tag_tool:n{\key val}</code>
<code>\tagtool</code>	

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

2 Commands related to mc-chunks

<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 `pdflatex` as `lualatex` doesn't mind much if a mc tag is not correctly closed. Unlike the `expl3` command it is not expandable.

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

3 Commands related to structures

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

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

4 Debugging

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

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

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

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

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

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

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

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

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

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

5 Extension commands

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

The commands and keys should be view as experimental!

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

5.1 Fake space

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

5.2 Paratagging

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

<code>paratagging_␣(setup-key)</code>	<code>paratagging = true false</code>
<code>paratagging-show_␣(setup-key)</code>	<code>paratagging-show = true false</code>

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

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

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

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

5.3 Header and footer

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

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

5.4 Link tagging

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

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

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

6 User commands and extensions of document commands

```
1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-user} {2023-06-14} {0.98i}
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 of definition for \tagpdfsetup. This function is documented on page 33.)

`\tag_tool:n`
`\tagtool`

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

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

(End of definition for \tag_tool:n and \tagtool. These functions are documented on page 33.)

8 Commands for the mc-chunks

```

\tagmcbegin
\tagmcend
\tagmcuse
22 < *base>
23 \NewDocumentCommand \tagmcbegin { m }
24 {
25   \tag_mc_begin:n {#1}
26 }
27
28
29 \NewDocumentCommand \tagmcend { }
30 {
31   \tag_mc_end:
32 }
33
34 \NewDocumentCommand \tagmcuse { m }
35 {
36   \tag_mc_use:n {#1}
37 }
38 < /base>

```

(End of definition for `\tagmcbegin`, `\tagmcend`, and `\tagmcuse`. These functions are documented on page 33.)

`\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.

```

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

```

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

9 Commands for the structure

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

```

45 < *base>
46 \NewDocumentCommand \tagstructbegin { m }
47 {
48   \tag_struct_begin:n {#1}
49 }
50
51 \NewDocumentCommand \tagstructend { }
52 {
53   \tag_struct_end:
54 }
55
56 \NewDocumentCommand \tagstructuse { m }
57 {

```

```

58     \tag_struct_use:n {#1}
59   }
60 </base>

```

(End of definition for `\tagstructbegin`, `\tagstructend`, and `\tagstructuse`. These functions are documented on page 34.)

10 Debugging

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

```

61 <*package>
62 \NewDocumentCommand\ShowTagging { m }
63 {
64   \keys_set:nn { __tag / show }{ #1}
65
66 }

```

(End of definition for `\ShowTagging`. This function is documented on page 34.)

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.

```

67 \keys_define:nn { __tag / show }
68 {
69   mc-data .code:n =
70     {
71       \sys_if_engine_luatex:T
72       {
73         \lua_now:e{!tx.__tag.trace.show_all_mc_data(#1,\__tag_get_mc_abs_cnt:,0)}
74       }
75     }
76   ,mc-data .default:n = 1
77 }
78

```

(End of definition for `mc-data (show-key)`. This function is documented on page 34.)

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

```

79 \keys_define:nn { __tag / show }
80 { mc-current .code:n =
81   {
82     \bool_if:NTF \g__tag_mode_lua_bool
83     {
84       \sys_if_engine_luatex:T
85       {
86         \int_compare:nNnTF
87         { -2147483647 }
88         =
89         {
90           \lua_now:e
91           {
92             tex.print

```

```

93         (tex.getattribute
94         (luatexbase.attributes.g__tag_mc_cnt_attr))
95     }
96 }
97 {
98     \lua_now:e
99     {
100         ltx.__tag.trace.log
101         (
102             "mc-current:~no~MC~open,~current~absent
103             =\__tag_get_mc_abs_cnt:"
104             ,0
105         )
106         texio.write_nl("")
107     }
108 }
109 {
110     \lua_now:e
111     {
112         ltx.__tag.trace.log
113         (
114             "mc-current:~absent=\__tag_get_mc_abs_cnt:=="
115             ..
116             tex.getattribute(luatexbase.attributes.g__tag_mc_cnt_attr)
117             ..
118             "~=>tag="
119             ..
120             tostring
121             (ltx.__tag.func.get_tag_from
122             (tex.getattribute
123             (luatexbase.attributes.g__tag_mc_type_attr)))
124             ..
125             "="
126             ..
127             tex.getattribute
128             (luatexbase.attributes.g__tag_mc_type_attr)
129             ,0
130         )
131         texio.write_nl("")
132     }
133 }
134 }
135 }
136 {
137     \msg_note:nn{ tag }{ mc-current }
138 }
139 }
140 }

```

(End of definition for mc-current (show-key). This function is documented on page 34.)

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).

```

141 \keys_define:nn { __tag / show }

```

```

142 {
143   mc-marks .choice: ,
144   mc-marks / show .code:n =
145   {
146     \__tag_mc_get_marks:
147     \__tag_check_if_mc_in_galley:TF
148     {
149       \iow_term:n {Marks~from~this~page:~}
150     }
151     {
152       \iow_term:n {Marks~from~a~previous~page:~}
153     }
154     \seq_show:N \l__tag_mc_firstmarks_seq
155     \seq_show:N \l__tag_mc_botmarks_seq
156     \__tag_check_if_mc_tmb_missing:T
157     {
158       \iow_term:n {BDC~missing~on~this~page!}
159     }
160     \__tag_check_if_mc_tme_missing:T
161     {
162       \iow_term:n {EMC~missing~on~this~page!}
163     }
164   },
165   mc-marks / use .code:n =
166   {
167     \__tag_mc_get_marks:
168     \__tag_check_if_mc_in_galley:TF
169     { Marks~from~this~page:~}
170     { Marks~from~a~previous~page:~}
171     \seq_use:Nn \l__tag_mc_firstmarks_seq {,~}\quad
172     \seq_use:Nn \l__tag_mc_botmarks_seq {,~}\quad
173     \__tag_check_if_mc_tmb_missing:T
174     {
175       BDC~missing~
176     }
177     \__tag_check_if_mc_tme_missing:T
178     {
179       EMC~missing
180     }
181   },
182   mc-marks .default:n = show
183 }

```

(End of definition for mc-marks (show-key). This function is documented on page 34.)

struct-stack_⌊(show-key)

```

184 \keys_define:nn { __tag / show }
185 {
186   struct-stack .choice:
187   ,struct-stack / log .code:n = \seq_log:N \g__tag_struct_tag_stack_seq
188   ,struct-stack / show .code:n = \seq_show:N \g__tag_struct_tag_stack_seq
189   ,struct-stack .default:n = show
190 }

```

(End of definition for struct-stack (show-key). This function is documented on page 34.)

11 Commands to extend document 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. This part should be regularly revisited to check if the code should go to a better place or can be improved.

11.1 new ref system

Until l3ref is in the kernel, we provide a definition for `\newlabeldata` in the aux-file to avoid errors if a document switches between tagging and non-tagging.

```

191 </package>
192 <base>\AddToHook{begindocument}
193 <base> {\immediate\write\@mainaux{\string\providecommand\string\newlabeldata[2]{}}}
194 <*package>

```

11.2 Document structure

```

\g__tag_root_default_tl
activate_␣(setup-key)
195 \tl_new:N\g__tag_root_default_tl
196 \tl_gset:Nn\g__tag_root_default_tl {Document}
197
198 \hook_gput_code:nnn{begindocument}{tagpdf}{\tagstructbegin{tag=\g__tag_root_default_tl}}
199 \hook_gput_code:nnn{tagpdf/finish/before}{tagpdf}{\tagstructend}
200
201 \keys_define:nn { __tag / setup }
202 {
203   activate .code:n =
204   {
205     \keys_set:nn { __tag / setup }
206     { activate-mc,activate-tree,activate-struct }
207     \tl_gset:Nn\g__tag_root_default_tl {#1}
208   },
209   activate .default:n = Document
210 }
211

```

(End of definition for `\g__tag_root_default_tl` and `activate (setup-key)`. This function is documented on page 33.)

11.3 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: structures should be activated, the code in the pdfmanagement must be there. Structure destinations are actually PDF 2.0 only but they don't harm in older PDF and can improve html export.

```

212 \AddToHook{begindocument/before}
213 {
214   \bool_lazy_all:nT
215   {
216     { \g__tag_active_struct_dest_bool }

```

```

217         { \g__tag_active_struct_bool }
218         { \cs_if_exist_p:N \pdf_activate_structure_destination: }
219     }
220     {
221         \tl_set:Nn \l_pdf_current_structure_destination_tl { __tag/struct/\g__tag_struct_stack
222         \pdf_activate_structure_destination:
223     }
224 }

```

11.4 Fake space

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

```

225 \sys_if_engine_luatex:T
226 {
227     \NewDocumentCommand\pdffakespace { }
228     {
229         \__tag_fakespace:
230     }
231 }
232 \providecommand\pdffakespace{}

```

(End of definition for \pdffakespace. This function is documented on page 35.)

11.5 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
\l__tag_para_flattened_bool
\l__tag_para_show_bool
\g__tag_para_begin_int
\g__tag_para_end_int
\g__tag_para_main_begin_int
\g__tag_para_main_end_int
\l__tag_para_tag_default_tl
\l__tag_para_tag_tl
\l__tag_para_main_tag_tl
233 \bool_new:N \l__tag_para_bool
234 \</package>
235 \<base>\bool_new:N \l__tag_para_flattened_bool
236 \<*package>
237 \bool_new:N \l__tag_para_show_bool
238 \int_new:N \g__tag_para_begin_int
239 \int_new:N \g__tag_para_end_int
240 \int_new:N \g__tag_para_main_begin_int
241 \int_new:N \g__tag_para_main_end_int
242 \tl_new:N \l__tag_para_tag_default_tl
243 \tl_set:Nn \l__tag_para_tag_default_tl { text }
244 \tl_new:N \l__tag_para_tag_tl
245 \tl_set:Nn \l__tag_para_tag_tl { \l__tag_para_tag_default_tl }
246 \tl_new:N \l__tag_para_main_tag_tl
247 \tl_set:Nn \l__tag_para_main_tag_tl {text-unit}

```

(End of definition for \l__tag_para_bool and others.)

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. The paratag key sets the tag used by the next automatic paratagging, it can also be changed with \tag_tool:n

paratagging-show_␣(setup-key)

paratag_␣(setup-key)

paratag_␣(tool-key)

unittag_␣(tool-key)

para-flattened_␣(tool-key)

```

248 \keys_define:nn { __tag / setup }

```

```

249 {
250   paratagging .bool_set:N = \l__tag_para_bool,
251   paratagging-show .bool_set:N = \l__tag_para_show_bool,
252   paratag .tl_set:N = \l__tag_para_tag_tl
253 }
254 \keys_define:nn { tag / tool}
255 {
256   paratag .tl_set:N = \l__tag_para_tag_tl,
257   unittag .tl_set:N = \l__tag_para_main_tag_tl,
258   para-flattened .bool_set:N = \l__tag_para_flattened_bool
259 }

```

(End of definition for paratagging (setup-key) and others. These functions are documented on page 35.)

This fills the para hooks with the needed code.

```

260 \cs_new_protected:Npn \__tag_check_para_begin_show:nn #1 #2
261   % #1 color, #2 prefix
262   {
263     \bool_if:NT \l__tag_para_show_bool
264     {
265       \tag_mc_begin:n{artifact}
266       \llap{\color_select:n{#1}\tiny#2\int_use:N\g__tag_para_begin_int\ }
267       \tag_mc_end:
268     }
269   }
270
271 \cs_new_protected:Npn \__tag_check_para_end_show:nn #1 #2
272   % #1 color, #2 prefix
273   {
274     \bool_if:NT \l__tag_para_show_bool
275     {
276       \tag_mc_begin:n{artifact}
277       \rlap{\color_select:n{#1}\tiny\ #2\int_use:N\g__tag_para_end_int}
278       \tag_mc_end:
279     }
280   }
281
282 \AddToHook{para/begin}
283 {
284   \bool_if:NT \l__tag_para_bool
285   {
286     \bool_if:NF \l__tag_para_flattened_bool
287     {
288       \int_gincr:N \g__tag_para_main_begin_int
289       \tag_struct_begin:n
290       {
291         tag=\l__tag_para_main_tag_tl,
292       }
293     }
294     \int_gincr:N \g__tag_para_begin_int
295     \tag_struct_begin:n {tag=\l__tag_para_tag_tl}
296     \__tag_check_para_begin_show:nn {green}{}
297     \tag_mc_begin:n {}
298   }

```

```

299   }
300   \AddToHook{para/end}
301   {
302     \bool_if:NT \l__tag_para_bool
303     {
304       \int_gincr:N \g__tag_para_end_int
305       \tag_mc_end:
306       \__tag_check_para_end_show:nn {red}{}
307       \tag_struct_end:
308       \bool_if:NF \l__tag_para_flattened_bool
309       {
310         \int_gincr:N \g__tag_para_main_end_int
311         \tag_struct_end:
312       }
313     }
314   }

```

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

```

315   \AddToHook{enddocument/info}
316   {
317     \tag_if_active:F
318     {
319       \msg_redirect_name:nnn { tag } { para-hook-count-wrong } { warning }
320     }
321     \int_compare:nNnF {\g__tag_para_main_begin_int}={\g__tag_para_main_end_int}
322     {
323       \msg_error:nnxxx
324       {tag}
325       {para-hook-count-wrong}
326       {\int_use:N\g__tag_para_main_begin_int}
327       {\int_use:N\g__tag_para_main_end_int}
328       {text-unit}
329     }
330     \int_compare:nNnF {\g__tag_para_begin_int}={\g__tag_para_end_int}
331     {
332       \msg_error:nnxxx
333       {tag}
334       {para-hook-count-wrong}
335       {\int_use:N\g__tag_para_begin_int}
336       {\int_use:N\g__tag_para_end_int}
337       {text}
338     }
339   }

```

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

```

340   \AddToHook{begindocument/before}
341   {
342     \@ifundefined{@mult@ptagging@hook}{\RequirePackage{output-patches-tmp-ltx}}{} %
343     \bool_if:NF \g__tag_mode_lua_bool
344     {
345       \cs_if_exist:NT \@kernel@before@footins
346       {
347         \tl_put_right:Nn \@kernel@before@footins

```

```

348         { \_tag_add_missing_mcs_to_stream:Nn \footins {footnote} }
349     \tl_put_right:Nn \@kernel@before@cclv
350     {
351         \_tag_check_typeout_v:n {====>~In~\token_to_str:N \@makecol\c_space_tl\the\c@
352         \_tag_add_missing_mcs_to_stream:Nn \@cclv {main}
353     }
354     \tl_put_right:Nn \@mult@ptagging@hook
355     {
356         \_tag_check_typeout_v:n {====>~In~\string\page@sofar}
357         \process@cols\mult@firstbox
358         {
359             \_tag_add_missing_mcs_to_stream:Nn \count@ {multicol}
360         }
361         \_tag_add_missing_mcs_to_stream:Nn \mult@rightbox {multicol}
362     }
363 }
364 }
365 }
366 \end{package}

```

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

\tagpdfparaOff

```

367 \base\newcommand\tagpdfparaOn {}
368 \base\newcommand\tagpdfparaOff{}
369 *package)
370 \renewcommand\tagpdfparaOn {\bool_set_true:N \l__tag_para_bool}
371 \renewcommand\tagpdfparaOff{\bool_set_false:N \l__tag_para_bool}
372 \keys_define:nn { tag / tool}
373 {
374     para .bool_set:N = \l__tag_para_bool,
375     para-flattened .bool_set:N = \l__tag_para_flattened_bool,
376 }

```

(End of definition for \tagpdfparaOn and \tagpdfparaOff. These functions are documented on page 35.)

\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}%

377 \NewDocumentCommand\tagpdfsuppressmarks{m}
378 {{\use:c{\_tag_mc_disable_marks:} #1}}

```

(End of definition for \tagpdfsuppressmarks. This function is documented on page 35.)

11.6 Header and footer

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

```

379 \cs_new_protected:Npn\__tag_hook_kernel_before_head:{}
380 \cs_new_protected:Npn\__tag_hook_kernel_after_head:{}
381 \cs_new_protected:Npn\__tag_hook_kernel_before_foot:{}
382 \cs_new_protected:Npn\__tag_hook_kernel_after_foot:{}
383
384 \AddToHook{begindocument}
385 {
386   \cs_if_exist:NT \@kernel@before@head
387   {
388     \tl_put_right:Nn \@kernel@before@head {\__tag_hook_kernel_before_head:}
389     \tl_put_left:Nn \@kernel@after@head {\__tag_hook_kernel_after_head:}
390     \tl_put_right:Nn \@kernel@before@foot {\__tag_hook_kernel_before_foot:}
391     \tl_put_left:Nn \@kernel@after@foot {\__tag_hook_kernel_after_foot:}
392   }
393 }
394
395 \bool_new:N \g__tag_saved_in_mc_bool
396 \cs_new_protected:Npn \__tag_exclude_headfoot_begin:
397 {
398   \bool_set_false:N \l__tag_para_bool
399   \bool_if:NTF \g__tag_mode_lua_bool
400   {
401     \tag_mc_end_push:
402   }
403   {
404     \bool_gset_eq:NN \g__tag_saved_in_mc_bool \g__tag_in_mc_bool
405     \bool_gset_false:N \g__tag_in_mc_bool
406   }
407   \tag_mc_begin:n {artifact}
408   \tag_stop:n{headfoot}
409 }
410 \cs_new_protected:Npn \__tag_exclude_headfoot_end:
411 {
412   \tag_start:n{headfoot}
413   \tag_mc_end:
414   \bool_if:NTF \g__tag_mode_lua_bool
415   {
416     \tag_mc_begin_pop:n{}
417   }
418   {
419     \bool_gset_eq:NN \g__tag_in_mc_bool\g__tag_saved_in_mc_bool
420   }
421 }

```

This version allows to use an Artifact structure

```

422 \__tag_attr_new_entry:nn {\__tag/attr/pagination}{/0/Artifact/Type/Pagination}
423 \cs_new_protected:Npn \__tag_exclude_struct_headfoot_begin:n #1
424 {
425   \bool_set_false:N \l__tag_para_bool

```

```

426     \bool_if:NTF \g__tag_mode_lua_bool
427     {
428         \tag_mc_end_push:
429     }
430     {
431         \bool_gset_eq:NN \g__tag_saved_in_mc_bool \g__tag_in_mc_bool
432         \bool_gset_false:N \g__tag_in_mc_bool
433     }
434     \tag_struct_begin:n{tag=Artifact,attribute-class=__tag/attr/#1}
435     \tag_mc_begin:n {artifact=#1}
436     \tag_stop:n{headfoot}
437 }
438
439 \cs_new_protected:Npn \__tag_exclude_struct_headfoot_end:
440 {
441     \tag_start:n{headfoot}
442     \tag_mc_end:
443     \tag_struct_end:
444     \bool_if:NTF \g__tag_mode_lua_bool
445     {
446         \tag_mc_begin_pop:n{ }
447     }
448     {
449         \bool_gset_eq:NN \g__tag_in_mc_bool \g__tag_saved_in_mc_bool
450     }
451 }

```

And now the keys

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

```

452 \keys_define:nn { __tag / setup }
453 {
454     exclude-header-footer .choice:,
455     exclude-header-footer / true .code:n =
456     {
457         \cs_set_eq:NN \__tag_hook_kernel_before_head: \__tag_exclude_headfoot_begin:
458         \cs_set_eq:NN \__tag_hook_kernel_before_foot: \__tag_exclude_headfoot_begin:
459         \cs_set_eq:NN \__tag_hook_kernel_after_head: \__tag_exclude_headfoot_end:
460         \cs_set_eq:NN \__tag_hook_kernel_after_foot: \__tag_exclude_headfoot_end:
461     },
462     exclude-header-footer / pagination .code:n =
463     {
464         \cs_set:Nn \__tag_hook_kernel_before_head: { \__tag_exclude_struct_headfoot_begin:n {p
465         \cs_set:Nn \__tag_hook_kernel_before_foot: { \__tag_exclude_struct_headfoot_begin:n {p
466         \cs_set_eq:NN \__tag_hook_kernel_after_head: \__tag_exclude_struct_headfoot_end:
467         \cs_set_eq:NN \__tag_hook_kernel_after_foot: \__tag_exclude_struct_headfoot_end:
468     },
469     exclude-header-footer / false .code:n =
470     {
471         \cs_set_eq:NN \__tag_hook_kernel_before_head: \prg_do_nothing:
472         \cs_set_eq:NN \__tag_hook_kernel_before_foot: \prg_do_nothing:
473         \cs_set_eq:NN \__tag_hook_kernel_after_head: \prg_do_nothing:
474         \cs_set_eq:NN \__tag_hook_kernel_after_foot: \prg_do_nothing:
475     },
476     exclude-header-footer .default:n = true,

```

```

477     exclude-header-footer .initial:n = true
478 }

```

(End of definition for exclude-header-footer (setup-key). This function is documented on page 35.)

11.7 Links

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

```

479 \hook_gput_code:nnn
480 {pdfannot/link/URI/before}
481 {tagpdf}
482 {
483     \tag_mc_end_push:
484     \tag_struct_begin:n { tag=Link }
485     \tag_mc_begin:n { tag=Link }
486     \pdfannot_dict_put:nnx
487     { link/URI }
488     { StructParent }
489     { \tag_struct_parent_int: }
490 }
491
492 \hook_gput_code:nnn
493 {pdfannot/link/URI/after}
494 {tagpdf}
495 {
496     \tag_struct_insert_annot:xx {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
497     \tag_mc_end:
498     \tag_struct_end:
499     \tag_mc_begin_pop:n{ }
500 }
501
502 \hook_gput_code:nnn
503 {pdfannot/link/GoTo/before}
504 {tagpdf}
505 {
506     \tag_mc_end_push:
507     \tag_struct_begin:n{tag=Link}
508     \tag_mc_begin:n{tag=Link}
509     \pdfannot_dict_put:nnx
510     { link/GoTo }
511     { StructParent }
512     { \tag_struct_parent_int: }
513 }
514
515 \hook_gput_code:nnn
516 {pdfannot/link/GoTo/after}
517 {tagpdf}
518 {
519     \tag_struct_insert_annot:xx {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
520     \tag_mc_end:
521     \tag_struct_end:
522     \tag_mc_begin_pop:n{ }

```



```

523
524   }
525
526   % "alternative descriptions " for PAX3. How to get better text here??
527   \pdfannot_dict_put:nnn
528   { link/URI }
529   { Contents }
530   { (url) }
531
532   \pdfannot_dict_put:nnn
533   { link/GoTo }
534   { Contents }
535   { (ref) }
536
537   </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} {2023-06-14} {0.98i}
4 {part of tagpdf - code related to writing trees and dictionaries to the pdf}
5 </header>
```

1 Trees, pdfmanagement and finalization code

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

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

1.1 Check structure

__tag_tree_final_checks:

```
20 \cs_new_protected:Npn \__tag_tree_final_checks:
21 {
22   \int_compare:nNnF {\seq_count:N\g__tag_struct_stack_seq}={1}
23   {
24     \msg_warning:nn {tag}{tree-struct-still-open}
25     \int_step_inline:nnn{2}{\seq_count:N\g__tag_struct_stack_seq}
26     {\tag_struct_end:}
27   }
28 }
```

(End of definition for __tag_tree_final_checks:.)

1.2 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
29 \pdf_object_new:n { __tag/struct/0 }
(End of definition for __tag/struct/0.)
30 \hook_gput_code:nnn{shipout/lastpage}{tagpdf}
31 {
32   \bool_if:NT \g__tag_active_tree_bool
33   {
34     \pdfmanagement_add:nnn { Catalog / MarkInfo } { Marked } { true }
35     \pdfmanagement_add:nnx
36       { Catalog }
37       { StructTreeRoot }
38     { \pdf_object_ref:n { __tag/struct/0 } }
39   }
40 }

```

1.3 Writing the IDtree

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

```

\g__tag_tree_id_pad_int
41 \int_new:N\g__tag_tree_id_pad_int
(End of definition for \g__tag_tree_id_pad_int.)
Now we get the needed padding
42 \cs_generate_variant:Nn \tl_count:n {e}
43 \hook_gput_code:nnn{begindocument}{tagpdf}
44 {
45   \int_gset:Nn\g__tag_tree_id_pad_int
46   {\tl_count:e { \__tag_ref_value_lastpage:nn{tagstruct}{1000}}+1}
47 }
48

```

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

```

49 \cs_new_protected:Npn \__tag_tree_write_idtree:
50 {
51   \tl_clear:N \l__tag_tmpa_tl
52   \tl_clear:N \l__tag_tmpb_tl
53   \int_zero:N \l__tag_tmpa_int
54   \int_step_inline:nn {\c@g__tag_struct_abs_int}
55   {
56     \int_incr:N\l__tag_tmpa_int
57     \tl_put_right:Nx \l__tag_tmpa_tl
58     {
59       \__tag_struct_get_id:n{##1}~\pdf_object_ref:n{__tag/struct/##1}~

```

```

60     }
61     \int_compare:nNnF {\l__tag_tmpa_int}<{50} %
62     {
63         \pdf_object_unnamed_write:nx {dict}
64         { /Limits~[\__tag_struct_get_id:n{##1-\l__tag_tmpa_int+1}~\__tag_struct_get_id:n{##2-\l__tag_tmpa_int+1}~
65           /Names~[\l__tag_tmpa_tl]
66         }
67         \tl_put_right:Nx\l__tag_tmpb_tl {\pdf_object_ref_last:\c_space_tl}
68         \int_zero:N \l__tag_tmpa_int
69         \tl_clear:N \l__tag_tmpa_tl
70     }
71 }
72 \tl_if_empty:NF \l__tag_tmpa_tl
73 {
74     \pdf_object_unnamed_write:nx {dict}
75     {
76         /Limits~
77         [\__tag_struct_get_id:n{\c@g__tag_struct_abs_int-\l__tag_tmpa_int+1}~
78         \__tag_struct_get_id:n{\c@g__tag_struct_abs_int}]
79         /Names~[\l__tag_tmpa_tl]
80     }
81     \tl_put_right:Nx\l__tag_tmpb_tl {\pdf_object_ref_last:}
82 }
83 \pdf_object_unnamed_write:nx {dict}{/Kids~[\l__tag_tmpb_tl]}
84 \__tag_prop_gput:cnx
85   { g__tag_struct_0_prop }
86   { IDTree }
87   { \pdf_object_ref_last: }
88 }

```

1.4 Writing structure elements

The following commands are needed to write out the structure.

```

\__tag_tree_write_structtreeroot: This writes out the root object.
89 \pdf_version_compare:NnTF < {2.0}
90 {
91     \cs_new_protected:Npn \__tag_tree_write_structtreeroot:
92     {
93         \__tag_prop_gput:cnx
94         { g__tag_struct_0_prop }
95         { ParentTree }
96         { \pdf_object_ref:n { __tag/tree/parenttree } }
97         \__tag_prop_gput:cnx
98         { g__tag_struct_0_prop }
99         { RoleMap }
100        { \pdf_object_ref:n { __tag/tree/rolemap } }
101        \__tag_struct_fill_kid_key:n { 0 }
102        \__tag_struct_get_dict_content:nN { 0 } \l__tag_tmpa_tl
103        \pdf_object_write:nnx
104        { __tag/struct/0 }
105        {dict}
106        {
107            \l__tag_tmpa_tl

```

```

108         }
109     }
110 }
no RoleMap in pdf 2.0
111 {
112     \cs_new_protected:Npn \__tag_tree_write_structtreeroot:
113     {
114         \__tag_prop_gput:cnx
115         { g__tag_struct_0_prop }
116         { ParentTree }
117         { \pdf_object_ref:n { __tag/tree/parenttree } }
118         \__tag_struct_fill_kid_key:n { 0 }
119         \__tag_struct_get_dict_content:nN { 0 } \l__tag_tmpa_tl
120         \pdf_object_write:nnx
121         { __tag/struct/0 }
122         {dict}
123         {
124             \l__tag_tmpa_tl
125         }
126     }
127 }

```

(End of definition for __tag_tree_write_structtreeroot:.)

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

```

128 \cs_new_protected:Npn \__tag_tree_write_structelements:
129 {
130     \int_step_inline:nnnn {1}{1}{\c@g__tag_struct_abs_int}
131     {
132         \__tag_struct_write_obj:n { ##1 }
133     }
134 }

```

(End of definition for __tag_tree_write_structelements:.)

1.5 ParentTree

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

```

135 \pdf_object_new:n { __tag/tree/parenttree }

```

(End of definition for __tag/tree/parenttree.)

The ParentTree maps numbers to objects or (if the number represents a page) to arrays of objects. The numbers refer to two distinct types of entries: page streams and real objects like annotations. The numbers must be distinct and ordered. So we rely on abspage for the pages and put the real objects at the end. We use a counter to have a chance to get the correct number if code is processed twice.

\c@g__tag_parenttree_obj_int This is a counter for the real objects. It starts at the absolute last page value. It relies on l3ref.

```

136 \newcounter { g__tag_parenttree_obj_int }
137 \hook_gput_code:nnn{begindocument}{tagpdf}
138 {
139     \int_gset:Nn

```

```

140      \c@g__tag_parenttree_obj_int
141      { \_tag_ref_value_lastpage:nn{abspage}{100} }
142    }

```

(End of definition for \c@g__tag_parenttree_obj_int.)

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

\g__tag_parenttree_objr_tl

```

143 \tl_new:N \g__tag_parenttree_objr_tl
(End of definition for \g__tag_parenttree_objr_tl.)

```

_tag_parenttree_add_objr:nn

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

```

144 \cs_new_protected:Npn \_tag_parenttree_add_objr:nn #1 #2 %1 StructParent number, #2 objref
145 {
146   \tl_gput_right:Nx \g__tag_parenttree_objr_tl
147   {
148     #1 \c_space_tl #2 ^^J
149   }
150 }

```

(End of definition for _tag_parenttree_add_objr:nn.)

\l__tag_parenttree_content_tl

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

```

151 \tl_new:N \l__tag_parenttree_content_tl
(End of definition for \l__tag_parenttree_content_tl.)

```

_tag_tree_fill_parenttree:

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

```

152 \cs_new_protected:Npn \_tag_tree_parenttree_rerun_msg: {}
153 \cs_new_protected:Npn \_tag_tree_fill_parenttree:
154 {
155   \int_step_inline:nnnn{1}{1}{\_tag_ref_value_lastpage:nn{abspage}{-1}} %not quite clear i
156   { %page ##1
157     \prop_clear:N \l__tag_tmpa_prop
158     \int_step_inline:nnnn{1}{1}{\_tag_ref_value_lastpage:nn{tagmcabs}{-1}}
159     {
160       %mcid###1
161       \int_compare:nT
162         {\_tag_ref_value:enn{mcid-###1}{tagabspage}{-1}=##1} %mcid is on current page
163       {% yes
164         \prop_put:Nxx
165         \l__tag_tmpa_prop
166         {\_tag_ref_value:enn{mcid-###1}{tagmcid}{-1}}
167         {\prop_item:Nn \g__tag_mc_parenttree_prop {###1}}
168       }
169     }
170     \tl_put_right:Nx \l__tag_parenttree_content_tl
171     {
172       \int_eval:n {##1-1} \c_space_tl
173       [\c_space_tl %]
174     }

```

```

175 \int_step_inline:nnnn
176 {0}
177 {1}
178 { \prop_count:N \l__tag_tmpa_prop -1 }
179 {
180   \prop_get:NnNTF \l__tag_tmpa_prop {###1} \l__tag_tmpa_tl
181   {% page#1:mcid##1:\l__tag_tmpa_tl :content
182     \tl_put_right:Nx \l__tag_parenttree_content_tl
183     {
184       \pdf_object_if_exist:eT { __tag/struct/\l__tag_tmpa_tl }
185       {
186         \pdf_object_ref:e { __tag/struct/\l__tag_tmpa_tl }
187       }
188       \c_space_tl
189     }
190   }
191   {
192     \cs_set_protected:Npn \__tag_tree_parenttree_rerun_msg:
193     {
194       \msg_warning:nn { tag } {tree-mcid-index-wrong}
195     }
196   }
197 }
198 \tl_put_right:Nn
199 \l__tag_parenttree_content_tl
200 {%[
201   ]^^J
202 }
203 }
204 }

```

(End of definition for __tag_tree_fill_parenttree:.)

__tag_tree_lua_fill_parenttree: This is a special variant for luatex. lua mode must/can do it differently.

```

205 \cs_new_protected:Npn \__tag_tree_lua_fill_parenttree:
206 {
207   \tl_set:Nn \l__tag_parenttree_content_tl
208   {
209     \lua_now:e
210     {
211       ltx.__tag.func.output_parenttree
212       (
213         \int_use:N\g_shipout_readonly_int
214       )
215     }
216   }
217 }

```

(End of definition for __tag_tree_lua_fill_parenttree:.)

__tag_tree_write_parenttree: This combines the two parts and writes out the object. TODO should the check for lua be moved into the backend code?

```

218 \cs_new_protected:Npn \__tag_tree_write_parenttree:
219 {

```

```

220     \bool_if:NTF \g__tag_mode_lua_bool
221     {
222       \__tag_tree_lua_fill_parenttree:
223     }
224     {
225       \__tag_tree_fill_parenttree:
226     }
227     \__tag_tree_parenttree_rerun_msg:
228     \tl_put_right:NV \l__tag_parenttree_content_tl\g__tag_parenttree_objr_tl
229     \pdf_object_write:nmx { \__tag/tree/parenttree }{dict}
230     {
231       /Nums\c_space_tl [\l__tag_parenttree_content_tl]
232     }
233   }

```

(End of definition for __tag_tree_write_parenttree:.)

1.6 Rolemap dictionary

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

```

__tag/tree/rolemap At first we reserve again an object.
234 \pdf_version_compare:NnT < {2.0}
235 {
236   \pdf_object_new:n { __tag/tree/rolemap }
237 }

```

(End of definition for __tag/tree/rolemap.)

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

```

238 \pdf_version_compare:NnTF < {2.0}
239 {
240   \cs_new_protected:Npn \__tag_tree_write_rolemap:
241   {
242     \prop_map_inline:Nn\g__tag_role_rolemap_prop
243     {
244       \tl_if_eq:nnF {##1}{##2}
245       {
246         \pdfdict_gput:nmx {g__tag_role/RoleMap_dict}
247         {##1}
248         {\pdf_name_from_unicode_e:n{##2}}
249       }
250     }
251     \pdf_object_write:nmx { __tag/tree/rolemap }{dict}
252     {
253       \pdfdict_use:n{g__tag_role/RoleMap_dict}
254     }
255   }
256 }
257 {
258   \cs_new_protected:Npn \__tag_tree_write_rolemap: {}

```



```

259     }
260
261 (End of definition for \__tag_tree_write_rolemap:.)

```

1.7 Classmap dictionary

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

```

\__tag_tree_write_classmap:
261 \cs_new_protected:Npn \__tag_tree_write_classmap:
262 {
263   \tl_clear:N \l__tag_tmpa_tl
264   \seq_gremove_duplicates:N \g__tag_attr_class_used_seq
265   \seq_set_map:NNn \l__tag_tmpa_seq \g__tag_attr_class_used_seq
266   {
267     ##1\c_space_tl
268     <<
269     \prop_item:Nn
270     \g__tag_attr_entries_prop
271     {##1}
272     >>
273   }
274   \tl_set:Nx \l__tag_tmpa_tl
275   {
276     \seq_use:Nn
277     \l__tag_tmpa_seq
278     { \iow_newline: }
279   }
280   \tl_if_empty:NF
281   \l__tag_tmpa_tl
282   {
283     \pdf_object_new:n { __tag/tree/classmap }
284     \pdf_object_write:nxx
285     { __tag/tree/classmap }
286     {dict}
287     { \l__tag_tmpa_tl }
288     \__tag_prop_gput:cnx
289     { g__tag_struct_0_prop }
290     { ClassMap }
291     { \pdf_object_ref:n { __tag/tree/classmap } }
292   }
293 }
294
295 (End of definition for \__tag_tree_write_classmap:.)

```

1.8 Namespaces

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

```

__tag/tree/namespaces
294 \pdf_object_new:n { __tag/tree/namespaces }

```

(End of definition for `__tag/tree/namespaces.`)

`__tag_tree_write_namespaces:`

```
295 \cs_new_protected:Npn \__tag_tree_write_namespaces:
296 {
297   \pdf_version_compare:NnF < {2.0}
298   {
299     \prop_map_inline:Nn \g__tag_role_NS_prop
300     {
301       \pdfdict_if_empty:nF {g__tag_role/RoleMapNS_##1_dict}
302       {
303         \pdf_object_write:nx {__tag/RoleMapNS/##1}{dict}
304         {
305           \pdfdict_use:n {g__tag_role/RoleMapNS_##1_dict}
306         }
307         \pdfdict_gput:nx{g__tag_role/namespace_##1_dict}
308         {RoleMapNS}{\pdf_object_ref:n {__tag/RoleMapNS/##1}}
309       }
310       \pdf_object_write:nx{tag/NS/##1}{dict}
311       {
312         \pdfdict_use:n {g__tag_role/namespace_##1_dict}
313       }
314     }
315     \pdf_object_write:nx {__tag/tree/namespaces}{array}
316     {
317       \prop_map_tokens:Nn \g__tag_role_NS_prop{\use_ii:nn}
318     }
319   }
320 }
```

(End of definition for `__tag_tree_write_namespaces:.`)

1.9 Finishing the structure

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

`__tag_finish_structure:`

```
321 \hook_new:n {tagpdf/finish/before}
322 \cs_new_protected:Npn \__tag_finish_structure:
323 {
324   \bool_if:NT\g__tag_active_tree_bool
325   {
326     \hook_use:n {tagpdf/finish/before}
327     \__tag_tree_final_checks:
328     \__tag_tree_write_parenttree:
329     \__tag_tree_write_idtree:
330     \__tag_tree_write_rolemap:
331     \__tag_tree_write_classmap:
332     \__tag_tree_write_namespaces:
333     \__tag_tree_write_structelements: %this is rather slow!!
334     \__tag_tree_write_structtreeroot:
335   }
336 }
```

(End of definition for `_tag_finish_structure:`.)

1.10 StructParents entry for Page

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

```
337 \hook_gput_code:nnn{begindocument}{tagpdf}  
338 {  
339   \bool_if:NT\g__tag_active_tree_bool  
340   {  
341     \hook_gput_code:nnn{shipout/before} { tagpdf/structparents }  
342     {  
343       \pdfmanagement_add:nnx  
344       { Page }  
345       { StructParents }  
346       { \int_eval:n { \g_shipout_readonly_int } }  
347     }  
348   }  
349 }  
350 </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.

<code>\tag_mc_reset_box:N</code> *	<code>\tag_mc_reset:N</code> $\{ \langle box \rangle \}$
------------------------------------	----------------------------------------------------------

New: 2023-06-11

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

2 Public keys

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

<code>tag_␣(mc-key)</code>

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).

<code>artifact_␣(mc-key)</code>

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

<code>raw_␣(mc-key)</code>

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

<code>alt_␣(mc-key)</code>

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

<code>actualtext_␣(mc-key)</code>

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

<code>label_␣(mc-key)</code>

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

<code>stash_␣(mc-key)</code>

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} {2023-06-14} {0.98i}
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 <*base>
8 \newcounter { g__tag_MCID_abs_int }

```

(End of definition for `g__tag_MCID_abs_int`.)

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

```

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

```

(End of definition for `__tag_get_data_mc_counter:`.)

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

```

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

```

(End of definition for `__tag_get_mc_abs_cnt:`.)

`\g__tag_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.

```

16 \int_new:N \g__tag_MCID_tmp_bypage_int

```

(End of definition for `\g__tag_MCID_tmp_bypage_int`.)

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

```

17 \bool_new:N \g__tag_in_mc_bool

```

(End of definition for `\g__tag_in_mc_bool`.)

`\g__tag_mc_parenttree_prop` For every chunk we need to know the structure it is in, to record this in the parent tree. We store this in a property.
key: absolute number of the mc (tagmcabs)
value: the structure number the mc is in
¹⁸ `__tag_prop_new:N \g__tag_mc_parenttree_prop`
(End of definition for \g__tag_mc_parenttree_prop.)

`\g__tag_mc_parenttree_prop` Some commands (e.g. links) want to close a previous mc and reopen it after they did their work. For this we create a stack:
¹⁹ `\seq_new:N \g__tag_mc_stack_seq`
(End of definition for \g__tag_mc_parenttree_prop.)

`\l__tag_mc_artifact_type_tl` Artifacts can have various types like Pagination or Layout. This stored in this variable.
²⁰ `\tl_new:N \l__tag_mc_artifact_type_tl`
(End of definition for \l__tag_mc_artifact_type_tl.)

`\l__tag_mc_key_stash_bool` This booleans store the stash and artifact status of the mc-chunk.
`\l__tag_mc_artifact_bool` ²¹ `\bool_new:N \l__tag_mc_key_stash_bool`
²² `\bool_new:N \l__tag_mc_artifact_bool`
(End of definition for \l__tag_mc_key_stash_bool and \l__tag_mc_artifact_bool.)

`\l__tag_mc_key_tag_tl` Variables used by the keys. `\l__@@_mc_key_properties_tl` will collect a number of values. TODO: should this be a pdfdict now?
`\g__tag_mc_key_tag_tl` ²³ `\tl_new:N \l__tag_mc_key_tag_tl`
`\l__tag_mc_key_label_tl` ²⁴ `\tl_new:N \g__tag_mc_key_tag_tl`
`\l__tag_mc_key_properties_tl` ²⁵ `\tl_new:N \l__tag_mc_key_label_tl`
²⁶ `\tl_new:N \l__tag_mc_key_properties_tl`
(End of definition for \l__tag_mc_key_tag_tl and others.)

3.2 Functions

`__tag_mc_handle_mc_label: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
tagabspage: 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.
²⁷ `\cs_new:Nn __tag_mc_handle_mc_label:n`
²⁸ `{`
²⁹ `__tag_ref_label:en{tagpdf-#1}{mc}`
³⁰ `}`
(End of 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

```

31 \cs_new_protected:Npn \__tag_mc_set_label_used:n #1 %#1 labelname
32 {
33   \tl_new:c { g__tag_mc_label_\tl_to_str:n{#1}_used_tl }
34 }
35 </shared>

```

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

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

TODO: is testing for struct the right test?

```

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

```

(End of definition for `\tag_mc_use:n`. This function is documented on page 60.)

`\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:`

```

60 <base>\cs_new_protected:Npn \tag_mc_artifact_group_begin:n #1 {}
61 <base>\cs_new_protected:Npn \tag_mc_artifact_group_end: {}
62 <*shared>
63 \cs_set_protected:Npn \tag_mc_artifact_group_begin:n #1
64 {
65   \tag_mc_end_push:
66   \tag_mc_begin:n {artifact=#1}
67   \tag_stop_group_begin:
68 }
69

```



```

70 \cs_set_protected:Npn \tag_mc_artifact_group_end:
71 {
72   \tag_stop_group_end:
73   \tag_mc_end:
74   \tag_mc_begin_pop:n{ }
75 }
76 </shared>

```

(End of definition for \tag_mc_artifact_group_begin:n and \tag_mc_artifact_group_end:. These functions are documented on page 60.)

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

```

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

```

(End of definition for \tag_mc_reset_box:N. This function is documented on page 61.)

\tag_mc_end_push:
\tag_mc_begin_pop:n

```

78 <base>\cs_new_protected:Npn \tag_mc_end_push: {}
79 <base>\cs_new_protected:Npn \tag_mc_begin_pop:n #1 {}
80 <*shared>
81 \cs_set_protected:Npn \tag_mc_end_push:
82 {
83   \__tag_check_if_active_mc:T
84   {
85     \__tag_mc_if_in:TF
86     {
87       \seq_gpush:Nx \g__tag_mc_stack_seq { \tag_get:n {mc_tag} }
88       \__tag_check_mc_pushed_popped:nn
89       { pushed }
90       { \tag_get:n {mc_tag} }
91       \tag_mc_end:
92     }
93     {
94       \seq_gpush:Nn \g__tag_mc_stack_seq {-1}
95       \__tag_check_mc_pushed_popped:nn { pushed }{-1}
96     }
97   }
98 }
99
100 \cs_set_protected:Npn \tag_mc_begin_pop:n #1
101 {
102   \__tag_check_if_active_mc:T
103   {
104     \seq_gpop:NNTF \g__tag_mc_stack_seq \l__tag_tmpa_tl
105     {
106       \tl_if_eq:NnTF \l__tag_tmpa_tl {-1}
107       {
108         \__tag_check_mc_pushed_popped:nn {popped}{-1}
109       }
110       {
111         \__tag_check_mc_pushed_popped:nn {popped}{\l__tag_tmpa_tl}
112         \tag_mc_begin:n {tag=\l__tag_tmpa_tl,#1}
113       }
114     }
115   }
116 }

```

```

115         {
116             \_tag\_check\_mc\_pushed\_popped:nn {popped}{empty-stack,~nothing}
117         }
118     }
119 }

```

(End of definition for \tag_mc_end_push: and \tag_mc_begin_pop:n. These functions are documented on page 60.)

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.

```

120 \keys\_define:nn { \_tag / mc }
121 {
122     stash .bool\_set:N = \l\_tag\_mc\_key\_stash\_bool,
123     \_artifact\_bool .bool\_set:N = \l\_tag\_mc\_artifact\_bool,
124     \_artifact\_type .choice:,
125     \_artifact\_type / pagination .code:n =
126     {
127         \tl\_set:Nn \l\_tag\_mc\_artifact\_type\_tl { Pagination }
128     },
129     \_artifact\_type / pagination/header .code:n =
130     {
131         \tl\_set:Nn \l\_tag\_mc\_artifact\_type\_tl { Pagination/Subtype/Header }
132     },
133     \_artifact\_type / pagination/footer .code:n =
134     {
135         \tl\_set:Nn \l\_tag\_mc\_artifact\_type\_tl { Pagination/Subtype/Footer }
136     },
137     \_artifact\_type / layout .code:n =
138     {
139         \tl\_set:Nn \l\_tag\_mc\_artifact\_type\_tl { Layout }
140     },
141     \_artifact\_type / page .code:n =
142     {
143         \tl\_set:Nn \l\_tag\_mc\_artifact\_type\_tl { Page }
144     },
145     \_artifact\_type / background .code:n =
146     {
147         \tl\_set:Nn \l\_tag\_mc\_artifact\_type\_tl { Background }
148     },
149     \_artifact\_type / notype .code:n =
150     {
151         \tl\_set:Nn \l\_tag\_mc\_artifact\_type\_tl {}
152     },
153     \_artifact\_type / .code:n =
154     {
155         \tl\_set:Nn \l\_tag\_mc\_artifact\_type\_tl {}
156     },

```

157 }

(End of definition for `stash` (`mc-key`), `__artifact-bool`, and `__artifact-type`. This function is documented on page 61.)

158 </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} {2023-06-14} {0.98i}
4 {part of tagpdf - code related to marking chunks - generic mode}
5 </generic>
6 <*debug>
7 \ProvidesExplPackage {tagpdf-debug-generic} {2023-06-14} {0.98i}
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 of 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 of 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 of definition for `\l__tag_mc_tmpa_tl`.)

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

```
14 \newmarks \g__tag_mc_marks
```

(End of definition for `\g__tag_mc_marks`.)

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

```
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 of definition for `\g__tag_mc_main_marks_seq`, `\g__tag_mc_footnote_marks_seq`, and `\g__tag_mc_multicol_marks_seq`.)

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

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

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

1.2 Functions

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

```
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 of definition for __tag_mc_begin_marks:nn, __tag_mc_artifact_begin_marks:n, and __tag_mc_end_marks:.)

__tag_mc_disable_marks: This disables the marks. They can't be reenabled, so it should only be used in groups.

```

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 of definition for __tag_mc_disable_marks:.)

__tag_mc_get_marks: This stores the current content of the marks in the sequences. It naturally should only be used in places where it makes sense.

```

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 of definition for __tag_mc_get_marks:.)

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

```

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 of 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 of definition for __tag_mc_insert_extra_tmb:n and __tag_mc_insert_extra_tme:n.)

1.3 Looking at MC marks in boxes

__tag_add_missing_mcs:Nn Assumptions:

- test for tagging active outside;
- mark retrieval also outside.

This takes a box register as its first argument (or the register number in a count register, as used by `multicol`). It adds an extra tmb at the top of the box if necessary and similarly an extra tme at the end. This is done by adding hboxes in a way that the positioning and the baseline of the given box is not altered. The result is written back to the box.

The second argument is the stream this box belongs to and is currently either `main` for the main galley, `footnote` for footnote text, or `multicol` for boxes produced for columns in that environment. Other streams may follow over time.

```

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_tmppb_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_tmppb_box \c_zero_dim
177   \box_set_dp:Nn \l__tag_tmppb_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_tmppb_box

```

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

```

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

```

(End of definition for `__tag_add_missing_mcs:Nn`.)

`__tag_add_missing_mcs_to_stream:Nn` This is the main command to add mc to the stream. It is 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 of definition for `__tag_add_missing_mcs_to_stream:Nn`.)

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

`_tag_mc_if_in:TF`
`\tag_mc_if_in_p:` 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 of definition for `_tag_mc_if_in:TF` and `\tag_mc_if_in:TF`. This function is documented on page 60.)

`_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 of definition for `_tag_mc_bmc:n`, `_tag_mc_emc:`, and `_tag_mc_bdc:nn`.)

`_tag_mc_bdc_mcid:nn` This create a BDC mark with an /MCID key. Most of the work here is to get the current 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_abspage_tl
239   {
240     \_tag_ref_value:enn %3 args
241     {
242       mcid-\int_use:N \c@g__tag_MCID_abs_int
243     }
244     { tagabspage }
245     {-1}
246   }
247   \prop_get:NoNTF
248   \g__tag_MCID_byabspage_prop
249   {
250     \l__tag_mc_ref_abspage_tl
```

```

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 of definition for __tag_mc_bdc_mcid:nn, __tag_mc_bdc_mcid:n, and __tag_mc_handle_mcid:nn.)

__tag_mc_handle_stash:n This is the handler which puts a mc into the the current structure. The argument is the
 __tag_mc_handle_stash:x 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 of definition for __tag_mc_handle_stash:n.)

__tag_mc_bmc_artifact: Two commands to create artifacts, one without type, and one with. We define also a wrapper handler as luamode will need a different definition. TODO: perhaps later: more properties for artifacts

```

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 of definition for __tag_mc_bmc_artifact:, __tag_mc_bmc_artifact:n, and __tag_mc_handle_artifact:N.)

__tag_get_data_mc_tag: This allows to retrieve the active mc-tag. It is use by the get command.

```

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

```

(End of definition for __tag_get_data_mc_tag:.)

\tag_mc_begin:n These are the core public commands to open and close an mc. They don't need to be in the same group or grouping level, but the code expect that they are issued linearly. 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: \int_gincr:N \c@g__tag_MCID_
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     \cs_set_protected:Npn \tag_mc_begin:n #1 %#1 keyval
329       {
330         \__tag_check_if_active_mc:TF
331         {
332           \__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
338 set default MC tags to structure:
339
340     \tl_set_eq:NN \l__tag_mc_key_tag_tl \g__tag_struct_tag_tl
341     \tl_gset_eq:NN \g__tag_mc_key_tag_tl \g__tag_struct_tag_tl
342     \keys_set:nn { __tag / mc } {#1}
343     \bool_if:NTF \l__tag_mc_artifact_bool
344     { %handle artifact
345       \__tag_mc_handle_artifact:N \l__tag_mc_artifact_type_tl
346       \exp_args:NV
347       \__tag_mc_artifact_begin_marks:n \l__tag_mc_artifact_type_tl
348     }
349     { %handle mcid type
350       \__tag_check_mc_tag:N \l__tag_mc_key_tag_tl
351       \__tag_mc_handle_mcid:VV
352       \l__tag_mc_key_tag_tl
353       \l__tag_mc_key_properties_tl
354       \__tag_mc_begin_marks:oo{\l__tag_mc_key_tag_tl}{\l__tag_mc_key_label_tl}
355       \tl_if_empty:NF {\l__tag_mc_key_label_tl}
356       {
357         \exp_args:NV
358         \__tag_mc_handle_mc_label:n \l__tag_mc_key_label_tl
359       }
360       \bool_if:NF \l__tag_mc_key_stash_bool
361       {
362         \exp_args:NV \__tag_struct_get_parentrole:nNN
363         \g__tag_struct_stack_current_tl
364         \l__tag_get_parent_tmpa_tl
365         \l__tag_get_parent_tmpb_tl
366         \__tag_check_parent_child:VVnnN
367         \l__tag_get_parent_tmpa_tl
368         \l__tag_get_parent_tmpb_tl
369         {MC}{ }
370         \l__tag_parent_child_check_tl
371         \int_compare:nNnT {\l__tag_parent_child_check_tl}<{0}
372         {
373           \prop_get:cnN
374           { g__tag_struct_ \g__tag_struct_stack_current_tl _prop }
375           {S}
376           \l__tag_tmpa_tl
377           \msg_warning:nnxxx
378           { tag }
379           {role-parent-child}
380           { \l__tag_get_parent_tmpa_tl/\l__tag_get_parent_tmpb_tl }
381           { MC~(real content) }
382           { not~allowed~
383             (struct~\g__tag_struct_stack_current_tl,~\l__tag_tmpa_tl)
384           }
385         }
386       }
387       \__tag_mc_handle_stash:x { \int_use:N \c@g__tag_MCID_abs_int }
388     }
389   }

```

```

386     \group_end:
387   }
388   < *debug >
389   {
390     \__tag_debug_mc_begin_ignore:n { #1 }
391   }
392 < /debug >
393   }
394   < *generic >
395   \cs_set_protected:Nn \tag_mc_end:
396   {
397     \__tag_check_if_active_mc:T
398     {
399       < /generic >
400       < *debug >
401       \cs_set_protected:Nn \tag_mc_end:
402       {
403         \__tag_check_if_active_mc:TF
404         {
405           \__tag_debug_mc_end_insert:
406           < /debug >
407           \__tag_check_mc_if_open:
408           \bool_gset_false:N \g__tag_in_mc_bool
409           \tl_gset:Nn \g__tag_mc_key_tag_tl { }
410           \__tag_mc_emc:
411           \__tag_mc_end_marks:
412         }
413         < *debug >
414         {
415           \__tag_debug_mc_end_ignore:
416         }
417       < /debug >
418     }
419   < /generic | debug >

```

(End of definition for \tag_mc_begin:n and \tag_mc_end:. These functions are documented on page 60.)

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) 420 < *generic >
alt_␣(mc-key) 421 \keys_define:nn { __tag / mc }
actualtext_␣(mc-key) 422 {
label_␣(mc-key) 423   tag .code:n = % the name (H,P,Span) etc
artifact_␣(mc-key) 424   {
425     \tl_set:Nx \l__tag_mc_key_tag_tl { #1 }
426     \tl_gset:Nx \g__tag_mc_key_tag_tl { #1 }
427   },
428   raw .code:n =
429   {

```

```

430     \tl_put_right:Nx \l__tag_mc_key_properties_tl { #1 }
431   },
432   alt .code:n      = % Alt property
433   {
434     \str_set_convert:Noon
435     \l__tag_tmpa_str
436     { #1 }
437     { default }
438     { utf16/hex }
439     \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt-< }
440     \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
441   },
442   alttext .meta:n = {alt=#1},
443   actualtext .code:n      = % ActualText property
444   {
445     \tl_if_empty:oF{#1}
446     {
447       \str_set_convert:Noon
448       \l__tag_tmpa_str
449       { #1 }
450       { default }
451       { utf16/hex }
452       \tl_put_right:Nn \l__tag_mc_key_properties_tl { /ActualText~< }
453       \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
454     }
455   },
456   label .tl_set:N      = \l__tag_mc_key_label_tl,
457   artifact .code:n     =
458   {
459     \exp_args:Nnx
460     \keys_set:nn
461     { __tag / mc }
462     { __artifact-bool, __artifact-type=#1 }
463   },
464   artifact .default:n   = {notype}
465 }
466 </generic>

```

(End of definition for tag (mc-key) and others. These functions are documented on page 61.)

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} {2023-06-14} {0.98i}
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 of definition for `__tag_add_missing_mcs_to_stream:Nn`.)

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

`__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(\int_use:N \c_document_cctab,tex.getattribute(luatexbase.attributes.g__tag_mc_if_in_p:TF))`

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 of definition for `__tag_mc_if_in:TF` and `\tag_mc_if_in:TF`. This function is documented on page 60.)

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

`__tag_mc_lua_set_mc_type_attr:o` 74 `\cs_new:Nn __tag_mc_lua_set_mc_type_attr:n % #1 is a tag name`

`__tag_mc_lua_unset_mc_type_attr:` 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 of definition for __tag_mc_lua_set_mc_type_attr:n and __tag_mc_lua_unset_mc_type_attr:.)

__tag_mc_insert_mcid_kids:n These commands will in the finish code replace the dummy for a mc by the real mcid
 __tag_mc_insert_mcid_single_kids:n kids we need a variant for the case that it is the only kid, to get the array right

```

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 of definition for __tag_mc_insert_mcid_kids:n and __tag_mc_insert_mcid_single_kids:n)

__tag_mc_handle_stash:n This is the lua variant for the command to put an mcid absolute number in the current
 __tag_mc_handle_stash: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 of definition for \__tag_mc_handle_stash:n.)

```

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

```

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         \tl_set_eq:NN \l__tag_mc_key_tag_tl \g__tag_struct_tag_tl
165         \tl_gset_eq:NN \g__tag_mc_key_tag_tl \g__tag_struct_tag_tl
166         \lua_now:e
167         {
168             ltx.__tag.func.store_mc_data(\__tag_get_mc_abs_cnt:,"tag"," \g__tag_struct_tag_tl",
169             }
170         \keys_set:nn { __tag / mc }{ label={}, #1 }
171         %check that a tag or artifact has been used
172         \__tag_check_mc_tag:N \l__tag_mc_key_tag_tl
173         %set the attributes:
174         \__tag_mc_lua_set_mc_type_attr:o { \l__tag_mc_key_tag_tl }
175         \bool_if:NF \l__tag_mc_artifact_bool
176         { % store the absolute num name in a label:
177             \tl_if_empty:NF {\l__tag_mc_key_label_tl}
178             {
179                 \exp_args:NV
180                 \__tag_mc_handle_mc_label:n \l__tag_mc_key_label_tl
181             }
182             % if not stashed record the absolute number
183             \bool_if:NF \l__tag_mc_key_stash_bool
184             {
185                 \exp_args:NV \__tag_struct_get_parentrole:nNN
186                 \g__tag_struct_stack_current_tl
187                 \l__tag_get_parent_tmpa_tl
188                 \l__tag_get_parent_tmpb_tl
189                 \__tag_check_parent_child:VVnnN
190                 \l__tag_get_parent_tmpa_tl
191                 \l__tag_get_parent_tmpb_tl
192                 {MC}{ }
193                 \l__tag_parent_child_check_tl
194                 \int_compare:nNnT {\l__tag_parent_child_check_tl}<{0}
195                 {

```

```

196         \prop_get:cnN
197         { g__tag_struct_ \g__tag_struct_stack_current_tl _prop}
198         {S}
199         \l__tag_tmpa_tl
200         \msg_warning:nnxxx
201         { tag }
202         {role-parent-child}
203         { \l__tag_get_parent_tmpa_tl/\l__tag_get_parent_tmpb_tl }
204         { MC~(real content) }
205         {
206             not-allowed~
207             (struct~\g__tag_struct_stack_current_tl,~\l__tag_tmpa_tl)
208         }
209     }
210     \__tag_mc_handle_stash:x { \__tag_get_mc_abs_cnt: }
211 }
212 }
213 \group_end:
214 }
215 }

```

(End of definition for \tag_mc_begin:n. This function is documented on page 60.)

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

```

216 \cs_set_protected:Nn \tag_mc_end:
217 {
218     \__tag_check_if_active_mc:T
219     {
220         %\__tag_check_mc_if_open:
221         \bool_gset_false:N \g__tag_in_mc_bool
222         \bool_set_false:N\l__tag_mc_artifact_bool
223         \__tag_mc_lua_unset_mc_type_attr:
224         \tl_set:Nn \l__tag_mc_key_tag_tl { }
225         \tl_gset:Nn \g__tag_mc_key_tag_tl { }
226     }
227 }

```

(End of definition for \tag_mc_end:. This function is documented on page 60.)

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

```

228 \cs_set_protected:Npn \tag_mc_reset_box:N #1
229 {
230     \lua_now:e
231     {
232         local~type=tex.getattribute(luatexbase.attributes.g__tag_mc_type_attr)
233         local~mc=tex.getattribute(luatexbase.attributes.g__tag_mc_cnt_attr)
234         ltx.__tag.func.update_mc_attributes(tex.getbox(\int_use:N #1),mc,type)
235     }
236 }

```

(End of definition for \tag_mc_reset_box:N. This function is documented on page 61.)

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

```

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

```

(End of definition for `__tag_get_data_mc_tag:.`)

1.2 Key definitions

```
tag_(mc-key)  TODO: check conversion, check if local/global setting is right.
raw_(mc-key)  238 \keys_define:nn { __tag / mc }
alt_(mc-key)  239 {
actualtext_(mc-key) 240   tag .code:n = %
label_(mc-key)  241   {
artifact_(mc-key) 242     \tl_set:Nx \l__tag_mc_key_tag_tl { #1 }
                243     \tl_gset:Nx \g__tag_mc_key_tag_tl { #1 }
                244     \lua_now:e
                245     {
                246       ltx.__tag.func.store_mc_data(\__tag_get_mc_abs_cnt:,"tag","#1")
                247     }
                248   },
                249   raw .code:n =
                250   {
                251     \tl_put_right:Nx \l__tag_mc_key_properties_tl { #1 }
                252     \lua_now:e
                253     {
                254       ltx.__tag.func.store_mc_data(\__tag_get_mc_abs_cnt:,"raw","#1")
                255     }
                256   },
                257   alt .code:n      = % Alt property
                258   {
                259     \tl_if_empty:oF{#1}
                260     {
                261       \str_set_convert:Noon
                262       \l__tag_tmpa_str
                263       { #1 }
                264       { default }
                265       { utf16/hex }
                266       \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
                267       \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
                268       \lua_now:e
                269       {
                270         ltx.__tag.func.store_mc_data
                271         (
                272           \__tag_get_mc_abs_cnt:,"alt","/Alt~<\str_use:N \l__tag_tmpa_str>"
                273         )
                274       }
                275     }
                276   },
                277   alttext .meta:n = {alt=#1},
                278   actualtext .code:n      = % Alt property
                279   {
                280     \tl_if_empty:oF{#1}
                281     {
                282       \str_set_convert:Noon
                283       \l__tag_tmpa_str
                284       { #1 }
                285       { default }
                286       { utf16/hex }
```

```

287         \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
288         \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
289         \lua_now:e
290         {
291             ltx.__tag.func.store_mc_data
292             (
293                 \__tag_get_mc_abs_cnt:,
294                 "actualtext",
295                 "/ActualText~<\str_use:N \l__tag_tmpa_str>"
296             )
297         }
298     },
299     label .code:n =
300     {
301         \tl_set:Nn\l__tag_mc_key_label_tl { #1 }
302         \lua_now:e
303         {
304             ltx.__tag.func.store_mc_data
305             (
306                 \__tag_get_mc_abs_cnt:,"label","#1"
307             )
308         }
309     },
310     __artifact-store .code:n =
311     {
312         \lua_now:e
313         {
314             ltx.__tag.func.store_mc_data
315             (
316                 \__tag_get_mc_abs_cnt:,"artifact","#1"
317             )
318         }
319     },
320     artifact .code:n =
321     {
322         \exp_args:Nnx
323         \keys_set:nn
324         { __tag / mc }
325         { __artifact-bool, __artifact-type=#1, tag=Artifact }
326         \exp_args:Nnx
327         \keys_set:nn
328         { __tag / mc }
329         { __artifact-store=\l__tag_mc_artifact_type_tl }
330     },
331     artifact .default:n = { notype }
332 }
333
334
335 </luamode>

```

(End of definition for tag (mc-key) and others. These functions are documented on page 61.)

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>
<code>\tag_struct_end:n</code>	<code>\tag_struct_end:n{<tag>}</code>

These commands start and end a new structure. They don't start a group. They set all their values globally. `\tag_struct_end:n` does nothing special normally (apart from swallowing its argument, but if `tagpdf-debug` is loaded, it will check if the `{<tag>}` (after expansion) is identical to the current structure on the stack. The tag is not role mapped!

<code>\tag_struct_use:n</code>	<code>\tag_struct_use:n{<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 \tag_struct_use:n (where a real label is actually not needed as you can only use structures already defined), and by the ref key (which can refer to future structures). Internally the label name will start with tagpdfstruct- and it stores the two attributs 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 \tag_get:n , but one can also use a label on the parent structure and then use \ref_value:nn{tagpdfstruct-label}{tagstruct} 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. If it is empty, nothing will happen.
<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. If it is empty, nothing will happen.
<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} {2023-06-14} {0.98i}
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 of definition for \c@g__tag_struct_abs_int.)

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

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

(End of definition for \g__tag_struct_objR_seq.)

`\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 of definition for \g__tag_struct_cont_mc_prop.)

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

```
11 \seq_new:N \g__tag_struct_stack_seq
12 \seq_gpush:Nn \g__tag_struct_stack_seq {0}
```

(End of definition for `\g__tag_struct_stack_seq`.)

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

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

(End of definition for `\g__tag_struct_tag_stack_seq`.)

`\g__tag_struct_stack_current_tl` The global variable will hold the current structure number. It is already defined in `tagpdf-base`.
`\l__tag_struct_stack_parent_tmpa_tl` 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 of 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 of definition for \c__tag_struct_StructTreeRoot_entries_seq and \c__tag_struct_StructElem_entries_seq.)

3.1 Variables used by the keys

<pre> \g__tag_struct_tag_tl \g__tag_struct_tag_NS_tl \l__tag_struct_roletag_tl \g__tag_struct_roletag_NS_tl </pre>	<p>Use by the tag key to store the tag and the namespace. The role tag variables will hold locally rolemapping info needed for the parent-child checks</p> <pre> 57 \tl_new:N \g__tag_struct_tag_tl 58 \tl_new:N \g__tag_struct_tag_NS_tl 59 \tl_new:N \l__tag_struct_roletag_tl 60 \tl_new:N \l__tag_struct_roletag_NS_tl </pre> <p>(End of definition for \g__tag_struct_tag_tl and others.)</p>
<pre> \l__tag_struct_key_label_tl </pre>	<p>This will hold the label value.</p> <pre> 61 \tl_new:N \l__tag_struct_key_label_tl </pre> <p>(End of definition for \l__tag_struct_key_label_tl.)</p>
<pre> \l__tag_struct_elem_stash_bool </pre>	<p>This will keep track of the stash status</p> <pre> 62 \bool_new:N \l__tag_struct_elem_stash_bool </pre> <p>(End of definition for \l__tag_struct_elem_stash_bool.)</p>

3.2 Variables used by tagging code of basic elements

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

```

63 \endpackage
64 \base\prop_new:N \g__tag_struct_dest_num_prop
65 \endpackage

```

(End of definition for \g__tag_struct_dest_num_prop.)

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

```

66 \prop_new:N \g__tag_struct_ref_by_dest_prop

```

(End of definition for \g__tag_struct_ref_by_dest_prop.)

4 Commands

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

```

\__tag_struct_output_prop_aux:nn
\__tag_new_output_prop_handler:n
67 \cs_new:Npn \__tag_struct_output_prop_aux:nn #1 #2 % #1 num, #2 key
68 {
69   \prop_if_in:cnT
70     { \g__tag_struct_#1_prop }
71     { #2 }
72   {
73     \c_space_tl/#2~ \prop_item:cn{ \g__tag_struct_#1_prop } { #2 }
74   }
75 }
76
77 \cs_new_protected:Npn \__tag_new_output_prop_handler:n #1
78 {
79   \cs_new:cn { \__tag_struct_output_prop_#1:n }
80   {
81     \__tag_struct_output_prop_aux:nn {#1}{##1}
82   }
83 }

```

(End of 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.

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

__tag_pdf_name_e:n

85 \cs_new:Npn \__tag_pdf_name_e:n #1{\pdf_name_from_unicode_e:n{#1}}

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

```
g__tag_struct_0_prop
g__tag_struct_kids_0_seq

86 \__tag_prop_new:c { g__tag_struct_0_prop }
87 \__tag_new_output_prop_handler:n {0}
88 \__tag_seq_new:c { g__tag_struct_kids_0_seq }
89
90 \__tag_prop_gput:cnx
91 { g__tag_struct_0_prop }
92 { Type }
93 { \pdf_name_from_unicode_e:n {StructTreeRoot} }
94
95 \__tag_prop_gput:cnx
96 { g__tag_struct_0_prop }
97 { S }
98 { \pdf_name_from_unicode_e:n {StructTreeRoot} }
99
100 \__tag_prop_gput:cnx
101 { g__tag_struct_0_prop }
102 { rolemap }
103 { {StructTreeRoot}{pdf} }
104
105 \__tag_prop_gput:cnx
106 { g__tag_struct_0_prop }
107 { parentrole }
108 { {StructTreeRoot}{pdf} }
109
```

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

```
110 \pdf_version_compare:NnF < {2.0}
111 {
112   \__tag_prop_gput:cnx
113   { g__tag_struct_0_prop }
114   { Namespaces }
115   { \pdf_object_ref:n { __tag/tree/namespaces } }
116 }
```

(End of definition for g__tag_struct_0_prop and g__tag_struct_kids_0_seq.)

4.2 Adding the /ID key

Every structure gets automatically an ID which is currently simply calculated from the structure number.

```

__tag_struct_get_id:n
117 \cs_new:Npn __tag_struct_get_id:n #1 %#1=struct num
118 {
119   (
120     ID.
121     \prg_replicate:nn
122     { \int_abs:n{\g__tag_tree_id_pad_int - \tl_count:e { \int_to_arabic:n { #1 } }} }
123     { 0 }
124     \int_to_arabic:n { #1 }
125   )
126 }

(End of definition for __tag_struct_get_id:n.)

```

4.3 Filling in the tag info

This adds or updates the tag info to a structure given by a number. We need also the original data, so we store both.

```

__tag_struct_set_tag_info:nnn
127 \pdf_version_compare:NnTF < {2.0}
128 {
129   \cs_new_protected:Npn __tag_struct_set_tag_info:nnn #1 #2 #3
130   %#1 structure number, #2 tag, #3 NS
131   {
132     __tag_prop_gput:cnx
133     { g__tag_struct_#1_prop }
134     { S }
135     { \pdf_name_from_unicode_e:n {#2} } %
136   }
137 }
138 {
139   \cs_new_protected:Npn __tag_struct_set_tag_info:nnn #1 #2 #3
140   {
141     __tag_prop_gput:cnx
142     { g__tag_struct_#1_prop }
143     { S }
144     { \pdf_name_from_unicode_e:n {#2} } %
145     \prop_get:NnNT \g__tag_role_NS_prop {#3} \l__tag_get_tmpc_tl
146     {
147       __tag_prop_gput:cnx
148       { g__tag_struct_#1_prop }
149       { NS }
150       { \l__tag_get_tmpc_tl } %
151     }
152   }
153 }
154 \cs_generate_variant:Nn __tag_struct_set_tag_info:nnn {eVV}

(End of definition for __tag_struct_set_tag_info:nnn.)

```

`_tag_struct_get_parentrole:nNN`

We also need a way to get the tag info needed for parent child check from parent structures.

```
155 \cs_new_protected:Npn \_tag_struct_get_parentrole:nNN #1 #2 #3
156   {%#1 struct num, #2 tlvvar for tag , #3 tlvvar for NS
157   {
158     \prop_get:cnNTF
159     { g__tag_struct_#1_prop }
160     { parentrole }
161     \l__tag_get_tmpc_tl
162     {
163       \tl_set:Nx #2{\exp_last_unbraced:NV\use_i:nn \l__tag_get_tmpc_tl}
164       \tl_set:Nx #3{\exp_last_unbraced:NV\use_ii:nn \l__tag_get_tmpc_tl}
165     }
166     {
167       \tl_clear:N#2
168       \tl_clear:N#3
169     }
170   }
171 \cs_generate_variant:Nn\_tag_struct_get_parentrole:nNN {eNN}

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

4.4 Handlings kids

Commands to store the kids. Kids in a structure can be a reference to a mc-chunk, an object reference to another structure element, or a object reference to an annotation (through an OBJR object).

`_tag_struct_kid_mc_gput_right:nn`

`_tag_struct_kid_mc_gput_right:nx`

The command to store an mc-chunk, this is a dictionary of type MCR. It would be possible to write out the content directly as unnamed object and to store only the object reference, but probably this would be slower, and the PDF is more readable like this. The code doesn't try to avoid the use of the /Pg key by checking page numbers. That imho only slows down without much gain. In generic mode the page break code will perhaps have to insert an additional mcid after an existing one. For this we use a property list At first an auxiliary to write the MCID dict. This should normally be expanded!

```
172 \cs_new:Npn \_tag_struct_mcid_dict:n #1 {%#1 MCID absnum
173   {
174     <<
175     /Type \c_space_tl /MCR \c_space_tl
176     /Pg
177     \c_space_tl
178     \pdf_pageobject_ref:n { \_tag_ref_value:enn{mcid-#1}{tagabspage}{1} }
179     /MCID \c_space_tl \_tag_ref_value:enn{mcid-#1}{tagmcid}{1}
180     >>
181   }
182 \cs_new_protected:Npn \_tag_struct_kid_mc_gput_right:nn #1 #2 {%#1 structure num, #2 MCID abs
183   {
184     \_tag_seq_gput_right:cx
185     { g__tag_struct_kids_#1_seq }
186     {
187       \_tag_struct_mcid_dict:n {#2}
188     }
189     \_tag_seq_gput_right:cn
```

```

190     { g__tag_struct_kids_#1_seq }
191     {
192         \prop_item:Nn \g__tag_struct_cont_mc_prop {#2}
193     }
194 }
195 \cs_generate_variant:Nn \__tag_struct_kid_mc_gput_right:nn {nx}
196

```

(End of 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.

```

197 \cs_new_protected:Npn \__tag_struct_kid_struct_gput_right:nn #1 #2 %#1 num of parent struct, #2
198 {
199     \__tag_seq_gput_right:cx
200     { g__tag_struct_kids_#1_seq }
201     {
202         \pdf_object_ref:n { __tag/struct/#2 }
203     }
204 }
205
206 \cs_generate_variant:Nn \__tag_struct_kid_struct_gput_right:nn {xx}

```

(End of 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

```

207 \cs_new_protected:Npn \__tag_struct_kid_OBJR_gput_right:nnn #1 #2 #3 %#1 num of parent struct,
208                                     %#2 obj reference
209                                     %#3 page object reference
210 {
211     \pdf_object_unnamed_write:nn
212     { dict }
213     {
214         /Type/OBJR/Obj~#2/Pg~#3
215     }
216     \__tag_seq_gput_right:cx
217     { g__tag_struct_kids_#1_seq }
218     {
219         \pdf_object_ref_last:
220     }
221 }
222
223 \cs_generate_variant:Nn \__tag_struct_kid_OBJR_gput_right:nnn { xxx }
224

```

(End of definition for __tag_struct_kid_OBJR_gput_right:nnn.)

__tag_struct_exchange_kid_command:N
__tag_struct_exchange_kid_command:c

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

```

225 \cs_new_protected:Npn \__tag_struct_exchange_kid_command:N #1 %#1 = seq var

```

```

226 {
227   \seq_gpop_left:NN #1 \l__tag_tmpa_tl
228   \regex_replace_once:nnN
229     { \c{\__tag_mc_insert_mcid_kids:n} }
230     { \c{\__tag_mc_insert_mcid_single_kids:n} }
231     \l__tag_tmpa_tl
232   \seq_gput_left:NV #1 \l__tag_tmpa_tl
233 }
234
235 \cs_generate_variant:Nn\__tag_struct_exchange_kid_command:N { c }

```

(End of definition for __tag_struct_exchange_kid_command:N.)

__tag_struct_fill_kid_key:n This command adds the kid info to the K entry. In lua mode the content contains commands which are expanded later. The argument is the structure number.

```

236 \cs_new_protected:Npn \__tag_struct_fill_kid_key:n #1 %#1 is the struct num
237 {
238   \bool_if:NF\g__tag_mode_lua_bool
239   {
240     \seq_clear:N \l__tag_tmpa_seq
241     \seq_map_inline:cn { g__tag_struct_kids_#1_seq }
242       { \seq_put_right:Nx \l__tag_tmpa_seq { ##1 } }
243     %\seq_show:c { g__tag_struct_kids_#1_seq }
244     %\seq_show:N \l__tag_tmpa_seq
245     \seq_remove_all:Nn \l__tag_tmpa_seq {}
246     %\seq_show:N \l__tag_tmpa_seq
247     \seq_gset_eq:cN { g__tag_struct_kids_#1_seq } \l__tag_tmpa_seq
248   }
249
250   \int_case:nnF
251   {
252     \seq_count:c
253     {
254       g__tag_struct_kids_#1_seq
255     }
256   }
257   {
258     { 0 }
259     { } %no kids, do nothing
260     { 1 } % 1 kid, insert
261     {
262       % in this case we need a special command in
263       % luamode to get the array right. See issue #13
264       \bool_if:NT\g__tag_mode_lua_bool
265       {
266         \__tag_struct_exchange_kid_command:c
267         {g__tag_struct_kids_#1_seq}
268       }
269       \__tag_prop_gput:cnx { g__tag_struct_#1_prop } {K}
270       {
271         \seq_item:cn
272         {
273           g__tag_struct_kids_#1_seq
274         }

```

```

275         {1}
276     }
277 } %
278 }
279 { %many kids, use an array
280   \__tag_prop_gput:cnx { g__tag_struct_#1_prop } {K}
281   {
282     [
283       \seq_use:cn
284       {
285         g__tag_struct_kids_#1_seq
286       }
287       {
288         \c_space_tl
289       }
290     ]
291   }
292 }
293 }
294

```

(End of definition for __tag_struct_fill_kid_key:n.)

4.5 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.

```

295 \cs_new_protected:Npn \__tag_struct_get_dict_content:nN #1 #2 %#1: structure num
296 {
297   \tl_clear:N #2
298   \seq_map_inline:cn
299   {
300     c__tag_struct_
301     \int_compare:nNnTF{#1}={0}{StructTreeRoot}{StructElem}
302     _entries_seq
303   }
304   {
305     \tl_put_right:Nx
306     #2
307     {
308       \prop_if_in:cnT
309       { g__tag_struct_#1_prop }
310       { ##1 }
311       {
312         \c_space_tl/##1~

```

Some keys needs the option to format the key, e.g. add brackets for an array

```

313     \cs_if_exist_use:cTF {__tag_struct_format_##1:e}
314     {
315       { \prop_item:cn{ g__tag_struct_#1_prop } { ##1 } }
316     }
317     {
318       \prop_item:cn{ g__tag_struct_#1_prop } { ##1 }

```

```

319         }
320     }
321 }
322 }
323 }

```

(End of 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:

```

324 \cs_new:Nn __tag_struct_format_Ref:n{[#1]}
325 \cs_generate_variant:Nn __tag_struct_format_Ref:n{e}

```

(End of 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.

```

326 \cs_new_protected:Npn __tag_struct_write_obj:n #1 % #1 is the struct num
327 {
328     \pdf_object_if_exist:nTF { __tag/struct/#1 }
329     {

```

It can happen that a structure is not used and so has not parent. Simply ignoring it is problematic as it is also recorded in the IDTree, so we make an artifact out of it.

```

330         \prop_get:cnNF { g__tag_struct_#1_prop } {P}\l__tag_tmpb_tl
331     {
332         \prop_gput:cnx { g__tag_struct_#1_prop } {P}{\pdf_object_ref:n { __tag/struct/0 }
333         \prop_gput:cnx { g__tag_struct_#1_prop } {S}{/Artifact}
334         \seq_if_empty:cF {g__tag_struct_kids_#1_seq}
335         {
336             \msg_warning:nnxx
337                 {tag}
338                 {struct-orphan}
339                 { #1 }
340             {\seq_count:c{g__tag_struct_kids_#1_seq}}
341         }
342     }
343     \__tag_struct_fill_kid_key:n { #1 }
344     \__tag_struct_get_dict_content:nN { #1 } \l__tag_tmpa_tl
345     \exp_args:Nx
346         \pdf_object_write:nnx
347         { __tag/struct/#1 }
348         {dict}
349         {
350             \l__tag_tmpa_tl\c_space_tl
351             /ID~\__tag_struct_get_id:n{#1}
352         }
353
354     }
355     {
356         \msg_error:nnn { tag } { struct-no-objnum } { #1}
357     }
358 }

```

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

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

Annotations used as structure content must

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

For a link this looks like this

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

```
359 \cs_new_protected:Npn \__tag_struct_insert_annot:nn #1 #2 %1 object reference to the annotat
360                                     %#2 structparent number
361 {
362   \bool_if:NT \g__tag_active_struct_bool
363   {
364     %get the number of the parent structure:
365     \seq_get:NNF
366       \g__tag_struct_stack_seq
367       \l__tag_struct_stack_parent_tmpa_tl
368       {
369         \msg_error:nn { tag } { struct-faulty-nesting }
370       }
371     %put the obj number of the annot in the kid entry, this also creates
372     %the OBJR object
373     \ref_label:nn {__tag_objr_page_#2 }{ tagabspage }
374     \__tag_struct_kid_OBJR_gput_right:xxx
375     {
376       \l__tag_struct_stack_parent_tmpa_tl
377     }
378     {
379       #1 %
380     }
381     {
382       \pdf_pageobject_ref:n { \__tag_ref_value:nnn {__tag_objr_page_#2 }{ tagabspage }{
383     }
384     % add the parent obj number to the parent tree:
385     \exp_args:Nnx
386     \__tag_parenttree_add_objr:nn
387     {
388       #2
389     }
390     {
```

```

391         \pdf_object_ref:e { __tag/struct/\l__tag_struct_stack_parent_tmpa_tl }
392     }
393     % increase the int:
394     \stepcounter{ g__tag_parenttree_obj_int }
395 }
396 }

```

(End of definition for __tag_struct_insert_annot:nn.)

__tag_get_data_struct_tag: this command allows \tag_get:n to get the current structure tag with the keyword **struct_tag**.

```

397 \cs_new:Npn \__tag_get_data_struct_tag:
398 {
399     \exp_args:Ne
400     \tl_tail:n
401     {
402         \prop_item:cn {g__tag_struct_\g__tag_struct_stack_current_tl _prop}{S}
403     }
404 }

```

(End of definition for __tag_get_data_struct_tag:.)

__tag_get_data_struct_id: this command allows \tag_get:n to get the current structure id with the keyword **struct_id**.

```

405 \cs_new:Npn \__tag_get_data_struct_id:
406 {
407     \__tag_struct_get_id:n {\g__tag_struct_stack_current_tl}
408 }
409 \</package>

```

(End of definition for __tag_get_data_struct_id:.)

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

```

410 \< *base>
411 \cs_new:Npn \__tag_get_data_struct_num:
412 {
413     \g__tag_struct_stack_current_tl
414 }
415 \</base>

```

(End of definition for __tag_get_data_struct_num:.)

__tag_get_data_struct_counter: this command allows \tag_get:n to get the current state of the structure counter with the keyword **struct_counter**. By comparing the numbers it can be used to check the number of structure commands in a piece of code.

```

416 \< *base>
417 \cs_new:Npn \__tag_get_data_struct_counter:
418 {
419     \int_use:N \c@g__tag_struct_abs_int
420 }
421 \</base>

```

(End of definition for __tag_get_data_struct_counter:.)

5 Keys

This are the keys for the user commands. we store the tag in a variable. But we should be careful, it is only reliable at the begin.

```

label_(struct-key)
stash_(struct-key) 422 (*package)
parent_(struct-key) 423 \keys_define:nn { __tag / struct }
tag_(struct-key)    424 {
title_(struct-key)  425   label .tl_set:N      = \l__tag_struct_key_label_tl,
title-o_(struct-key) 426   stash .bool_set:N    = \l__tag_struct_elem_stash_bool,
alt_(struct-key)    427   parent .code:n      =
actualtext_(struct-key) 428   {
lang_(struct-key)   429     \bool_lazy_and:nnTF
ref_(struct-key)    430     {
E_(struct-key)      431       \prop_if_exist_p:c { g__tag_struct\_int_eval:n {#1}_prop }
432     }
433     {
434       \int_compare_p:nNn {#1}<{\c@g__tag_struct_abs_int}
435     }
436     { \tl_set:Nx \l__tag_struct_stack_parent_tmpa_tl { \int_eval:n {#1} } }
437     {
438       \msg_warning:nxxx { tag } { struct-unknown }
439       { \int_eval:n {#1} }
440       { parent-key-ignored }
441     }
442   },
443   parent .default:n    = {-1},
444   tag .code:n          = % S property
445   {
446     \seq_set_split:Nne \l__tag_tmpa_seq { / } {#1/\prop_item:Ne\g__tag_role_tags_NS_prop{
447     \tl_gset:Nx \g__tag_struct_tag_tl { \seq_item:Nn\l__tag_tmpa_seq {1} }
448     \tl_gset:Nx \g__tag_struct_tag_NS_tl{ \seq_item:Nn\l__tag_tmpa_seq {2} }
449     \__tag_check_structure_tag:N \g__tag_struct_tag_tl
450   },
451   title .code:n        = % T property
452   {
453     \str_set_convert:Nnnn
454     \l__tag_tmpa_str
455     { #1 }
456     { default }
457     { utf16/hex }
458     \__tag_prop_gput:cnx
459     { g__tag_struct\_int_eval:n {\c@g__tag_struct_abs_int}_prop }
460     { T }
461     { <\l__tag_tmpa_str> }
462   },
463   title-o .code:n      = % T property
464   {
465     \str_set_convert:Nonn
466     \l__tag_tmpa_str
467     { #1 }
468     { default }
469     { utf16/hex }

```

```

470     \_tag_prop_gput:cnx
471     { g__tag_struct\_int_eval:n {\c@g__tag_struct_abs_int}_prop }
472     { T }
473     { <\l__tag_tmpa_str> }
474 },
475 alt .code:n          = % Alt property
476 {
477     \tl_if_empty:oF{#1}
478     {
479         \str_set_convert:Noon
480         \l__tag_tmpa_str
481         { #1 }
482         { default }
483         { utf16/hex }
484         \_tag_prop_gput:cnx
485         { g__tag_struct\_int_eval:n {\c@g__tag_struct_abs_int}_prop }
486         { Alt }
487         { <\l__tag_tmpa_str> }
488     }
489 },
490 alttext .meta:n = {alt=#1},
491 actualtext .code:n = % ActualText property
492 {
493     \tl_if_empty:oF{#1}
494     {
495         \str_set_convert:Noon
496         \l__tag_tmpa_str
497         { #1 }
498         { default }
499         { utf16/hex }
500         \_tag_prop_gput:cnx
501         { g__tag_struct\_int_eval:n {\c@g__tag_struct_abs_int}_prop }
502         { ActualText }
503         { <\l__tag_tmpa_str>}
504     }
505 },
506 lang .code:n          = % Lang property
507 {
508     \_tag_prop_gput:cnx
509     { g__tag_struct\_int_eval:n {\c@g__tag_struct_abs_int}_prop }
510     { Lang }
511     { (#1) }
512 },

```

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

```

513 ref .code:n          = % ref property
514 {
515     \tl_clear:N\l__tag_tmpa_tl
516     \clist_map_inline:on {#1}
517     {
518         \tl_put_right:Nx \l__tag_tmpa_tl
519         {~\ref_value:nn{tagpdfstruct-##1}{tagstructobj} }
520     }
521     \_tag_struct_gput_data_ref:ee { \int_eval:n {\c@g__tag_struct_abs_int} } {\l__tag_tm
522 },

```

```

523     E .code:n          = % E property
524     {
525         \str_set_convert:Nnon
526         \l__tag_tmpa_str
527         { #1 }
528         { default }
529         { utf16/hex }
530         \__tag_prop_gput:cnx
531         { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
532         { E }
533         { <\l__tag_tmpa_str> }
534     },
535 }

```

(End of definition for label (struct-key) and others. These functions are documented on page 90.)

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

This variable is used to number the AF-object names

```

536 \int_new:N\g__tag_struct_AFobj_int

\g__tag_struct_AFobj_int 537 \cs_if_free:NTF \pdffile_embed_stream:nnN
538 {
539     \cs_new_protected:Npn \__tag_struct_add_inline_AF:nn #1 #2
540     % #1 content, #2 extension
541     {
542         \group_begin:
543         \int_gincr:N \g__tag_struct_AFobj_int
544         \pdf_object_if_exist:eF {__tag/fileobj\int_use:N\g__tag_struct_AFobj_int}
545         {
546             \pdffile_embed_stream:nxx
547             {#1}
548             {tag-AFfile\int_use:N\g__tag_struct_AFobj_int.#2}
549             {__tag/fileobj\int_use:N\g__tag_struct_AFobj_int}
550             \__tag_struct_add_AF:ee
551             { \int_eval:n {\c@g__tag_struct_abs_int} }
552             { \pdf_object_ref:e {__tag/fileobj\int_use:N\g__tag_struct_AFobj_int} }
553             \__tag_prop_gput:cnx
554             { g__tag_struct_\int_use:N\c@g__tag_struct_abs_int _prop }
555             { AF }
556             {
557                 [
558                     \tl_use:c
559                     { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
560                 ]
561             }
562         }
563         \group_end:
564     }

```

```

565     }
566     {
567         \cs_generate_variant:Nn \pdffile_embed_stream:nnN {nxN}
568         \cs_new_protected:Npn \__tag_struct_add_inline_AF:nn #1 #2
569         % #1 content, #2 extension
570         {
571             \group_begin:
572             \int_gincr:N \g__tag_struct_AFobj_int
573             \pdffile_embed_stream:nxN
574             {#1}
575             {tag-AFfile\int_use:N\g__tag_struct_AFobj_int.#2}
576             \l__tag_tmpa_tl
577             \__tag_struct_add_AF:ee
578             { \int_eval:n {\c@g__tag_struct_abs_int} }
579             { \l__tag_tmpa_tl }
580             \__tag_prop_gput:cnx
581             { g__tag_struct_\int_use:N\c@g__tag_struct_abs_int _prop }
582             { AF }
583             {
584                 [
585                     \tl_use:c
586                     { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
587                 ]
588             }
589             \group_end:
590         }
591     }
592     \cs_generate_variant:Nn \__tag_struct_add_inline_AF:nn {on}
593     \cs_new_protected:Npn \__tag_struct_add_AF:nn #1 #2 % #1 struct num #2 object reference
594     {
595         \tl_if_exist:cTF
596         {
597             g__tag_struct_#1_AF_tl
598         }
599         {
600             \tl_gput_right:cx
601             { g__tag_struct_#1_AF_tl }
602             { \c_space_tl #2 }
603         }
604         {
605             \tl_new:c
606             { g__tag_struct_#1_AF_tl }
607             \tl_gset:cx
608             { g__tag_struct_#1_AF_tl }
609             { #2 }
610         }
611     }
612     \cs_generate_variant:Nn \__tag_struct_add_AF:nn {en,ee}
613     \keys_define:nn { __tag / struct }
614     {
615         AF .code:n          = % AF property
616         {
617             \pdf_object_if_exist:nTF {#1}
618             {

```

```

619     \_tag_struct_add_AF:ee { \int_eval:n {\c@g__tag_struct_abs_int} }\pdf_object_ref
620     \_tag_prop_gput:cnx
621     { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
622     { AF }
623     {
624     [
625         \tl_use:c
626         { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
627     ]
628     }
629     }
630     {
631     }
632     }
633     },
634     ,AFinline .code:n =
635     {
636         \_tag_struct_add_inline_AF:nn {#1}{txt}
637     }
638     ,AFinline-o .code:n =
639     {
640         \_tag_struct_add_inline_AF:on {#1}{txt}
641     }
642     ,texsource .code:n =
643     {
644         \group_begin:
645         \pdfdict_put:nnn { l_pdffile/Filespec }{AFRelationship} { /Source }

```

we set the mime type as pdfresources uses currently text/plain

```

646         \pdfdict_put:nnx
647         { l_pdffile }{Subtype}
648         { \pdf_name_from_unicode_e:n{application/x-tex} }
649         \_tag_struct_add_inline_AF:on {#1}{tex}
650         \group_end:
651     }
652 }

```

(End of definition for AF (struct-key) and others. These functions are documented on page 91.)

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!

```

653 \keys_define:nn { __tag / setup }
654 {
655     root-AF .code:n =
656     {
657         \pdf_object_if_exist:nTF {#1}
658         {
659             \_tag_struct_add_AF:ee { 0 }\pdf_object_ref:n {#1}
660             \_tag_prop_gput:cnx
661             { g__tag_struct_0_prop }
662             { AF }
663             {
664             [
665                 \tl_use:c

```

```

666         { g__tag_struct_0_AF_tl }
667     ]
668 }
669 }
670 {
671
672 }
673 },
674 }
675 </package>

```

(End of definition for root-AF (setup-key). This function is documented on page 92.)

6 User commands

```

\tag_struct_begin:n
\tag_struct_end: 666 <base>\cs_new_protected:Npn \tag_struct_begin:n #1 {\int_gincr:N \c@g__tag_struct_abs_int}
667 <base>\cs_new_protected:Npn \tag_struct_end: {}
668 <base>\cs_new_protected:Npn \tag_struct_end:n {}
669 *package| debug)
670 <package>\cs_set_protected:Npn \tag_struct_begin:n #1 %#1 key-val
671 <debug>\cs_set_protected:Npn \tag_struct_begin:n #1 %#1 key-val
672 {
673 <package>\__tag_check_if_active_struct:T
674 <debug>\__tag_check_if_active_struct:TF
675 {
676     \group_begin:
677     \int_gincr:N \c@g__tag_struct_abs_int
678     \__tag_prop_new:c { g__tag_struct\int_eval:n { \c@g__tag_struct_abs_int }_prop }
679     \__tag_new_output_prop_handler:n {\int_eval:n { \c@g__tag_struct_abs_int }}
680     \__tag_seq_new:c { g__tag_struct_kids\int_eval:n { \c@g__tag_struct_abs_int }_seq}
681     \exp_args:Ne
682     \pdf_object_new:n
683     { __tag/struct/\int_eval:n { \c@g__tag_struct_abs_int } }
684     \__tag_prop_gput:cno
685     { g__tag_struct\int_eval:n { \c@g__tag_struct_abs_int }_prop }
686     { Type }
687     { /StructElem }
688     \tl_set:Nn \l__tag_struct_stack_parent_tmpa_tl {-1}
689     \keys_set:nn { __tag / struct} { #1 }
690
691     \__tag_struct_set_tag_info:eVV
692     { \int_eval:n {\c@g__tag_struct_abs_int} }
693     \g__tag_struct_tag_tl
694     \g__tag_struct_tag_NS_tl
695     \__tag_check_structure_has_tag:n { \int_eval:n {\c@g__tag_struct_abs_int} }
696     \tl_if_empty:NF
697     \l__tag_struct_key_label_tl
698     {
699         \__tag_ref_label:en{tagpdfstruct-\l__tag_struct_key_label_tl}{struct}
700     }
701
702
703
704
705
706
707
708
709

```

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

```

710 \int_compare:nNnT { \l__tag_struct_stack_parent_tmpa_tl } = { -1 }
711 {
712   \seq_get:NNT
713     \g__tag_struct_stack_seq
714     \l__tag_struct_stack_parent_tmpa_tl
715     {
716       \msg_error:nn { tag } { struct-faulty-nesting }
717     }
718   }
719   \seq_gpush:NV \g__tag_struct_stack_seq \c@g__tag_struct_abs_int
720   \__tag_role_get:VVNN
721     \g__tag_struct_tag_tl
722     \g__tag_struct_tag_NS_tl
723     \l__tag_struct_roletag_tl
724     \l__tag_struct_roletag_NS_tl

```

to target role and role NS

```

725   \__tag_prop_gput:cnx
726     { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
727     { rolemap }
728     {
729       {\l__tag_struct_roletag_tl}{\l__tag_struct_roletag_NS_tl}
730     }

```

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

```

731   \str_case:VnTF \l__tag_struct_roletag_tl
732   {
733     {Part} {}
734     {Div} {}
735     {NonStruct} {}
736   }
737   {
738     \prop_get:cnNT
739     { g__tag_struct_ \l__tag_struct_stack_parent_tmpa_tl _prop }
740     { parentrole }
741     \l__tag_get_tmpc_tl
742     {
743       \__tag_prop_gput:cno
744       { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
745       { parentrole }
746       {
747         \l__tag_get_tmpc_tl
748       }
749     }
750   }
751   {
752     \__tag_prop_gput:cnx
753     { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
754     { parentrole }
755     {
756       {\l__tag_struct_roletag_tl}{\l__tag_struct_roletag_NS_tl}
757     }
758   }

```

```

759 \seq_gpush:Nx \g__tag_struct_tag_stack_seq
760   {\g__tag_struct_tag_tl}{\l__tag_struct_roletag_tl}
761 \tl_gset:NV \g__tag_struct_stack_current_tl \c@g__tag_struct_abs_int
762 %\seq_show:N \g__tag_struct_stack_seq
763 \bool_if:NF
764   \l__tag_struct_elem_stash_bool
765   {

```

check if the tag can be used inside the parent. It only makes sense, if the structure is actually used here, so it is guarded by the stash boolean. For now we ignore the namespace!

```

766   \__tag_struct_get_parentrole:eNN
767   {\l__tag_struct_stack_parent_tmpa_tl}
768   \l__tag_get_parent_tmpa_tl
769   \l__tag_get_parent_tmpb_tl
770   \__tag_check_parent_child:VVVVN
771   \l__tag_get_parent_tmpa_tl
772   \l__tag_get_parent_tmpb_tl
773   \g__tag_struct_tag_tl
774   \g__tag_struct_tag_NS_tl
775   \l__tag_parent_child_check_tl
776   \int_compare:nNnT {\l__tag_parent_child_check_tl}<0
777   {
778     \prop_get:cnN
779     { \g__tag_struct_ \l__tag_struct_stack_parent_tmpa_tl _prop }
780     {S}
781     \l__tag_tmpa_tl
782     \msg_warning:nnxxx
783     { tag }
784     {role-parent-child}
785     { \l__tag_get_parent_tmpa_tl/\l__tag_get_parent_tmpb_tl }
786     { \g__tag_struct_tag_tl/\g__tag_struct_tag_NS_tl }
787     { not-allowed~
788       (struct~\l__tag_struct_stack_parent_tmpa_tl,~\l__tag_tmpa_tl
789       \c_space_tl-->~struct~\int_eval:n {\c@g__tag_struct_abs_int})
790     }
791     \cs_set_eq:NN \l__tag_role_remap_tag_tl \g__tag_struct_tag_tl
792     \cs_set_eq:NN \l__tag_role_remap_NS_tl \g__tag_struct_tag_NS_tl
793     \__tag_role_remap:
794     \cs_gset_eq:NN \g__tag_struct_tag_tl \l__tag_role_remap_tag_tl
795     \cs_gset_eq:NN \g__tag_struct_tag_NS_tl \l__tag_role_remap_NS_tl
796     \__tag_struct_set_tag_info:eVV
797     { \int_eval:n {\c@g__tag_struct_abs_int} }
798     \g__tag_struct_tag_tl
799     \g__tag_struct_tag_NS_tl
800   }

```

Set the Parent.

```

801   \__tag_prop_gput:cnx
802   { \g__tag_struct_ \int_eval:n {\c@g__tag_struct_abs_int}_prop }
803   { P }
804   {
805     \pdf_object_ref:e { __tag/struct/\l__tag_struct_stack_parent_tmpa_tl }
806   }
807   %record this structure as kid:

```



```

808         %\tl_show:N \g__tag_struct_stack_current_tl
809         %\tl_show:N \l__tag_struct_stack_parent_tmpa_tl
810         \__tag_struct_kid_struct_gput_right:xx
811         { \l__tag_struct_stack_parent_tmpa_tl }
812         { \g__tag_struct_stack_current_tl }
813         %\prop_show:c { g__tag_struct_\g__tag_struct_stack_current_tl _prop }
814         %\seq_show:c {g__tag_struct_kids_\l__tag_struct_stack_parent_tmpa_tl _seq}
815     }
816     %\prop_show:c { g__tag_struct_\g__tag_struct_stack_current_tl _prop }
817     %\seq_show:c {g__tag_struct_kids_\l__tag_struct_stack_parent_tmpa_tl _seq}
818 <debug> \__tag_debug_struct_begin_insert:n { #1 }
819 \group_end:
820 }
821 <debug>{ \__tag_debug_struct_begin_ignore:n { #1 }}
822 }
823 <package>\cs_set_protected:Nn \tag_struct_end:
824 <debug>\cs_set_protected:Nn \tag_struct_end:
825 { %take the current structure num from the stack:
826   %the objects are written later, lua mode hasn't all needed info yet
827   %\seq_show:N \g__tag_struct_stack_seq
828 <package>\__tag_check_if_active_struct:T
829 <debug>\__tag_check_if_active_struct:TF
830 {
831     \seq_gpop:NN \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
832     \seq_gpop:NNTF \g__tag_struct_stack_seq \l__tag_tmpa_tl
833     {
834         \__tag_check_info_closing_struct:o { \g__tag_struct_stack_current_tl }
835     }
836     { \__tag_check_no_open_struct: }
837     % get the previous one, shouldn't be empty as the root should be there
838     \seq_get:NNTF \g__tag_struct_stack_seq \l__tag_tmpa_tl
839     {
840         \tl_gset:NV \g__tag_struct_stack_current_tl \l__tag_tmpa_tl
841     }
842     {
843         \__tag_check_no_open_struct:
844     }
845     \seq_get:NNT \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
846     {
847         \tl_gset:Nx \g__tag_struct_tag_tl
848         { \exp_last_unbraced:NV\use_i:nn \l__tag_tmpa_tl }
849         \prop_get:NVNT\g__tag_role_tags_NS_prop \g__tag_struct_tag_tl\l__tag_tmpa_tl
850         {
851             \tl_gset:Nx \g__tag_struct_tag_NS_tl { \l__tag_tmpa_tl }
852         }
853     }
854 <debug>\__tag_debug_struct_end_insert:
855 }
856 <debug>{ \__tag_debug_struct_end_ignore:}
857 }
858
859 \cs_set_protected:Npn \tag_struct_end:n #1
860 {
861 <debug>\__tag_debug_struct_end_check:n{#1}

```

```

862   \tag_struct_end:
863 }
864 </package|debug>

```

(End of definition for \tag_struct_begin:n and \tag_struct_end:. These functions are documented on page 89.)

\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.

```

865 <base>\cs_new_protected:Npn \tag_struct_use:n #1 {}
866 *package)
867 \cs_set_protected:Npn \tag_struct_use:n #1 % #1 is the label
868 {
869   \__tag_check_if_active_struct:T
870   {
871     \prop_if_exist:cTF
872     { g__tag_struct_\__tag_ref_value:enn{tagpdfstruct-#1}{tagstruct}{unknown}_prop } %
873     {
874       \__tag_check_struct_used:n {#1}
875       %add the label structure as kid to the current structure (can be the root)
876       \__tag_struct_kid_struct_gput_right:xx
877       { \g__tag_struct_stack_current_tl }
878       { \__tag_ref_value:enn{tagpdfstruct-#1}{tagstruct}{0} }
879       %add the current structure to the labeled one as parents
880       \__tag_prop_gput:cnx
881       { g__tag_struct_\__tag_ref_value:enn{tagpdfstruct-#1}{tagstruct}{0}_prop }
882       { P }
883       {
884         \pdf_object_ref:e { __tag/struct/\g__tag_struct_stack_current_tl }
885       }
886     }
887   }
888 }

```

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

```

886   \__tag_struct_get_parentrole:eNN
887   {\__tag_ref_value:enn{tagpdfstruct-#1}{tagstruct}{0}}
888   \l__tag_tmpa_tl
889   \l__tag_tmpb_tl
890   \__tag_check_parent_child:VVVVN
891   \g__tag_struct_tag_tl
892   \g__tag_struct_tag_NS_tl
893   \l__tag_tmpa_tl
894   \l__tag_tmpb_tl
895   \l__tag_parent_child_check_tl
896   \int_compare:nNnT {\l__tag_parent_child_check_tl}<0
897   {
898     \cs_set_eq:NN \l__tag_role_remap_tag_tl \g__tag_struct_tag_tl
899     \cs_set_eq:NN \l__tag_role_remap_NS_tl \g__tag_struct_tag_NS_tl
900     \__tag_role_remap:
901     \cs_gset_eq:NN \g__tag_struct_tag_tl \l__tag_role_remap_tag_tl
902     \cs_gset_eq:NN \g__tag_struct_tag_NS_tl \l__tag_role_remap_NS_tl
903     \__tag_struct_set_tag_info:eVV
904     { \int_eval:n {\c@g__tag_struct_abs_int} }
905     \g__tag_struct_tag_tl
906     \g__tag_struct_tag_NS_tl
907   }

```

```

908     }
909     {
910         \msg_warning:nnn{ tag }{struct-label-unknown}{#1}
911     }
912 }
913 }
914 \end{package}

```

(End of definition for \tag_struct_use:n. This function is documented on page 89.)

\tag_struct_use_num:n This command allows to use a stashed structure in another place. differently to the previous command it doesn't use a label but directly a structure number to find the parent. TODO: decide how it should be guarded. Probably by the struct-check.

```

915 \base\cs_new_protected:Npn \tag_struct_use_num:n #1 {}
916 \*package
917 \cs_set_protected:Npn \tag_struct_use_num:n #1 {%#1 is structure number
918 {
919     \__tag_check_if_active_struct:T
920     {
921         \prop_if_exist:cTF
922         { g__tag_struct_#1_prop } %
923         {
924             \prop_get:cnNT
925             {g__tag_struct_#1_prop}
926             {P}
927             \l__tag_tmpa_tl
928             {
929                 \msg_warning:nnn { tag } {struct-used-twice} {#1}
930             }
931             %add the label structure as kid to the current structure (can be the root)
932             \__tag_struct_kid_struct_gput_right:xx
933             { \g__tag_struct_stack_current_tl }
934             { #1 }
935             %add the current structure to the labeled one as parents
936             \__tag_prop_gput:cnx
937             { g__tag_struct_#1_prop }
938             { P }
939             {
940                 \pdf_object_ref:e { __tag/struct/\g__tag_struct_stack_current_tl }
941             }

```

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

```

942         \__tag_struct_get_parentrole:eNN
943         {#1}
944         \l__tag_tmpa_tl
945         \l__tag_tmpb_tl
946         \__tag_check_parent_child:VVVVN
947         \g__tag_struct_tag_tl
948         \g__tag_struct_tag_NS_tl
949         \l__tag_tmpa_tl
950         \l__tag_tmpb_tl
951         \l__tag_parent_child_check_tl
952         \int_compare:nNnT {\l__tag_parent_child_check_tl}<0

```

```

953         {
954             \cs_set_eq:NN \l__tag_role_remap_tag_tl \g__tag_struct_tag_tl
955             \cs_set_eq:NN \l__tag_role_remap_NS_tl \g__tag_struct_tag_NS_tl
956             \__tag_role_remap:
957             \cs_gset_eq:NN \g__tag_struct_tag_tl \l__tag_role_remap_tag_tl
958             \cs_gset_eq:NN \g__tag_struct_tag_NS_tl \l__tag_role_remap_NS_tl
959             \__tag_struct_set_tag_info:eVV
960             { \int_eval:n {\c@g__tag_struct_abs_int} }
961             \g__tag_struct_tag_tl
962             \g__tag_struct_tag_NS_tl
963         }
964     }
965     {
966         \msg_warning:nnn{ tag }{struct-label-unknown}{#1}
967     }
968 }
969 }
970 \end{package}

```

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

\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.

```

971 \begin{package}
972 \cs_new:Npn \tag_struct_object_ref:n #1
973 {
974     \pdf_object_ref:n {__tag/struct/#1}
975 }
976 \cs_generate_variant:Nn \tag_struct_object_ref:n {e}

```

(End of definition for \tag_struct_object_ref:n. This function is documented on page 89.)

\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

```

977 \cs_new_protected:Npn \tag_struct_gput:nnn #1 #2 #3
978 {
979     \cs_if_exist_use:cF {__tag_struct_gput_data_#2:nn}
980     { %warning??
981         \use_none:nn
982     }
983     {#1}{#3}
984 }
985 \cs_generate_variant:Nn \tag_struct_gput:nnn {ene,nne}

```

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

__tag_struct_gput_data_ref:nn

```

986 \cs_new_protected:Npn \__tag_struct_gput_data_ref:nn #1 #2
987 % #1 receiving struct num, #2 list of object ref
988 {
989     \prop_get:cnN
990     { g__tag_struct_#1_prop }

```

```

991     {Ref}
992     \l__tag_get_tmpc_tl
993     \__tag_prop_gput:cnx
994     { g__tag_struct_#1_prop }
995     { Ref }
996     { \quark_if_no_value:NF\l__tag_get_tmpc_tl { \l__tag_get_tmpc_tl\c_space_tl }#2 }
997   }
998   \cs_generate_variant:Nn \__tag_struct_gput_data_ref:nn {ee}

```

(End of 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.

```

999   \cs_new_protected:Npn \tag_struct_insert_annot:nn #1 #2 %#1 should be an object reference
1000                                     %#2 struct parent num
1001   {
1002     \__tag_check_if_active_struct:T
1003     {
1004       \__tag_struct_insert_annot:nn {#1}{#2}
1005     }
1006   }
1007
1008   \cs_generate_variant:Nn \tag_struct_insert_annot:nn {xx}
1009   \cs_new:Npn \tag_struct_parent_int: {\int_use:c { c@g__tag_parenttree_obj_int }}
1010
1011   \</package>
1012

```

(End of definition for \tag_struct_insert_annot:nn and \tag_struct_parent_int:. These functions are documented on page 89.)

7 Attributes and attribute classes

```

1013 \<header>
1014 \ProvidesExplPackage {tagpdf-attr-code} {2023-06-14} {0.98i}
1015   {part of tagpdf - code related to attributes and attribute classes}
1016 \</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_@@_attr_objref_prop

```

1017 \<package>
1018 \prop_new:N \g__tag_attr_entries_prop
1019 \seq_new:N \g__tag_attr_class_used_seq
1020 \tl_new:N \l__tag_attr_value_tl
1021 \prop_new:N \g__tag_attr_objref_prop %will contain obj num of used attributes

```

(End of definition for \g__tag_attr_entries_prop and others.)

7.2 Commands and keys

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

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

1022 \cs_new_protected:Npn \__tag_attr_new_entry:nn #1 #2 {%#1:name, #2: content
1023   {
1024     \prop_gput:Nen \g__tag_attr_entries_prop
1025       {\pdf_name_from_unicode_e:n{#1}}{#2}
1026   }
1027
1028 \keys_define:nn { __tag / setup }
1029 {
1030   newattribute .code:n =
1031   {
1032     \__tag_attr_new_entry:nn #1
1033   }
1034 }
```

(End of definition for `__tag_attr_new_entry:nn` and `newattribute (setup-key)`. This function is documented on page 92.)

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

```
1035 \keys_define:nn { __tag / struct }
1036 {
1037   attribute-class .code:n =
1038   {
1039     \clist_set:Nx \l__tag_tmpa_clist { #1 }
1040     \seq_set_from_clist:NN \l__tag_tmpb_seq \l__tag_tmpa_clist
1041     we convert the names into pdf names with slash
1042     \seq_set_map_x:NNn \l__tag_tmpa_seq \l__tag_tmpb_seq
1043     {
1044       \pdf_name_from_unicode_e:n {##1}
1045     }
1046     \seq_map_inline:Nn \l__tag_tmpa_seq
1047     {
1048       \prop_if_in:NnF \g__tag_attr_entries_prop {##1}
1049       {
1050         \msg_error:nnn { tag } { attr-unknown } { ##1 }
1051       }
1052       \seq_gput_left:Nn \g__tag_attr_class_used_seq { ##1 }
1053     }
1054     \tl_set:Nx \l__tag_tmpa_tl
1055     {
```

```

1055         \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}
1056         \seq_use:Nn \l__tag_tmpa_seq { \c_space_tl }
1057         \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}
1058     }
1059     \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 0 }
1060     {
1061         \__tag_prop_gput:cnx
1062         { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
1063         { C }
1064         { \l__tag_tmpa_tl }
1065         %\prop_show:c { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
1066     }
1067 }
1068 }

```

(End of definition for attribute-class (struct-key). This function is documented on page 91.)

attribute_□(struct-key)

```

1069 \keys_define:nn { __tag / struct }
1070 {
1071     attribute .code:n = % A property (attribute, value currently a dictionary)
1072     {
1073         \clist_set:Nx          \l__tag_tmpa_clist { #1 }
1074         \clist_if_empty:NF \l__tag_tmpa_clist
1075         {
1076             \seq_set_from_clist:NN \l__tag_tmpb_seq \l__tag_tmpa_clist

```

we convert the names into pdf names with slash

```

1077             \seq_set_map_x:NNn \l__tag_tmpa_seq \l__tag_tmpb_seq
1078             {
1079                 \pdf_name_from_unicode_e:n {##1}
1080             }
1081             \tl_set:Nx \l__tag_attr_value_tl
1082             {
1083                 \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}%
1084             }
1085             \seq_map_inline:Nn \l__tag_tmpa_seq
1086             {
1087                 \prop_if_in:NnF \g__tag_attr_entries_prop {##1}
1088                 {
1089                     \msg_error:nnn { tag } { attr-unknown } { ##1 }
1090                 }
1091                 \prop_if_in:NnF \g__tag_attr_objref_prop {##1}
1092                 {%\prop_show:N \g__tag_attr_entries_prop
1093                 \pdf_object_unnamed_write:nx
1094                 { dict }
1095                 {
1096                     \prop_item:Nn \g__tag_attr_entries_prop {##1}
1097                 }
1098                 \prop_gput:Nnx \g__tag_attr_objref_prop {##1} {\pdf_object_ref_last:}
1099             }
1100             \tl_put_right:Nx \l__tag_attr_value_tl
1101             {
1102                 \c_space_tl
1103                 \prop_item:Nn \g__tag_attr_objref_prop {##1}

```

```

1104         }
1105     % \tl_show:N \l__tag_attr_value_tl
1106     }
1107     \tl_put_right:Nx \l__tag_attr_value_tl
1108     { %[
1109         \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{]}%
1110     }
1111     % \tl_show:N \l__tag_attr_value_tl
1112     \__tag_prop_gput:cnx
1113     { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
1114     { A }
1115     { \l__tag_attr_value_tl }
1116     }
1117 },
1118 }
1119 \</package>

```

(End of definition for attribute (struct-key). This function is documented on page 91.)

Part VIII

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

```
1 <@@=tag>
2 <*luatex>
3 \ProvidesExplFile {tagpdf-luatex.def} {2023-06-14} {0.98i}
4 {tagpdf~driver~for~luatex}
```

1 Loading the lua

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

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

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

```

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

```

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

```

(End of 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   = "0.98i",      --TAGVERSION
70   date      = "2023-06-14", --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 table
168                                     -- used for pure lua tables too now!
169 ltx.__tag.page = ltx.__tag.page or { } -- page data, currently only i->{0->mcnum,1->mcnum}
170 ltx.__tag.trace = ltx.__tag.trace or { } -- show commands
```

```

171 ltx.__tag.func      = ltx.__tag.func    or { } -- functions
172 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.

```

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

```

(End of definition for `__tag_log` and `ltx.__tag.trace.log`.)

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

```

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

```

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

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

```

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

```

```

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

```

(End of definition for __tag_pairs_prop and ltx.__tag.trace.show_prop.)

`ltx.__tag.trace.show_mc_data` This shows some data for a mc given by num. If something is shown depends on the log level. The function is used by the following function and then in \ShowTagging

```

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

```

(End of definition for ltx.__tag.trace.show_mc_data.)

`ltx.__tag.trace.show_all_mc_data` This shows data for the mc's between min and max (numbers). It is used by the \ShowTagging function.

```

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

```

(End of definition for ltx.__tag.trace.show_all_mc_data.)

`ltx.__tag.trace.show_struct_data` This function shows some struct data. Unused but kept for debugging.

```

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

```

(End of definition for ltx.__tag.trace.show_struct_data.)

3 Helper functions

3.1 Retrieve data functions

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

```

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

```

(End of definition for `--tag_get_mc_cnt_type_tag`.)

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

```

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

```

(End of definition for `--tag_get_mathsubtype`.)

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

```

259 ltx.__tag.tables.role_tag_attribute = {}
260 ltx.__tag.tables.role_attribute_tag = {}

```

(End of definition for `ltx.__tag.tables.role_tag_attribute`.)

`ltx.__tag.func.alloctag`

```

261 local __tag_alloctag =
262   function (tag)
263     if not ltx.__tag.tables.role_tag_attribute[tag] then
264       table.insert(ltx.__tag.tables.role_attribute_tag,tag)
265       ltx.__tag.tables.role_tag_attribute[tag]=#ltx.__tag.tables.role_attribute_tag
266       __tag_log ("Add " ..tag.." " ..ltx.__tag.tables.role_tag_attribute[tag],3)
267     end
268   end
269 ltx.__tag.func.alloctag = __tag_alloctag

```

(End of definition for `ltx.__tag.func.alloctag`.)

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

`ltx.__tag.func.get_num_from`
`ltx.__tag.func.output_num_from`

```

270 local __tag_get_num_from =
271   function (tag)
272     if ltx.__tag.tables.role_tag_attribute[tag] then

```

```

273     a= ltx.__tag.tables.role_tag_attribute[tag]
274     else
275         a= -1
276     end
277     return a
278 end
279
280 ltx.__tag.func.get_num_from = __tag_get_num_from
281
282 function ltx.__tag.func.output_num_from (tag)
283     local num = __tag_get_num_from (tag)
284     tex.sprint(catlatex,num)
285     if num == -1 then
286         __tag_log ("Unknown tag "..tag.." used")
287     end
288 end

(End of definition for __tag_get_num_from, ltx.__tag.func.get_num_from, and ltx.__tag.func.output_
num_from.)

```

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

```

289 local __tag_get_tag_from =
290 function (num)
291     if ltx.__tag.tables.role_attribute_tag[num] then
292         a = ltx.__tag.tables.role_attribute_tag[num]
293     else
294         a= "UNKNOWN"
295     end
296     return a
297 end
298
299 ltx.__tag.func.get_tag_from = __tag_get_tag_from
300
301 function ltx.__tag.func.output_tag_from (num)
302     tex.sprint(catlatex,__tag_get_tag_from (num))
303 end

(End of definition for __tag_get_tag_from, ltx.__tag.func.get_tag_from, and ltx.__tag.func.output_
tag_from.)

```

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

```

304 function ltx.__tag.func.store_mc_data (num,key,data)
305     ltx.__tag.mc[num] = ltx.__tag.mc[num] or { }
306     ltx.__tag.mc[num][key] = data
307     __tag_log ("INFO TEX-STORE-MC-DATA: "..num.." => "..tostring(key).. " => "..tostring(data),3)
308 end

(End of definition for ltx.__tag.func.store_mc_data.)

```

ltx.__tag.func.store_mc_label This function stores the label=num relationship in the labels subtable. TODO: this is
 probably unused and can go.

```

309 function ltx.__tag.func.store_mc_label (label,num)

```



```

310 ltx.__tag.mc["labels"] = ltx.__tag.mc["labels"] or { }
311 ltx.__tag.mc.labels[label] = num
312 end

```

(End of definition for ltx.__tag.func.store_mc_label.)

`ltx.__tag.func.store_mc_kid` This function is used in the traversing code. It stores a sub-chunk of a mc mcnum into the kids table.

```

313 function ltx.__tag.func.store_mc_kid (mcnum,kid,page)
314 ltx.__tag.trace.log("INFO TAG-STORE-MC-KID: "..mcnum.." => " .. kid.." on page " .. page,3)
315 ltx.__tag.mc[mcnum]["kids"] = ltx.__tag.mc[mcnum]["kids"] or { }
316 local kidtable = {kid=kid,page=page}
317 tableinsert(ltx.__tag.mc[mcnum]["kids"], kidtable )
318 end

```

(End of definition for ltx.__tag.func.store_mc_kid.)

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

```

319 function ltx.__tag.func.mc_num_of_kids (mcnum)
320 local num = 0
321 if ltx.__tag.mc[mcnum] and ltx.__tag.mc[mcnum]["kids"] then
322     num = #ltx.__tag.mc[mcnum]["kids"]
323 end
324 ltx.__tag.trace.log ("INFO MC-KID-NUMBERS: " .. mcnum .. "has " .. num .. "KIDS",4)
325 return num
326 end

```

(End of 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.

```

327 local function __tag_insert_emc_node (head,current)
328 local emcnode = nodenew("whatsit","pdf_literal")
329     emcnode.data = "EMC"
330     emcnode.mode=1
331     head = node.insert_before(head,current,emcnode)
332 return head
333 end

```

(End of definition for __tag_insert_emc_node.)

`__tag_insert_bmc_node` This inserts a simple bmc node

```

334 local function __tag_insert_bmc_node (head,current,tag)
335 local bmcnode = nodenew("whatsit","pdf_literal")
336     bmcnode.data = "/"..tag.." BMC"
337     bmcnode.mode=1
338     head = node.insert_before(head,current,bmcnode)
339 return head
340 end

```

(End of 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.

```

341 local function __tag_insert_bdc_node (head,current,tag,dict)
342   local bdcnode = nodenew("whatsit","pdf_literal")
343   bdcnode.data = "/"..tag.."<<"..dict..">> BDC"
344   bdcnode.mode=1
345   head = node.insert_before(head,current,bdcnode)
346   return head
347 end

```

(End of 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

`ltx.__tag.func.pdf_object_ref`

```

348 local function __tag_pdf_object_ref (name)
349   local tokenname = 'c__pdf_backend_object_'..name..'__int'
350   local object = token.create(tokenname).index..' 0 R'
351   return object
352 end
353 ltx.__tag.func.pdf_object_ref=__tag_pdf_object_ref

```

(End of 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.

```

354 local function __tag_show_spacemark (head,current,color,height)
355   local markcolor = color or "1 0 0"
356   local markheight = height or 10
357   local pdfstring = node.new("whatsit","pdf_literal")
358   pdfstring.data =
359     string.format("q "..markcolor.." RG "..markcolor.." rg 0.4 w 0 %g m 0 %g l S Q",-
360       3,markheight)
361   head = node.insert_after(head,current,pdfstring)
362   return head
363 end

```

(End of definition for __tag_show_spacemark.)

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

`ltx.__tag.func.fakespace`

```

363 local function __tag_fakespace()
364   tex.setattribute(iwspaceattributeid,1)
365   tex.setattribute(iwfontattributeid,font.current())
366 end
367 ltx.__tag.func.fakespace = __tag_fakespace

```

(End of definition for __tag_fakespace and ltx.__tag.func.fakespace.)

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

```

368 --[[ a function to mark up places where real space chars should be inserted
369      it only sets an attribute.
370 --]]
371
372 local function __tag_mark_spaces (head)
373     local inside_math = false
374     for n in nodetraverse(head) do
375         local id = n.id
376         if id == GLYPH then
377             local glyph = n
378             if glyph.next and (glyph.next.id == GLUE)
379                 and not inside_math and (glyph.next.width > 0)
380             then
381                 nodesetattribute(glyph.next,iwspaceattributeid,1)
382                 nodesetattribute(glyph.next,iwfontattributeid,glyph.font)
383                 -- for debugging
384                 if ltx.__tag.trace.showspace then
385                     __tag_show_spacemark (head,glyph)
386                 end
387             elseif glyph.next and (glyph.next.id==KERN) and not inside_math then
388                 local kern = glyph.next
389                 if kern.next and (kern.next.id== GLUE) and (kern.next.width > 0)
390                 then
391                     nodesetattribute(kern.next,iwspaceattributeid,1)
392                     nodesetattribute(kern.next,iwfontattributeid,glyph.font)
393                 end
394             end
395             -- look also back
396             if glyph.prev and (glyph.prev.id == GLUE)
397                 and not inside_math
398                 and (glyph.prev.width > 0)
399                 and not nodehasattribute(glyph.prev,iwspaceattributeid)
400             then
401                 nodesetattribute(glyph.prev,iwspaceattributeid,1)
402                 nodesetattribute(glyph.prev,iwfontattributeid,glyph.font)
403                 -- for debugging
404                 if ltx.__tag.trace.showspace then
405                     __tag_show_spacemark (head,glyph)
406                 end
407             end
408             elseif id == PENALTY then
409                 local glyph = n
410                 -- ltx.__tag.trace.log ("PENALTY " .. n.subtype .. "VALUE" .. n.penalty,3)
411                 if glyph.next and (glyph.next.id == GLUE)
412                     and not inside_math and (glyph.next.width > 0) and n.subtype==0
413                 then
414                     nodesetattribute(glyph.next,iwspaceattributeid,1)
415                     -- nodesetattribute(glyph.next,iwfontattributeid,glyph.font)
416                     -- for debugging
417                     if ltx.__tag.trace.showspace then
418                         __tag_show_spacemark (head,glyph)

```

```

419         end
420     end
421     elseif id == MATH then
422         inside_math = (n.subtype == 0)
423     end
424 end
425 return head
426 end

```

(End of definition for `__tag_mark_spaces`.)

`__tag_activate_mark_space` These functions add/remove the function which marks the spaces to the callbacks
`ltx.__tag.func.markspaceon` `pre_linebreak_filter` and `hpack_filter`
`ltx.__tag.func.markspaceoff`

```

427 local function __tag_activate_mark_space ()
428     if not luatexbase.in_callback ("pre_linebreak_filter","markspaces") then
429         luatexbase.add_to_callback("pre_linebreak_filter",__tag_mark_spaces,"markspaces")
430         luatexbase.add_to_callback("hpack_filter",__tag_mark_spaces,"markspaces")
431     end
432 end
433
434 ltx.__tag.func.markspaceon=__tag_activate_mark_space
435

```

```

436 local function __tag_deactivate_mark_space ()
437     if luatexbase.in_callback ("pre_linebreak_filter","markspaces") then
438         luatexbase.remove_from_callback("pre_linebreak_filter","markspaces")
439         luatexbase.remove_from_callback("hpack_filter","markspaces")
440     end
441 end
442

```

```

443 ltx.__tag.func.markspaceoff=__tag_deactivate_mark_space

```

(End of definition for `__tag_activate_mark_space`, `ltx.__tag.func.markspaceon`, and `ltx.__tag.func.markspaceoff`.)

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

```

444 local default_space_char = node.new(GLYPH)
445 local default_fontid     = font.id("TU/lmr/m/n/10")
446 default_space_char.char  = 32
447 default_space_char.font  = default_fontid

```

`__tag_space_chars_shipout` These is the main function to insert real space chars. It inserts a glyph before every glue
`ltx.__tag.func.space_chars_shipout` 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.

```

448 local function __tag_space_chars_shipout (box)
449     local head = box.head
450     if head then
451         for n in node.traverse(head) do
452             local spaceattr = node.getattribute(n,iwspaceattributeid) or -1
453             if n.id == HLIST then -- enter the hlist
454                 __tag_space_chars_shipout (n)
455             elseif n.id == VLIST then -- enter the vlist
456                 __tag_space_chars_shipout (n)
457             elseif n.id == GLUE then
458                 if ltx.__tag.trace.showspaces and spaceattr==1 then
459                     __tag_show_spacemark (head,n,"0 1 0")
460                 end

```

```

461     if spaceattr==1 then
462         local space
463         local space_char = node.copy(default_space_char)
464         local curfont = node.get_attribute(n, i, font_attribute_id)
465         ltx.__tag.trace.log("INFO SPACE-FUNCTION-FONT: ".. tostring(curfont), 3)
466         if curfont and luaotfload.aux.slot_of_name(curfont, "space") then
467             space_char.font = curfont
468         end
469         head, space = node.insert_before(head, n, space_char) --
470         n.width = n.width - space.width
471         space.attr = n.attr
472     end
473 end
474 end
475 box.head = head
476 end
477 end
478
479 function ltx.__tag.func.space_chars_shipout (box)
480     __tag_space_chars_shipout (box)
481 end

```

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

5 Function for the tagging

`ltx.__tag.func.mc_insert_kids` This is the main function to insert the K entry into a StructElem object. It is used in `tagpdf-mc-luacode` module. The `single` attribute allows to handle the case that a single mc on the tex side can have more than one kid after the processing here, and so we get the correct array/non array setup.

```

482 function ltx.__tag.func.mc_insert_kids (mcnum, single)
483     if ltx.__tag.mc[mcnum] then
484         ltx.__tag.trace.log("INFO TEX-MC-INSERT-KID-TEST: " .. mcnum, 4)
485         if ltx.__tag.mc[mcnum]["kids"] then
486             if #ltx.__tag.mc[mcnum]["kids"] > 1 and single==1 then
487                 tex.sprint("")
488             end
489             for i, kidstable in ipairs( ltx.__tag.mc[mcnum]["kids"] ) do
490                 local kidnum = kidstable["kid"]
491                 local kidpage = kidstable["page"]
492                 local kidpageobjnum = pdfpageref(kidpage)
493                 ltx.__tag.trace.log("INFO TEX-MC-INSERT-KID: " .. mcnum ..
494                     " insert KID " .. i ..
495                     " with num " .. kidnum ..
496                     " on page " .. kidpage.." / ".. kidpageobjnum, 3)
497                 tex.sprint(catlatex, "<</Type /MCR /Pg ".. kidpageobjnum .. " 0 R /MCID ".. kidnum .. ">> ")
498             end
499             if #ltx.__tag.mc[mcnum]["kids"] > 1 and single==1 then
500                 tex.sprint("")
501             end
502         else
503             -- this is typically not a problem, e.g. empty hbox in footer/header can
504             -- trigger this warning.

```

```

505     ltx.__tag.trace.log("WARN TEX-MC-INSERT-NO-KIDS: "..mcnum.." has no kids",2)
506     if single==1 then
507         tex.sprint("null")
508     end
509 end
510 else
511     ltx.__tag.trace.log("WARN TEX-MC-INSERT-MISSING: "..mcnum.." doesn't exist",0)
512 end
513 end

```

(End of definition for ltx.__tag.func.mc_insert_kids.)

ltx.__tag.func.store_struct_mcabs

This function is used in the tagpdf-mc-luacode. It store the absolute count of the mc into the current structure. This must be done ordered.

```

514 function ltx.__tag.func.store_struct_mcabs (structnum,mcnum)
515     ltx.__tag.struct[structnum]=ltx.__tag.struct[structnum] or { }
516     ltx.__tag.struct[structnum]["mc"]=ltx.__tag.struct[structnum]["mc"] or { }
517     -- a structure can contain more than on mc chunk, the content should be ordered
518     tableinsert(ltx.__tag.struct[structnum]["mc"],"mc",mcnum)
519     ltx.__tag.trace.log("INFO TEX-MC-INTO-STRUCT: "..
520         mcnum.." inserted in struct "..structnum,3)
521     -- but every mc can only be in one structure
522     ltx.__tag.mc[mcnum]= ltx.__tag.mc[mcnum] or { }
523     ltx.__tag.mc[mcnum]["parent"] = structnum
524 end
525

```

(End of definition for ltx.__tag.func.store_struct_mcabs.)

ltx.__tag.func.store_mc_in_page

This is used in the traversing code and stores the relation between abs count and page count.

```

526 -- pay attention: lua counts arrays from 1, tex pages from one
527 -- mcid and arrays in pdf count from 0.
528 function ltx.__tag.func.store_mc_in_page (mcnum,mcpagencnt,page)
529     ltx.__tag.page[page] = ltx.__tag.page[page] or {}
530     ltx.__tag.page[page][mcpagencnt] = mcnum
531     ltx.__tag.trace.log("INFO TAG-MC-INTO-PAGE: page " .. page ..
532         ": inserting MCID " .. mcpagencnt .. " => " .. mcnum,3)
533 end

```

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

ltx.__tag.func.update_mc_attributes

This updates the mc-attributes of a box. It should only be used on boxes which don't contain structure elements. The arguments are a box, the mc-num and the type (as a number)

```

534 local function __tag_update_mc_attributes (head,mcnum,type)
535     for n in node.traverse(head) do
536         node.set_attribute(n,mccntattributeid,mcnum)
537         node.set_attribute(n,mctypeattributeid,type)
538         if n.id == HLIST or n.id == VLIST then
539             __tag_update_mc_attributes (n.list,mcnum,type)
540         end
541     end
542     return head
543 end
544 ltx.__tag.func.update_mc_attributes = __tag_update_mc_attributes

```

(End of definition for ltx.__tag.func.update_mc_attributes.)

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

```
545 --[[
546     Now follows the core function
547     It wades through the shipout box and checks the attributes
548     ARGUMENTS
549     box: is a box,
550     mcpagecnt: num, the current page cnt of mc (should start at -1 in shipout box), needed for
551     mcntprev: num, the attribute cnt of the previous node/whatever - if different we have a c
552     mcopen: num, records if some bdc/emc is open
553     These arguments are only needed for log messages, if not present are replaces by fix stri
554     name: string to describe the box
555     mctypeprev: num, the type attribute of the previous node/whatever
556
557     there are lots of logging messages currently. Should be cleaned up in due course.
558     One should also find ways to make the function shorter.
559 --]]
560
561 function ltx.__tag.func.mark_page_elements (box,mcpagecnt,mcntprev,mcopen,name,mctypeprev)
562     local name = name or ("SOMEBBOX")
563     local mctypeprev = mctypeprev or -1
564     local abspage = status.total_pages + 1 -- the real counter is increased
565                                           -- inside the box so one off
566                                           -- if the callback is not used. (???)
567     ltx.__tag.trace.log ("INFO TAG-ABSPAGE: " .. abspage,3)
568     ltx.__tag.trace.log ("INFO TAG-ARGS: pagecnt".. mcpagecnt..
569                         " prev "..mcntprev ..
570                         " type prev "..mctypeprev,4)
571     ltx.__tag.trace.log ("INFO TAG-TRAVERSING-BOX: ".. tostring(name)..
572                         " TYPE ".. node.type(node.getid(box)),3)
573     local head = box.head -- ShipoutBox is a vlist?
574     if head then
575         mcnthead, mctypehead,taghead = __tag_get_mc_cnt_type_tag (head)
576         ltx.__tag.trace.log ("INFO TAG-HEAD: " ..
577                             node.type(node.getid(head))..
578                             " MC"..tostring(mcnthead)..
579                             " => TAG " .. tostring(mctypehead)..
580                             " => ".. tostring(taghead),3)
581     else
582         ltx.__tag.trace.log ("INFO TAG-NO-HEAD: head is "..
583                             tostring(head),3)
584     end
585     for n in node.traverse(head) do
586         local mcnt, mctype, tag = __tag_get_mc_cnt_type_tag (n)
587         local spaceattr = nodegetattribute(n,iwspaceattributeid) or -1
588         ltx.__tag.trace.log ("INFO TAG-NODE: "..
589                             node.type(node.getid(n))..
590                             " MC".. tostring(mcnt)..
591                             " => TAG " .. tostring(mctype)..
592                             " => " .. tostring(tag),3)
593         if n.id == HLIST
594         then -- enter the hlist
595             mcopen,mcpagecnt,mcntprev,mctypeprev=
```

```

596     ltx.__tag.func.mark_page_elements (n,mcpagecnt,mccntprev,mcopen,"INTERNAL HLIST",mctypeprev)
597 elseif n.id == VLIST then -- enter the vlist
598     mcopyen,mcpagecnt,mccntprev,mctypeprev=
599     ltx.__tag.func.mark_page_elements (n,mcpagecnt,mccntprev,mcopen,"INTERNAL VLIST",mctypeprev)
600 elseif n.id == GLUE and not n.leader then -- at glue real space chars are inserted, but the
601     -- been done if the previous shipout wandering, so here it
602 elseif n.id == LOCAL_PAR then -- local_par is ignored
603 elseif n.id == PENALTY then -- penalty is ignored
604 elseif n.id == KERN then -- kern is ignored
605     ltx.__tag.trace.log ("INFO TAG-KERN-SUBTYPE: "..
606     node.type(node.getid(n)).." " "..n.subtype,4)
607 else
608     -- math is currently only logged.
609     -- we could mark the whole as math
610     -- for inner processing the mlist_to_hlist callback is probably needed.
611 if n.id == MATH then
612     ltx.__tag.trace.log("INFO TAG-MATH-SUBTYPE: "..
613     node.type(node.getid(n)).." " ".__tag_get_mathsubtype(n),4)
614 end
615 -- endmath
616 ltx.__tag.trace.log("INFO TAG-MC-COMPARE: current "..
617     mccnt.." prev "..mccntprev,4)
618 if mccnt~=mccntprev then -- a new mc chunk
619     ltx.__tag.trace.log ("INFO TAG-NEW-MC-NODE: "..
620     node.type(node.getid(n))..
621     " MC"..tostring(mccnt)..
622     " <=> PREVIOUS "..tostring(mccntprev),4)
623 if mcopen~=0 then -- there is a chunk open, close it (hope there is only one ...
624     box.list=__tag_insert_emc_node (box.list,n)
625     mcopen = mcopen - 1
626     ltx.__tag.trace.log ("INFO TAG-INSERT-EMC: " ..
627     mcpagecnt .. " MCOPEN = " .. mcopen,3)
628     if mcopen ~=0 then
629         ltx.__tag.trace.log ("WARN TAG-OPEN-MC: " .. mcopen,1)
630     end
631 end
632 if ltx.__tag.mc[mccnt] then
633     if ltx.__tag.mc[mccnt]["artifact"] then
634         ltx.__tag.trace.log("INFO TAG-INSERT-ARTIFACT: "..
635         tostring(ltx.__tag.mc[mccnt]["artifact"]),3)
636     if ltx.__tag.mc[mccnt]["artifact"] == "" then
637         box.list = __tag_insert_bmc_node (box.list,n,"Artifact")
638     else
639         box.list = __tag_insert_bdc_node (box.list,n,"Artifact", "/"Type /"..ltx.__tag.mc[mccnt]
640     end
641 else
642     ltx.__tag.trace.log("INFO TAG-INSERT-TAG: "..
643     tostring(tag),3)
644     mcpagecnt = mcpagecnt +1
645     ltx.__tag.trace.log ("INFO TAG-INSERT-BDC: "..mcpagecnt,3)
646     local dict= "/MCID "..mcpagecnt
647     if ltx.__tag.mc[mccnt]["raw"] then
648         ltx.__tag.trace.log("INFO TAG-USE-RAW: "..
649         tostring(ltx.__tag.mc[mccnt]["raw"]),3)

```



```

650     dict= dict .. " " .. ltx.__tag.mc[mccnt]["raw"]
651 end
652 if ltx.__tag.mc[mccnt]["alt"] then
653     ltx.__tag.trace.log("INFO TAG-USE-ALT: "...
654         tostring(ltx.__tag.mc[mccnt]["alt"]),3)
655     dict= dict .. " " .. ltx.__tag.mc[mccnt]["alt"]
656 end
657 if ltx.__tag.mc[mccnt]["actualtext"] then
658     ltx.__tag.trace.log("INFO TAG-USE-ACTUALTEXT: "...
659         tostring(ltx.__tag.mc[mccnt]["actualtext"]),3)
660     dict= dict .. " " .. ltx.__tag.mc[mccnt]["actualtext"]
661 end
662 box.list = __tag_insert_bdc_node (box.list,n,tag, dict)
663 ltx.__tag.func.store_mc_kid (mccnt,mcpagecnt,abspage)
664 ltx.__tag.func.store_mc_in_page(mccnt,mcpagecnt,abspage)
665 ltx.__tag.trace.show_mc_data (mccnt,3)
666 end
667 mcopen = mcopen + 1
668 else
669     if tagunmarkedbool.mode == truebool.mode then
670         ltx.__tag.trace.log("INFO TAG-NOT-TAGGED: this has not been tagged, using artifact",2)
671         box.list = __tag_insert_bmc_node (box.list,n,"Artifact")
672         mcopen = mcopen + 1
673     else
674         ltx.__tag.trace.log("WARN TAG-NOT-TAGGED: this has not been tagged",1)
675     end
676 end
677 mccntprev = mccnt
678 end
679 end -- end if
680 end -- end for
681 if head then
682     mccnthead, mctypehead,taghead = __tag_get_mc_cnt_type_tag (head)
683     ltx.__tag.trace.log ("INFO TAG-ENDHEAD: " ..
684         node.type(node.getid(head))..
685         " MC"..tostring(mccnthead)..
686         " => TAG "..tostring(mctypehead)..
687         " => "..tostring(taghead),4)
688 else
689     ltx.__tag.trace.log ("INFO TAG-ENDHEAD: ".. tostring(head),4)
690 end
691 ltx.__tag.trace.log ("INFO TAG-QUITTING-BOX "...
692     tostring(name)..
693     " TYPE ".. node.type(node.getid(box)),4)
694 return mcopen,mcpagecnt,mccntprev,mctypeprev
695 end
696

```

(End of definition for ltx.__tag.func.mark_page_elements.)

ltx.__tag.func.mark_shipout This is the function used in the callback. Beside calling the traversing function it also checks if there is an open MC-chunk from a page break and insert the needed EMC literal.

```

697 function ltx.__tag.func.mark_shipout (box)

```

```

698 mcopen = ltx.__tag.func.mark_page_elements (box,-1,-100,0,"Shipout",-1)
699 if mcopen~=0 then -- there is a chunk open, close it (hope there is only one ...
700     local emcnode = nodenew("whatsit","pdf_literal")
701     local list = box.list
702     emcnode.data = "EMC"
703     emcnode.mode=1
704     if list then
705         list = node.insert_after (list,node.tail(list),emcnode)
706         mcopen = mcopen - 1
707         ltx.__tag.trace.log ("INFO SHIPOUT-INSERT-LAST-EMC: MCOPEN " .. mcopen,3)
708     else
709         ltx.__tag.trace.log ("WARN SHIPOUT-UPS: this shouldn't happen",0)
710     end
711     if mcopen ~=0 then
712         ltx.__tag.trace.log ("WARN SHIPOUT-MC-OPEN: " .. mcopen,1)
713     end
714 end
715 end

```

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

6 Parenttree

ltx.__tag.func.fill_parent_tree_line
ltx.__tag.func.output_parenttree

These functions create the parent tree. The second, main function is used in the tagpdf-tree code. TODO check if the tree code can move into the backend code.

```

716 function ltx.__tag.func.fill_parent_tree_line (page)
717     -- we need to get page-> i=kid -> mcnum -> structnum
718     -- pay attention: the kid numbers and the page number in the parent tree start with 0!
719     local numsentry = ""
720     local pdfpage = page-1
721     if ltx.__tag.page[page] and ltx.__tag.page[page][0] then
722         mcchunks=#ltx.__tag.page[page]
723         ltx.__tag.trace.log("INFO PARENTTREE-NUM: page "..
724             page.." has "..mcchunks.." +1 Elements ",4)
725         for i=0,mcchunks do
726             -- what does this log??
727             ltx.__tag.trace.log("INFO PARENTTREE-CHUNKS: "..
728                 ltx.__tag.page[page][i],4)
729         end
730         if mcchunks == 0 then
731             -- only one chunk so no need for an array
732             local mcnum = ltx.__tag.page[page][0]
733             local structnum = ltx.__tag.mc[mcnum]["parent"]
734             local propname = "g__tag_struct_"..structnum.."_prop"
735             --local objref = ltx.__tag.tables[propname]["objref"] or "XXXX"
736             local objref = __tag_pdf_object_ref('__tag/struct/'..structnum)
737             ltx.__tag.trace.log("INFO PARENTTREE-STRUCT-OBJREF: =====>"..
738                 tostring(objref),5)
739             numsentry = pdfpage .. " [".. objref .. "]"
740             ltx.__tag.trace.log("INFO PARENTTREE-NUMENTRY: page " ..
741                 page.. " num entry = ".. numsentry,3)
742         else
743             numsentry = pdfpage .. " ["

```

```

744     for i=0,mcchunks do
745         local mcnum = ltx.__tag.page[page][i]
746         local structnum = ltx.__tag.mc[mcnum]["parent"] or 0
747         local propname = "g__tag_struct"..structnum.."__prop"
748         --local objref = ltx.__tag.tables[propname]["objref"] or "XXXX"
749         local objref = __tag_pdf_object_ref('__tag/struct/'..structnum)
750         numsentry = numsentry .. " " .. objref
751     end
752     numsentry = numsentry .. "]" "
753     ltx.__tag.trace.log("INFO PARENTTREE-NUMENTRY: page " ..
754         page.. " num entry = ".. numsentry,3)
755 end
756 else
757     ltx.__tag.trace.log ("INFO PARENTTREE-NO-DATA: page "..page,3)
758 end
759 return numsentry
760 end
761
762 function ltx.__tag.func.output_parenttree (abspage)
763     for i=1,abspage do
764         line = ltx.__tag.func.fill_parent_tree_line (i) .. "^^J"
765         tex.sprint(catlatex,line)
766     end
767 end

```

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

```

768 </lua>

```

Part IX

The tagpdf-roles module

Tags, roles and namespace code

Part of the tagpdf package

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

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

The key-value list knows the following keys:

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

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

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

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

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

```
\tag_check_child:nnTF \tag_check_child:nn{<tag>}{<namespace>} {<true code>} {<false code>}
```

This checks if the tag `<tag>` from the name space `<namespace>` can be used at the current position. In tagpdf-base it is always true.

```
1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-roles-code} {2023-06-14} {0.98i}
4 {part of tagpdf - code related to roles and structure names}
5 </header>
```

1 Code related to roles and structure names

6 `\package`

1.1 Variables

Tags are used in structures (`\tagstructbegin`) and mc-chunks (`\tagmcbegin`).

They have a name (a string), in lua a number (for the lua attribute), and in PDF 2.0 belong to one or more name spaces, with one being the default name space.

Tags of structures are classified, e.g. as grouping, inline or block level structure (and a few special classes like lists and tables), and must follow containments rules depending on their classification (for example a inline structure can not contain a block level structure). New tags inherit their classification from their rolemapping to the standard namespaces (`pdf` and/or `pdf2`). We store this classification as it will probably be needed for tests but currently the data is not much used. The classification for math (and the containment rules) is unclear currently and so not set.

The attribute number is only relevant in lua and only for the MC chunks (so tags with the same name from different names spaces can have the same number), and so only stored if luatex is detected.

Due to the namespaces the storing and processing of tags and there data are different in various places for PDF 2.0 and PDF <2.0, which makes things a bit difficult and leads to some duplications. Perhaps at some time there should be a clear split.

This are the main variables used by the code:

`\g__tag_role_tags_NS_prop` This is the core list of tag names. It uses tags as keys and the shorthand (e.g. `pdf2`, or `mathml`) of the default name space as value.

In pdf 2.0 the value is needed in the structure dictionaries.

`\g__tag_role_tags_class_prop` This contains for each tag a classification type. It is used in pdf <2.0.

`\g__tag_role_NS_prop` This contains the names spaces. The values are the object references. They are used in pdf 2.0.

`\g__tag_role_rolemap_prop` This contains for each tag the role to a standard tag. It is used in pdf<2.0 for tag checking and to fill at the end the RoleMap dictionary.

`g_@@_role/RoleMap_dict` This dictionary contains the standard rolemaps. It is relevant only for pdf <2.0.

`\g__tag_role_NS_<ns>_prop` This prop contains the tags of a name space and their role. The props are also use for remapping. As value they contain two brace groups: tag and namespace. In pdf <2.0 the namespace is empty.

`\g__tag_role_NS_<ns>_class_prop` This prop contains the tags of a name space and their type. The value is only needed for pdf 2.0.

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

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

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

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

(End of definition for `\g__tag_role_tags_NS_prop`.)

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

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

(End of definition for `\g__tag_role_tags_class_prop`.)

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

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

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

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

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

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

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

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

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

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

(End of definition for `\g__tag_role_NS_prop`.)

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

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

(End of definition for `\g__tag_role_index_prop`.)

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

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

(End of definition for `\l__tag_role_debug_prop`.)

We need also a bunch of temporary variables.

```
\l__tag_role_tag_tmpa_tl
```

```
\l__tag_role_tag_namespace_tmpa_tl
```

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

```
\l__tag_role_role_tmpa_tl
```

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

```
\l__tag_role_role_namespace_tmpa_tl
```

```
14 \tl_new:N \l__tag_role_role_tmpa_tl
```

```
\l__tag_role_role_namespace_tmpa_tl
```

```
15 \tl_new:N \l__tag_role_role_namespace_tmpa_tl
```

```
\l__tag_role_role_tmpa_seq
```

```
16 \seq_new:N \l__tag_role_role_tmpa_seq
```

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

1.2 Namespaces

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

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

```
17 \pdfdict_new:n {g__tag_role/RoleMap_dict}
18 \prop_new:N \g__tag_role_rolemap_prop
```

(End of definition for `g__tag_role/RoleMap_dict` and `\g__tag_role_rolemap_prop`.)

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

```
\__tag_role_NS_new:nnn
```

```
19 \pdf_version_compare:NnTF < {2.0}
20 {
21   \cs_new_protected:Npn \__tag_role_NS_new:nnn #1 #2 #3
22   {
23     \prop_new:c { g__tag_role_NS_#1_prop }
24     \prop_new:c { g__tag_role_NS_#1_class_prop }
25     \prop_gput:Nnx \g__tag_role_NS_prop {#1}{ }
26   }
27 }
28 {
29   \cs_new_protected:Npn \__tag_role_NS_new:nnn #1 #2 #3
30   {
31     \prop_new:c { g__tag_role_NS_#1_prop }
32     \prop_new:c { g__tag_role_NS_#1_class_prop }
33     \pdf_object_new:n {tag/NS/#1}
34     \pdfdict_new:n {g__tag_role/Namespace_#1_dict}
35     \pdf_object_new:n {__tag/RoleMapNS/#1}
36     \pdfdict_new:n {g__tag_role/RoleMapNS_#1_dict}
37     \pdfdict_gput:nnn
38       {g__tag_role/Namespace_#1_dict}
39       {Type}
40       {/Namespace}
41     \pdf_string_from_unicode:nnN{utf8/string}{#2}\l__tag_tmpa_str
42     \tl_if_empty:NF \l__tag_tmpa_str
43     {
44       \pdfdict_gput:nnx
45         {g__tag_role/Namespace_#1_dict}
46         {NS}
47         {\l__tag_tmpa_str}
48     }
49     %RoleMapNS is added in tree
50     \tl_if_empty:NF {#3}
```

```

51     {
52       \pdfdict_gput:nnx{g__tag_role/Namespace_#1_dict}
53       {Schema}{#3}
54     }
55     \prop_gput:Nnx \g__tag_role_NS_prop {#1}{\pdf_object_ref:n{tag/NS/#1}~}
56   }
57 }

```

(End of definition for __tag_role_NS_new:nnn.)

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

\c__tag_role_userNS_id_str

```

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

```

(End of definition for \c__tag_role_userNS_id_str.)

Now we setup the standard names spaces. The mathml space is currently only loaded for pdf 2.0.

```

72 \__tag_role_NS_new:nnn {pdf} {http://iso.org/pdf/ssn}{ }
73 \__tag_role_NS_new:nnn {pdf2} {http://iso.org/pdf2/ssn}{ }
74 \pdf_version_compare:NnF < {2.0}
75 {
76   \__tag_role_NS_new:nnn {mathml}{http://www.w3.org/1998/Math/MathML}{ }
77 }
78 \__tag_role_NS_new:nnn {latex} {https://www.latex-project.org/ns/dfl/2022}{ }
79 \__tag_role_NS_new:nnn {latex-book} {https://www.latex-project.org/ns/book/2022}{ }
80 \__tag_role_NS_new:nnn {latex-inline} {https://www.latex-project.org/ns/inline/2022}{ }
81 \exp_args:Nnx
82   \__tag_role_NS_new:nnn {user}{\c__tag_role_userNS_id_str}{ }

```

1.3 Adding a new tag

Both when reading the files and when setting up a tag manually we have to store data in various places.

__tag_role_alloctag:nnn

This command allocates a new tag without role mapping. In the lua backend it will also record the attribute value.

```

83 \pdf_version_compare:NnTF < {2.0}
84 {

```



```

85 \sys_if_engine luatex:TF
86 {
87   \cs_new_protected:Npn \__tag_role_alloctag:nnn #1 #2 #3 % #1 tagname, ns, type
88   {
89     \lua_now:e { ltx.__tag.func.alloctag ('#1') }
90     \prop_gput:Nnn \g__tag_role_tags_NS_prop {#1}{#2}
91     \prop_gput:cnn {g__tag_role_NS_#2_prop} {#1}{{}}{}
92     \prop_gput:Nnn \g__tag_role_tags_class_prop {#1}{#3}
93     \prop_gput:cnn {g__tag_role_NS_#2_class_prop} {#1}{--UNUSED--}
94   }
95 }
96 {
97   \cs_new_protected:Npn \__tag_role_alloctag:nnn #1 #2 #3
98   {
99     \prop_gput:Nnn \g__tag_role_tags_NS_prop {#1}{#2}
100    \prop_gput:cnn {g__tag_role_NS_#2_prop} {#1}{{}}{}
101    \prop_gput:Nnn \g__tag_role_tags_class_prop {#1}{#3}
102    \prop_gput:cnn {g__tag_role_NS_#2_class_prop} {#1}{--UNUSED--}
103  }
104 }
105 }
106 {
107   \sys_if_engine luatex:TF
108   {
109     \cs_new_protected:Npn \__tag_role_alloctag:nnn #1 #2 #3 % #1 tagname, ns, type
110     {
111       \lua_now:e { ltx.__tag.func.alloctag ('#1') }
112       \prop_gput:Nnn \g__tag_role_tags_NS_prop {#1}{#2}
113       \prop_gput:cnn {g__tag_role_NS_#2_prop} {#1}{{}}{}
114       \prop_gput:Nnn \g__tag_role_tags_class_prop {#1}{--UNUSED--}
115       \prop_gput:cnn {g__tag_role_NS_#2_class_prop} {#1}{#3}
116     }
117   }
118   {
119     \cs_new_protected:Npn \__tag_role_alloctag:nnn #1 #2 #3
120     {
121       \prop_gput:Nnn \g__tag_role_tags_NS_prop {#1}{#2}
122       \prop_gput:cnn {g__tag_role_NS_#2_prop} {#1}{{}}{}
123       \prop_gput:Nnn \g__tag_role_tags_class_prop {#1}{--UNUSED--}
124       \prop_gput:cnn {g__tag_role_NS_#2_class_prop} {#1}{#3}
125     }
126   }
127 }
128 \cs_generate_variant:Nn \__tag_role_alloctag:nnn {nnV}

```

(End of definition for __tag_role_alloctag:nnn.)

1.3.1 pdf 1.7 and earlier

`__tag_role_add_tag:nn` The pdf 1.7 version has only two arguments: new and rolemap name. The role must be an existing tag and should not be empty. We allow to change the role of an existing tag: as the rolemap is written at the end not confusion can happen.

```

129 \cs_new_protected:Nn \__tag_role_add_tag:nn % (new) name, reference to old
130 {

```

checks and messages

```

131   \__tag_check_add_tag_role:nn {#1}{#2}
132   \prop_if_in:NnF \g__tag_role_tags_NS_prop {#1}
133   {
134     \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
135     {
136       \msg_info:nnn { tag }{new-tag}{#1}
137     }
138   }

```

now the addition

```

139   \prop_get:NnN \g__tag_role_tags_class_prop {#2}\l__tag_tmpa_tl
140   \quark_if_no_value:NT \l__tag_tmpa_tl
141   {
142     \tl_set:Nn\l__tag_tmpa_tl{--UNKNOWN--}
143   }
144   \__tag_role_alloctag:nnV {#1}{user}\l__tag_tmpa_tl

```

We resolve rolemapping recursively so that all targets are stored as standard tags.

```

145   \tl_if_empty:nF { #2 }
146   {
147     \prop_get:NnN \g__tag_role_rolemap_prop {#2}\l__tag_tmpa_tl
148     \quark_if_no_value:NTF \l__tag_tmpa_tl
149     {
150       \prop_gput:Nnx \g__tag_role_rolemap_prop {#1}{\tl_to_str:n{#2}}
151     }
152     {
153       \prop_gput:NnV \g__tag_role_rolemap_prop {#1}\l__tag_tmpa_tl
154     }
155   }
156 }
157 \cs_generate_variant:Nn \__tag_role_add_tag:nn {VV,ne}

```

(End of definition for __tag_role_add_tag:nn.)

For the parent-child test we must be able to get the role. We use the same number of arguments as for the 2.0 command. If there is no role, we assume a standard tag.

__tag_role_get:nnNN

```

158 \pdf_version_compare:NnT < {2.0}
159 {
160   \cs_new:Npn \__tag_role_get:nnNN #1#2#3#4 {%#1 tag, #2 NS, #3 tlvar which hold the role tag
161   {
162     \prop_get:NnNF \g__tag_role_rolemap_prop {#1}#3
163     {
164       \tl_set:Nn #3 {#1}
165     }
166     \tl_set:Nn #4 {}
167   }
168   \cs_generate_variant:Nn \__tag_role_get:nnNN {VVNN}
169 }
170

```

(End of definition for __tag_role_get:nnNN.)

1.3.2 The pdf 2.0 version

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

```

171 \cs_new_protected:Nn \__tag_role_add_tag:nnnn %tag/namespace/role/namespace
172 {
173   \__tag_check_add_tag_role:nnn {#1/#2}{#3}{#4}
174   \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
175   {
176     \msg_info:nnn { tag }{new-tag}{#1}
177   }
178   \prop_get:cnN { g__tag_role_NS_#4_class_prop } {#3}\l__tag_tmpa_tl
179   \quark_if_no_value:NT \l__tag_tmpa_tl
180   {
181     \tl_set:Nn\l__tag_tmpa_tl{--UNKNOWN--}
182   }
183   \__tag_role_alloctag:nnV {#1}{#2}\l__tag_tmpa_tl

```

Do not remap standard tags. TODO add warning?

```

184   \tl_if_in:nnF {-pdf-pdf2-mathml-}{-#2-}
185   {
186     \pdfdict_gput:nnx {g__tag_role/RoleMapNS_#2_dict}{#1}
187     {
188       [
189         \pdf_name_from_unicode_e:n{#3}
190         \c_space_tl
191         \pdf_object_ref:n {tag/NS/#4}
192       ]
193     }
194   }

```

We resolve rolemapping recursively so that all targets are stored as standard tags for the tests.

```

195   \tl_if_empty:nF { #2 }
196   {
197     \prop_get:cnN { g__tag_role_NS_#4_prop } {#3}\l__tag_tmpa_tl
198     \quark_if_no_value:NTF \l__tag_tmpa_tl
199     {
200       \prop_gput:cnx { g__tag_role_NS_#2_prop } {#1}
201       {\tl_to_str:n{#3}}{\tl_to_str:n{#4}}
202     }
203     {
204       \prop_gput:cno { g__tag_role_NS_#2_prop } {#1}{\l__tag_tmpa_tl}
205     }
206   }
207 }
208 \cs_generate_variant:Nn \__tag_role_add_tag:nnnn {VVVV}

```

(End of definition for `__tag_role_add_tag:nnnn`.)

For the parent-child test we must be able to get the role. We use the same number of arguments as for the <2.0 command (and assume that we don't need a name space)

```

\__tag_role_get:nnNN
209 \pdf_version_compare:NnF < {2.0}
210 {
211   \cs_new:Npn \__tag_role_get:nnNN #1#2#3#4

```

```

212     %1 tag, #2 NS,
213     %3 tlvar which hold the role tag
214     %4 tlvar which hold the name of the target NS
215   {
216     \prop_get:cnNTF {g__tag_role_NS_#2_prop} {#1}\l__tag_tmpa_tl
217     {
218       \tl_set:Nx #3 {\exp_last_unbraced:NV\use_i:nn \l__tag_tmpa_tl}
219       \tl_set:Nx #4 {\exp_last_unbraced:NV\use_ii:nn \l__tag_tmpa_tl}
220     }
221     {
222       \tl_set:Nn #3 {#1}
223       \tl_set:Nn #4 {#2}
224     }
225   }
226   \cs_generate_variant:Nn \__tag_role_get:nnNN {VVNN}
227 }

```

(End of definition for __tag_role_get:nnNN.)

1.4 Helper command to read the data from files

In this section we setup the helper command to read namespace files.

__tag_role_read_namespace_line:nw

This command will process a line in the name space file. The first argument is the name of the name space. The definition differ for pdf 2.0. as we have proper name spaces there. With pdf<2.0 not special name spaces shouldn't update the default role or add to the rolemap again. We use a boolean here.

```

228 \bool_new:N\l__tag_role_update_bool
229 \bool_set_true:N \l__tag_role_update_bool
230 \pdf_version_compare:NnTF < {2.0}
231 {
232   \cs_new_protected:Npn \__tag_role_read_namespace_line:nw #1#2,#3,#4,#5,#6\q_stop %
233   % #1 NS, #2 tag, #3 rolemap, #4 NS rolemap #5 type
234   {
235     \tl_if_empty:nF { #2 }
236     {
237       \bool_if:NTF \l__tag_role_update_bool
238       {
239         \tl_if_empty:nTF {#5}
240         {
241           \prop_get:NnN \g__tag_role_tags_class_prop {#3}\l__tag_tmpa_tl
242           \quark_if_no_value:NT \l__tag_tmpa_tl
243           {
244             \tl_set:Nn\l__tag_tmpa_tl{--UNKNOWN--}
245           }
246         }
247         {
248           \tl_set:Nn \l__tag_tmpa_tl {#5}
249         }
250       }
251       \__tag_role_alloctag:nnV {#2}{#1}\l__tag_tmpa_tl
252       \tl_if_eq:nnF {#2}{#3}
253       {
254         \__tag_role_add_tag:nn {#2}{#3}
255       }

```

```

255     \prop_gput:cnn {g__tag_role_NS_#1_prop} {#2}{#{3}{}}
256   }
257   {
258     \prop_gput:cnn {g__tag_role_NS_#1_prop} {#2}{#{3}{}}
259     \prop_gput:cnn {g__tag_role_NS_#1_class_prop} {#2}{--UNUSED--}
260   }
261 }
262 }
263 }
264 {
265   \cs_new_protected:Npn \__tag_role_read_namespace_line:nw #1#2,#3,#4,#5,#6\q_stop %
266   % #1 NS, #2 tag, #3 rolemap, #4 NS rolemap #5 type
267   {
268     \tl_if_empty:nF {#2}
269     {
270       \tl_if_empty:nTF {#5}
271       {
272         \prop_get:cnN { g__tag_role_NS_#4_class_prop } {#3}\l__tag_tmpa_tl
273         \quark_if_no_value:NT \l__tag_tmpa_tl
274         {
275           \tl_set:Nn\l__tag_tmpa_tl{--UNKNOWN--}
276         }
277       }
278       {
279         \tl_set:Nn \l__tag_tmpa_tl {#5}
280       }
281       \__tag_role_alloctag:nnV {#2}{#1}\l__tag_tmpa_tl
282       \bool_lazy_and:nnT
283       { ! \tl_if_empty_p:n {#3} }{! \str_if_eq_p:nn {#1}{pdf2}}
284       {
285         \__tag_role_add_tag:nnnn {#2}{#1}{#3}{#4}
286       }
287       \prop_gput:cnn {g__tag_role_NS_#1_prop} {#2}{#{3}{#4}}
288     }
289   }
290 }

```

(End of definition for __tag_role_read_namespace_line:nw.)

__tag_role_read_namespace:n This command reads the namespace file.

```

291 \cs_new_protected:Npn \__tag_role_read_namespace:n #1 %name of namespace
292 {
293   \prop_if_exist:cF {g__tag_role_NS_#1_prop}
294   { \msg_warning:nnn {tag}{namespace-unknown}{#1} }
295   \file_if_exist:nTF { tagpdf-ns-#1.def}
296   {
297     \ior_open:Nn \g_tmpa_ior {tagpdf-ns-#1.def}
298     \msg_info:nnn {tag}{read-namespace}{#1}
299     \ior_map_inline:Nn \g_tmpa_ior
300     {
301       \__tag_role_read_namespace_line:nw {#1} ##1,,,\q_stop
302     }
303     \ior_close:N\g_tmpa_ior
304   }

```

```

305     {
306       \msg_warning:nnn{tag}{namespace-missing}{#1}
307     }
308   }
309

```

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

1.5 Reading the default data

The order is important as we want pdf2 and latex as default.

```

310 \__tag_role_read_namespace:n {pdf}
311 \__tag_role_read_namespace:n {pdf2}
312 \pdf_version_compare:NnF < {2.0}
313 { \__tag_role_read_namespace:n {mathml} }
314 \bool_set_false:N\l__tag_role_update_bool
315 \__tag_role_read_namespace:n {latex-inline}
316 \__tag_role_read_namespace:n {latex-book}
317 \bool_set_true:N\l__tag_role_update_bool
318 \__tag_role_read_namespace:n {latex}
319 \__tag_role_read_namespace:n {pdf}
320 \__tag_role_read_namespace:n {pdf2}

```

But is the class provides a `\chapter` command then we switch

```

321 \pdf_version_compare:NnTF < {2.0}
322 {
323   \hook_gput_code:nnn {begindocument}{tagpdf}
324   {
325     \cs_if_exist:NT \chapter
326     {
327       \prop_map_inline:cn{g__tag_role_NS_latex-book_prop}
328       {
329         \__tag_role_add_tag:ne {#1}{\use_i:nn #2\c_empty_tl\c_empty_tl}
330       }
331     }
332   }
333 }
334 {
335   \hook_gput_code:nnn {begindocument}{tagpdf}
336   {
337     \cs_if_exist:NT \chapter
338     {
339       \prop_map_inline:cn{g__tag_role_NS_latex-book_prop}
340       {
341         \prop_gput:Nnn \g__tag_role_tags_NS_prop { #1 }{ latex-book }
342       }
343     }
344   }
345 }

```

1.6 Parent-child rules

PDF define various rules about which tag can be a child of another tag. The following code implements the matrix to allow to use it in tests.

`\g_tag_role_parent_child_intarray` This intarray will store the rule as a number. For parent nm and child ij (n,m,i,j digits) the rule is at position nmij. As we have around 56 tags, we need roughly a size 6000.

```
346 \intarray_new:Nn \g_tag_role_parent_child_intarray {6000}
```

(End of definition for `\g_tag_role_parent_child_intarray`.)

`\c_tag_role_rules_prop` These two properties map the rule strings to numbers and back. There are in tagpdf-
`\c_tag_role_rules_num_prop` data.dtx near the csv files for easier maintenance.

(End of definition for `\c_tag_role_rules_prop` and `\c_tag_role_rules_num_prop`.)

`_tag_store_parent_child_rule:nnn` The helper command is used to store the rule. It assumes that parent and child are given as 2-digit number!

```
347 \cs_new_protected:Npn \_tag_store_parent_child_rule:nnn #1 #2 #3 % num parent, num child, #3
348 {
349   \intarray_gset:Nnn \g_tag_role_parent_child_intarray
350     { #1#2 }{0\prop_item:Nn\c_tag_role_rules_prop{#3}}
351 }
```

(End of definition for `_tag_store_parent_child_rule:nnn`.)

1.6.1 Reading in the csv-files

This counter will be used to identify the first (non-comment) line

```
352 \int_zero:N \l_tag_tmpa_int
```

Open the file depending on the PDF version

```
353 \pdf_version_compare:NnTF < {2.0}
354 {
355   \ior_open:Nn \g_tmpa_ior {tagpdf-parent-child.csv}
356 }
357 {
358   \ior_open:Nn \g_tmpa_ior {tagpdf-parent-child-2.csv}
359 }
```

Now the main loop over the file

```
360 \ior_map_inline:Nn \g_tmpa_ior
361 {
```

ignore lines containing only comments

```
362   \tl_if_empty:nF{#1}
363   {
```

count the lines ...

```
364     \int_incr:N\l_tag_tmpa_int
```

put the line into a seq. Attention! empty cells are dropped.

```
365     \seq_set_from_clist:Nn\l_tag_tmpa_seq { #1 }
366     \int_compare:nNnTF {\l_tag_tmpa_int}=1
```

This handles the header line. It gives the tags 2-digit numbers

```
367     {
368       \seq_map_indexed_inline:Nn \l_tag_tmpa_seq
369       {
370         \prop_gput:Nnx\g_tag_role_index_prop
371           {##2}
372         {\int_compare:nNnT{##1}<{10}{0}##1}
373       }
374     }
```

now the data lines.

```

375         {
376             \seq_set_from_clist:Nn\l__tag_tmpa_seq { #1 }
get the name of the child tag from the first column
377             \seq_pop_left:NN\l__tag_tmpa_seq\l__tag_tmpa_tl
get the number of the child, and store it in \l__tag_tmpb_tl
378             \prop_get:NVN \g__tag_role_index_prop \l__tag_tmpa_tl \l__tag_tmpb_tl
remove column 2+3
379             \seq_pop_left:NN\l__tag_tmpa_seq\l__tag_tmpa_tl
380             \seq_pop_left:NN\l__tag_tmpa_seq\l__tag_tmpa_tl
Now map over the rest. The index ##1 gives us the number of the parent, ##2 is the
data.
381             \seq_map_indexed_inline:Nn \l__tag_tmpa_seq
382             {
383                 \exp_args:Nnx
384                 \__tag_store_parent_child_rule:nnn {##1}{\l__tag_tmpb_tl}{ ##2 }
385             }
386         }
387     }
388 }

```

close the read handle.

```

389 \ior_close:N\g_tmpa_ior

```

The Root, Hn and mathml tags are special and need to be added explicitly

```

390 \prop_get:NnN\g__tag_role_index_prop{StructTreeRoot}\l__tag_tmpa_tl
391 \prop_gput:Nnx\g__tag_role_index_prop{Root}{\l__tag_tmpa_tl}
392 \prop_get:NnN\g__tag_role_index_prop{Hn}\l__tag_tmpa_tl
393 \pdf_version_compare:NnTF < {2.0}
394 {
395     \int_step_inline:nn{6}
396     {
397         \prop_gput:Nnx\g__tag_role_index_prop{H#1}{\l__tag_tmpa_tl}
398     }
399 }
400 {
401     \int_step_inline:nn{10}
402     {
403         \prop_gput:Nnx\g__tag_role_index_prop{H#1}{\l__tag_tmpa_tl}
404     }

```

all mathml tags are currently handled identically

```

405     \prop_get:NnN\g__tag_role_index_prop {mathml}\l__tag_tmpa_tl
406     \prop_get:NnN\g__tag_role_index_prop {math}\l__tag_tmpb_tl
407     \prop_map_inline:Nn \g__tag_role_NS_mathml_prop
408     {
409         \prop_gput:NnV\g__tag_role_index_prop{#1}\l__tag_tmpa_tl
410     }
411     \prop_gput:NnV\g__tag_role_index_prop{math}\l__tag_tmpb_tl
412 }

```


1.6.2 Retrieving the parent-child rule

This command retrieves the rule (as a number) and stores it in the tl-var. It assumes that the tag in #1 is a standard tag after role mapping for which a rule exist and is *not* one of Part, Div, NonStruct as the real parent has already been identified. #3 can be used to pass along data about the original tags and is only used in messages.

TODO check temporary variables. Check if the tl-var should be fix.

```

413 \tl_new:N \l__tag_parent_child_check_tl
414 \cs_new_protected:Npn \__tag_role_get_parent_child_rule:nnnN #1 #2 #3 #4
415   % #1 parent (string) #2 child (string) #3 text for messages (eg. about Div or Rolemapping)
416   % #4 tl for state
417   {
418     \prop_get:NnN \g__tag_role_index_prop{#1}\l__tag_tmpa_tl
419     \prop_get:NnN \g__tag_role_index_prop{#2}\l__tag_tmpb_tl
420     \bool_lazy_and:nnTF
421     { ! \quark_if_no_value_p:N \l__tag_tmpa_tl }
422     { ! \quark_if_no_value_p:N \l__tag_tmpb_tl }
423     {

```

Get the rule from the intarray

```

424     \tl_set:Nx#4
425     {
426       \intarray_item:Nn
427       \g__tag_role_parent_child_intarray
428       {\l__tag_tmpa_tl\l__tag_tmpb_tl}
429     }

```

If the state is something is wrong ...

```

430     \int_compare:nNnT
431     {#4} = {\prop_item:Nn\c__tag_role_rules_prop{}}
432     {
433       %warn ?

```

we must take the current child from the stack if is already there, depending on location the check is called, this could also remove the parent, but that is ok too.

```

434     }

```

This is the message, this can perhaps go into debug mode.

```

435     \group_begin:
436     \int_compare:nNnT {\l__tag_tmpa_int*\l__tag_loglevel_int} > { 0 }
437     {
438       \prop_get:NVNF\c__tag_role_rules_num_prop #4 \l__tag_tmpa_tl
439       {
440         \tl_set:Nn \l__tag_tmpa_tl {unknown}
441       }
442       \tl_set:Nn \l__tag_tmpb_tl {#1}
443       \msg_note:nnxxx
444       { tag }
445       { role-parent-child }
446       { #1 }
447       { #2 }
448       {
449         #4~(=\l__tag_tmpa_tl')
450         \iow_newline:
451         #3

```

```

452         }
453     }
454     \group_end:
455 }
456 {
457     \tl_set:Nn#4 {0}
458     \msg_warning:nnxxx
459     { tag }
460     {role-parent-child}
461     { #1 }
462     { #2 }
463     { unknown! }
464 }
465 }
466 \cs_generate_variant:Nn\__tag_role_get_parent_child_rule:nnnN {VVVN,VVnN}

```

(End of definition for __tag_role_get_parent_child_rule:nnnN.)

`__tag_check_parent_child:nnnnN` This commands translates rolemaps its arguments and then calls `__tag_role_get-parent_child_rule:nnnN`. It does not try to resolve inheritance of Div etc but instead warns that the rule can not be detected in this case. In pdf 2.0 the name spaces of the tags are relevant, so we have arguments for them, but in pdf <2.0 they are ignored and can be left empty.

```

467 \pdf_version_compare:NnTF < {2.0}
468 {
469     \cs_new_protected:Npn \__tag_check_parent_child:nnnnN #1 #2 #3 #4 #5
470     {%#1 parent tag, #2 NS, #3 child tag, #4 NS, #5 tl var
471     {

```

for debugging messages we store the arguments.

```

472     \prop_put:Nnn \l__tag_role_debug_prop {parent} {#1}
473     \prop_put:Nnn \l__tag_role_debug_prop {child} {#3}

```

get the standard tags through rolemapping if needed at first the parent

```

474     \prop_get:NnNTF \g__tag_role_index_prop {#1}\l__tag_tmpa_tl
475     {
476         \tl_set:Nn \l__tag_tmpa_tl {#1}
477     }
478     {
479         \prop_get:NnNF \g__tag_role_rolemap_prop {#1}\l__tag_tmpa_tl
480         {
481             \tl_set:Nn \l__tag_tmpa_tl {\q_no_value}
482         }
483     }

```

now the child

```

484     \prop_get:NnNTF \g__tag_role_index_prop {#3}\l__tag_tmpb_tl
485     {
486         \tl_set:Nn \l__tag_tmpb_tl {#3}
487     }
488     {
489         \prop_get:NnNF \g__tag_role_rolemap_prop {#3}\l__tag_tmpb_tl
490         {
491             \tl_set:Nn \l__tag_tmpb_tl {\q_no_value}
492         }
493     }

```

if we got tags for parent and child we call the checking command

```

494     \bool_lazy_and:nnTF
495     { ! \quark_if_no_value_p:N \l__tag_tmpa_tl }
496     { ! \quark_if_no_value_p:N \l__tag_tmpb_tl }
497     {
498         \__tag_role_get_parent_child_rule:VVnN
499         \l__tag_tmpa_tl \l__tag_tmpb_tl
500         {Rolemapped~from:~'#1'~-->~'#3'}
501         #5
502     }
503     {
504         \tl_set:Nn #5 {0}
505         \msg_warning:nnxxx
506         { tag }
507         {role-parent-child}
508         { #1 }
509         { #3 }
510         { unknown! }
511     }
512 }
513 \cs_new_protected:Npn \__tag_check_parent_child:nnN #1#2#3
514 {
515     \__tag_check_parent_child:nnnnN {#1}{#2}{#3}
516 }
517 }

```

and now the pdf 2.0 version The version with three arguments retrieves the default names space and then calls the full command. Not sure if this will ever be needed but we leave it for now.

```

518 {
519     \cs_new_protected:Npn \__tag_check_parent_child:nnN #1 #2 #3
520     {
521         \prop_get:NnN\g__tag_role_tags_NS_prop {#1}\l__tag_role_tag_namespace_tmpa_tl
522         \prop_get:NnN\g__tag_role_tags_NS_prop {#2}\l__tag_role_tag_namespace_tmpb_tl
523         \str_if_eq:nnT{#2}{MC}{\tl_clear:N \l__tag_role_tag_namespace_tmpb_tl}
524         \bool_lazy_and:nnTF
525         { ! \quark_if_no_value_p:N \l__tag_role_tag_namespace_tmpa_tl }
526         { ! \quark_if_no_value_p:N \l__tag_role_tag_namespace_tmpb_tl }
527         {
528             \__tag_check_parent_child:nVnVN
529             {#1}\l__tag_role_tag_namespace_tmpa_tl
530             {#2}\l__tag_role_tag_namespace_tmpb_tl
531             #3
532         }
533         {
534             \tl_set:Nn #3 {0}
535             \msg_warning:nnxxx
536             { tag }
537             {role-parent-child}
538             { #1 }
539             { #2 }
540             { unknown! }
541         }
542     }

```

and now the real command.

```

543     \cs_new_protected:Npn \__tag_check_parent_child:nnnnN #1 #2 #3 #4 #5 %tag,NS,tag,NS, t1 va
544     {
545         \prop_put:Nnn \l__tag_role_debug_prop {parent} {#1/#2}
546         \prop_put:Nnn \l__tag_role_debug_prop {child} {#3/#4}

```

If the namespace is empty, we assume a standard tag, otherwise we retrieve the rolemapping from the namespace

```

547         \tl_if_empty:nTF {#2}
548         {
549             \tl_set:Nn \l__tag_tmpa_tl {#1}
550         }
551         {
552             \prop_get:cnNTF
553             { g__tag_role_NS_#2_prop }
554             {#1}
555             \l__tag_tmpa_tl
556             {
557                 \tl_set:Nx \l__tag_tmpa_tl {\tl_head:N\l__tag_tmpa_tl}
558                 \tl_if_empty:NT\l__tag_tmpa_tl
559                 {
560                     \tl_set:Nn \l__tag_tmpa_tl {#1}
561                 }
562             }
563             {
564                 \tl_set:Nn \l__tag_tmpa_tl {\q_no_value}
565             }
566         }

```

and the same for the child If the namespace is empty, we assume a standard tag, otherwise we retrieve the rolemapping from the namespace

```

567         \tl_if_empty:nTF {#4}
568         {
569             \tl_set:Nn \l__tag_tmpb_tl {#3}
570         }
571         {
572             \prop_get:cnNTF
573             { g__tag_role_NS_#4_prop }
574             {#3}
575             \l__tag_tmpb_tl
576             {
577                 \tl_set:Nx \l__tag_tmpb_tl { \tl_head:N\l__tag_tmpb_tl }
578                 \tl_if_empty:NT\l__tag_tmpb_tl
579                 {
580                     \tl_set:Nn \l__tag_tmpb_tl {#3}
581                 }
582             }
583             {
584                 \tl_set:Nn \l__tag_tmpb_tl {\q_no_value}
585             }
586         }

```

and now get the relation

```

587         \bool_lazy_and:nnTF
588         { ! \quark_if_no_value_p:N \l__tag_tmpa_tl }

```

```

589     { ! \quark_if_no_value_p:N \l__tag_tmpb_tl }
590     {
591         \__tag_role_get_parent_child_rule:VVnN
592         \l__tag_tmpa_tl \l__tag_tmpb_tl
593         {Rolemapped~from~'#1/#2'--->~'#3\str_if_empty:nF{#4}{/#4}'}
594         #5
595     }
596     {
597         \tl_set:Nn #5 {0}
598         \msg_warning:nnxxx
599         { tag }
600         {role-parent-child}
601         { #1 }
602         { #3 }
603         { unknown! }
604     }
605 }
606 }
607 \cs_generate_variant:Nn\__tag_check_parent_child:nnN {VVN}
608 \cs_generate_variant:Nn\__tag_check_parent_child:nnnnN {VVVVN,nVnVN,VVnnN}
609 \endpackage

```

(End of definition for `__tag_check_parent_child:nnnnN`.)

`\tag_check_child:nnTF`

```

610 \base\prg_new_protected_conditional:Npnn \tag_check_child:nn #1 #2 {T,F,TF}{\prg_return_true:}
611 \*package
612 \prg_set_protected_conditional:Npnn \tag_check_child:nn #1 #2 {T,F,TF}
613 {
614     \seq_get:NN\g__tag_struct_stack_seq\l__tag_tmpa_tl
615     \__tag_struct_get_parentrole:eNN
616     {\l__tag_tmpa_tl}
617     \l__tag_get_parent_tmpa_tl
618     \l__tag_get_parent_tmpb_tl
619     \__tag_check_parent_child:VVnnN
620     \l__tag_get_parent_tmpa_tl
621     \l__tag_get_parent_tmpb_tl
622     {#1}{#2}
623     \l__tag_parent_child_check_tl
624     \int_compare:nNnTF { \l__tag_parent_child_check_tl } < {0}
625     {\prg_return_false:}
626     {\prg_return_true:}
627 }

```

(End of definition for `\tag_check_child:nnTF`. This function is documented on page 140.)

1.7 Remapping of tags

In some context it can be necessary to remap or replace the tags. That means instead of `tag=H1` or `tag=section` one wants the effect of `tag=Span`. Or instead of `tag=P` one wants `tag=Code`.

The following command provide some general interface for this. The core idea is that before a tag is set it is fed through a function that can change it. We want to be able to chain such functions, so all of them manipulate the same variables.

```

\l__tag_role_remap_tag_tl
\l__tag_role_remap_NS_tl 628 \tl_new:N \l__tag_role_remap_tag_tl
629 \tl_new:N \l__tag_role_remap_NS_tl

(End of definition for \l__tag_role_remap_tag_tl and \l__tag_role_remap_NS_tl.)

\__tag_role_remap: This function is used in the structure and the mc code before using a tag. By default it
does nothing with the tl vars. Perhaps this should be a hook?
630 \cs_new_protected:Npn \__tag_role_remap: { }

(End of definition for \__tag_role_remap:.)

\__tag_role_remap_id: This is copy in case we have to restore the main command.
631 \cs_set_eq:NN \__tag_role_remap_id: \__tag_role_remap:

(End of definition for \__tag_role_remap_id:.)

\__tag_role_remap_inline: The mapping is meant to “degrade” tags, e.g. if used inside some complex object. The
pdf<2.0 code maps the tag to the new role, the pdf 2.0 code only switch the NS.
632 \pdf_version_compare:NnTF < {2.0}
633 {
634   \cs_new_protected:Npn \__tag_role_remap_inline:
635   {
636     \prop_get:cVNT { g__tag_role_NS_latex-inline_prop } \l__tag_role_remap_tag_tl \l__tag_role_remap_NS_tl
637     {
638       \tl_set:Nx \l__tag_role_remap_tag_tl
639       {
640         \exp_last_unbraced:NV \use_i:nn \l__tag_tmpa_tl
641       }
642       \tl_set:Nx \l__tag_role_remap_NS_tl
643       {
644         \exp_last_unbraced:NV \use_ii:nn \l__tag_tmpa_tl
645       }
646     }
647     \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
648     {
649       \msg_note:nnx { tag } { role-remapping } { \l__tag_role_remap_tag_tl }
650     }
651   }
652 }
653 {
654   \cs_new_protected:Npn \__tag_role_remap_inline:
655   {
656     \prop_get:cVNT { g__tag_role_NS_latex-inline_prop } \l__tag_role_remap_tag_tl \l__tag_role_remap_NS_tl
657     {
658       \tl_set:Nn \l__tag_role_remap_NS_tl { latex-inline }
659     }
660     \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
661     {
662       \msg_note:nnx { tag } { role-remapping } { \l__tag_role_remap_tag_tl / latex-
inline }
663     }
664   }
665 }

(End of definition for \__tag_role_remap_inline:.)

```

1.8 Key-val user interface

The user interface uses the key `add-new-tag`, which takes either a keyval list as argument, or a tag/role.

```
tag_(rolemap-key)
tag-namespace_(rolemap-key) 666 \keys_define:nn { __tag / tag-role }
role_(rolemap-key)          667 {
role-namespace_(rolemap-key) 668   ,tag .tl_set:N = \l__tag_role_tag_tmpa_tl
add-new-tag_(setup-key)     669   ,tag-namespace .tl_set:N = \l__tag_role_tag_namespace_tmpa_tl
                             670   ,role .tl_set:N = \l__tag_role_role_tmpa_tl
                             671   ,role-namespace .tl_set:N = \l__tag_role_role_namespace_tmpa_tl
                             672 }
                             673
                             674 \keys_define:nn { __tag / setup }
                             675 {
                             676   add-new-tag .code:n =
                             677   {
                             678     \keys_set_known:nnnN
                             679     {__tag/tag-role}
                             680     {
                             681       tag-namespace=user,
                             682       role-namespace=, %so that we can test for it.
                             683       #1
                             684     }{__tag/tag-role}\l_tmpa_tl
                             685     \tl_if_empty:NF \l_tmpa_tl
                             686     {
                             687       \exp_args:NNno \seq_set_split:Nnn \l_tmpa_seq { / } {\l_tmpa_tl/}
                             688       \tl_set:Nx \l__tag_role_tag_tmpa_tl { \seq_item:Nn \l_tmpa_seq {1} }
                             689       \tl_set:Nx \l__tag_role_role_tmpa_tl { \seq_item:Nn \l_tmpa_seq {2} }
                             690     }
                             691     \tl_if_empty:NT \l__tag_role_role_namespace_tmpa_tl
                             692     {
                             693       \prop_get:NVNTF
                             694       \g__tag_role_tags_NS_prop
                             695       \l__tag_role_role_tmpa_tl
                             696       \l__tag_role_role_namespace_tmpa_tl
                             697       {
                             698         \prop_if_in:NVF\g__tag_role_NS_prop \l__tag_role_role_namespace_tmpa_tl
                             699         {
                             700           \tl_set:Nn \l__tag_role_role_namespace_tmpa_tl {user}
                             701         }
                             702       }
                             703       {
                             704         \tl_set:Nn \l__tag_role_role_namespace_tmpa_tl {user}
                             705       }
                             706     }
                             707     \pdf_version_compare:NnTF < {2.0}
                             708     {
                             709       %TODO add check for emptyness?
                             710       \__tag_role_add_tag:VV
                             711       \l__tag_role_tag_tmpa_tl
                             712       \l__tag_role_role_tmpa_tl
                             713     }
                             }
```

```

714     {
715         \__tag_role_add_tag:VVVV
716         \l__tag_role_tag_tmpa_tl
717         \l__tag_role_tag_namespace_tmpa_tl
718         \l__tag_role_role_tmpa_tl
719         \l__tag_role_role_namespace_tmpa_tl
720     }
721 }
722 }
723 \</package>

```

(End of definition for tag (rolemap-key) and others. These functions are documented on page 140.)

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} {2023-06-14} {0.98i}
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 of definition for `interwordspace` (setup-key) and `show-spaces` (setup-key). These functions are documented on page [161](#).)

`__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 of 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, 27, 28, 44, 45, 46
<code>_</code>	266, 277
A	
<code>activate_␣(setup-key)</code>	33, <u>195</u>
<code>activate-all_␣(setup-key)</code>	6, <u>229</u>
<code>activate-mc_␣(setup-key)</code>	6, <u>229</u>
<code>activate-space_␣(setup-key)</code>	6, <u>229</u>
<code>activate-struct_␣(setup-key)</code>	6, <u>229</u>
<code>activate-tree_␣(setup-key)</code>	6, <u>229</u>
<code>actualtext_␣(mc-key)</code>	61, <u>238</u> , <u>420</u>
<code>actualtext_␣(struct-key)</code>	90, <u>422</u>
<code>add-new-tag_␣(setup-key)</code>	140, <u>666</u>
<code>\AddToHook</code>	13, 16, 50, 80, 192, 212, 282, 300, 315, 340, 384
<code>AF_␣(struct-key)</code>	91, <u>536</u>
<code>AInline_␣(struct-key)</code>	91, <u>536</u>
<code>AInline-o_␣(struct-key)</code>	91, <u>536</u>
<code>alt_␣(mc-key)</code>	61, <u>238</u> , <u>420</u>
<code>alt_␣(struct-key)</code>	90, <u>422</u>
<code>artifact_␣(mc-key)</code>	61, <u>238</u> , <u>420</u>
artifact-bool internal commands:	
<code>__artifact-bool</code>	120
artifact-type internal commands:	
<code>__artifact-type</code>	120
<code>attr-unknown</code>	19, 49
<code>attribute_␣(struct-key)</code>	91, <u>1069</u>
<code>attribute-class_␣(struct-key)</code>	91, <u>1035</u>
B	
bool commands:	
<code>\bool_gset_eq:Nn</code>	404, 419, 431, 449
<code>\bool_gset_false:N</code>	43, 54, 221, 405, 408, 432
<code>\bool_gset_true:N</code>	42, 48, 120, 160, 336
<code>\bool_if:NTF</code>	9, 9, 18, 28, 32, 33, 37, 82, 175, 183, 220, 224, 237, 238, 263, 264, 271, 274, 280, 284, 286, 302, 308, 324, 339, 340, 343, 357, 362, 399, 414, 426, 444, 763
<code>\bool_if:NTF</code>	6, 323
<code>\bool_lazy_all:nTF</code>	79, 214
<code>\bool_lazy_and:nnTF</code>	113, 123, 282, 420, 429, 494, 524, 587
<code>\bool_lazy_and_p:nn</code>	8
<code>\bool_new:N</code>	17, 21, 22, 41, 62, 115, 116, 117, 118, 119, 121, 123, 125, 228, 233, 235, 237, 395
<code>\bool_set_false:N</code>	161, 186, 187, 192, 193, 205, 206, 222, 314, 371, 398, 425
<code>\bool_set_true:N</code>	122, 124, 197, 198, 218, 219, 229, 317, 370
box commands:	
<code>\box_dp:N</code>	177, 181
<code>\box_ht:N</code>	167
<code>\box_new:N</code>	110, 111
<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>	73, 178
C	
<code>\c</code>	229, 230
c@g internal commands:	
<code>\c@g__tag_MCID_abs_int</code>	11, 15, 25, 34, 47, 54, 65, 71, 73, 135, 149, 163, 237, 242, 271, 311, 318, 383
<code>\c@g__tag_parenttree_obj_int</code>	136
<code>\c@g__tag_struct_abs_int</code>	6, 17, 54, 74, 77, 78, 130, 138, 141, 143, 419, 434, 459, 471, 485, 501, 509, 521, 531, 551, 554, 559, 578, 581, 586, 619, 621, 626, 676, 687, 688, 689, 690, 693, 695, 701, 704, 719, 726, 744, 753, 761, 789, 797, 802, 904, 960, 1062, 1065, 1113
cctab commands:	
<code>\c_document_cctab</code>	66
<code>\chapter</code>	150, 325, 337
clist commands:	
<code>\clist_const:Nn</code>	112, 113
<code>\clist_if_empty:NTF</code>	1074
<code>\clist_map_inline:nn</code>	136, 516
<code>\clist_new:N</code>	108
<code>\clist_set:Nn</code>	1039, 1073
color commands:	
<code>\color_select:n</code>	266, 277
cs commands:	
<code>\cs_generate_variant:Nn</code>	41, 42, 98, 104, 127, 128, 128, 129, 130, 131, 132, 133, 134, 135, 136, 153, 154, 154, 157, 158, 163, 168, 171, 177, 178, 179, 180, 181, 182, 195, 206, 208, 223, 226,

234, 235, 288, 299, 325, 466, 567, 592, 607, 608, 612, 976, 985, 998, 1008	
\cs_gset_eq:NN	248, 794, 795, 901, 902, 957, 958
\cs_if_exist:NTF	82, 325, 337, 345, 386
\cs_if_exist_p:N	9, 218
\cs_if_exist_use:NTF	313, 979
\cs_if_free:NTF	48, 537
\cs_new:Nn	27, 74, 79, 100, 122, 127, 131, 316, 324
\cs_new:Npn	9, 15, 67, 71, 84, 85, 90, 117, 159, 160, 164, 172, 211, 237, 397, 405, 411, 417, 972, 1009
\cs_new_protected:Nn	68, 129, 171, 319
\cs_new_protected:Npn	13, 20, 20, 21, 29, 31, 36, 42, 49, 54, 55, 58, 58, 60, 60, 61, 63, 75, 77, 77, 78, 79, 81, 87, 90, 91, 97, 105, 109, 112, 119, 128, 129, 131, 139, 139, 144, 146, 150, 152, 152, 153, 155, 155, 159, 164, 169, 182, 183, 186, 188, 197, 205, 207, 207, 215, 218, 222, 224, 225, 225, 226, 227, 232, 234, 235, 236, 240, 245, 250, 258, 260, 261, 265, 269, 271, 278, 283, 289, 291, 295, 295, 300, 304, 308, 318, 322, 326, 343, 347, 350, 357, 359, 364, 379, 380, 381, 382, 384, 392, 396, 399, 407, 410, 414, 414, 423, 439, 469, 513, 519, 539, 543, 568, 593, 630, 634, 654, 676, 677, 678, 865, 915, 977, 986, 999, 1022
\cs_set:Nn	464, 465
\cs_set:Npn	38, 43
\cs_set_eq:NN	14, 20, 66, 77, 78, 79, 168, 169, 170, 171, 172, 173, 174, 175, 189, 231, 232, 233, 457, 458, 459, 460, 466, 467, 471, 472, 473, 474, 631, 791, 792, 898, 899, 954, 955
\cs_set_protected:Nn	154, 216, 246, 395, 401, 823, 824
\cs_set_protected:Npn	9, 16, 16, 23, 30, 38, 49, 56, 63, 70, 81, 100, 190, 192, 195, 201, 213, 228, 322, 328, 680, 681, 859, 867, 917
\cs_to_str:N	12, 19, 26, 33, 52, 53, 59, 60
D	
\DeclareOption	42, 43
dim commands:	
\c_max_dim	166, 191
\c_zero_dim	174, 175, 176
\documentclass	22
\DocumentMetadata	21
E	
E _␣ (struct-key)	91, 422
\endinput	28
exclude-header-footer _␣ (setup-key)	35, 452
\ExecuteOptions	44
exp commands:	
\exp_args:Ne	399, 691
\exp_args:Nee	86
\exp_args:NNno	687
\exp_args:NNnx	67
\exp_args:NNx	67, 83, 86, 192, 212
\exp_args:Nnx	81, 323, 327, 383, 385, 459
\exp_args:NV	179, 185, 314, 343, 354, 359
\exp_args:Nx	119, 345
\exp_last_unbraced:NV	163, 164, 218, 219, 422, 426, 640, 644, 848
\exp_not:n	276
F	
file commands:	
\file_if_exist:nTF	295
\file_input:n	267
\fontencoding	6
\fontfamily	6
\fontseries	6
\fontshape	6
\fontsize	6
\footins	348
G	
group commands:	
\group_begin:	70, 158, 185, 334, 435, 542, 571, 644, 686
\group_end:	73, 189, 213, 386, 454, 563, 589, 650, 819
H	
hbox commands:	
\hbox_set:Nn	168, 169
hook commands:	
\hook_gput_code:nnn	7, 7, 26, 30, 43, 50, 137, 198, 199, 323, 335, 337, 341, 479, 492, 502, 515
\hook_new:n	321
\hook_use:n	326
I	
\ignorespaces	33
\immediate	193
int commands:	
\int_abs:n	122
\int_case:nnTF	250
\int_compare:nNnTF	22, 61, 61, 86, 100,

113, 134, 148, 170, 174, 178, 194,
 197, 224, 227, 252, 258, 301, 321,
 330, 345, 352, 359, 366, 366, 368,
 372, 386, 394, 401, 409, 416, 430,
 436, 624, 647, 660, 710, 776, 896, 952
`\int_compare:nTF`
 161, 292, 1055, 1057, 1059, 1083, 1109
`\int_compare_p:nNn` 434
`\int_eval:n` 135, 172, 259, 276, 346,
 431, 436, 439, 459, 471, 485, 501,
 509, 521, 531, 551, 559, 578, 586,
 619, 621, 626, 688, 689, 690, 693,
 695, 701, 704, 726, 744, 753, 789,
 797, 802, 904, 960, 1062, 1065, 1113
`\int_gincr:N` ... 163, 237, 288, 294,
 304, 310, 311, 318, 543, 572, 676, 687
`\int_gset:Nn` 45, 139, 255
`\int_gzero:N` 7, 263
`\int_incr:N` 56, 364
`\int_new:N` 16,
 41, 109, 114, 238, 239, 240, 241, 536
`\int_rand:n` .. 60, 61, 63, 65, 67, 69, 70
`\int_set:Nn` ... 242, 245, 248, 249, 250
`\int_step_inline:nn` 54, 395, 401
`\int_step_inline:nnn` 25
`\int_step_inline:nnnn`
 130, 155, 158, 175, 277, 283
`\int_to_arabic:n` 122, 124
`\int_to_Hex:n` 60, 61, 63, 65, 67, 69, 70
`\int_use:N` 11,
 15, 17, 25, 34, 47, 54, 65, 66, 71, 72,
 73, 74, 98, 100, 138, 141, 143, 147,
 149, 151, 213, 234, 242, 266, 271,
 277, 326, 327, 335, 336, 383, 419,
 544, 548, 549, 552, 554, 575, 581, 1009
`\int_zero:N` 53, 68, 352
intarray commands:
`\intarray_gset:Nnn` 255, 349
`\intarray_item:Nn` 257, 260, 426
`\intarray_new:Nn` 247, 346
interwordspace_␣(setup-key) 161, 6
ior commands:
`\ior_close:N` 303, 389
`\ior_map_inline:Nn` 299, 360
`\ior_open:Nn` 297, 355, 358
`\g_tmpa_ior`
 297, 299, 303, 355, 358, 360, 389
iow commands:
`\iow_newline:` 202, 278, 450
`\iow_now:Nn` 67
`\iow_term:n` 149, 152, 158, 162, 195, 246

K

keys commands:

`\keys_define:nn` 7, 21, 30,
 43, 67, 79, 120, 141, 184, 201, 230,
 238, 248, 254, 372, 421, 423, 452,
 613, 653, 666, 674, 1028, 1035, 1069
`\keys_set:nn` 10,
 18, 64, 170, 205, 324, 328, 339, 460, 699
`\keys_set_known:nnnN` 678

L

`label␣(mc-key)` 61, 238, 420
`label␣(struct-key)` 90, 422
`lang␣(struct-key)` 90, 422
legacy commands:
`\legacy_if:nTF` 65
`\llap` 266
`log␣(setup-key)` 6, 241
ltx. internal commands:
`ltx.__tag.func.alloctag` 261
`ltx.__tag.func.fakespace` 363
`ltx.__tag.func.fill_parent_tree_-`
`line` 716
`ltx.__tag.func.get_num_from` ... 270
`ltx.__tag.func.get_tag_from` ... 289
`ltx.__tag.func.mark_page_-`
`elements` 545
`ltx.__tag.func.mark_shipout` ... 697
`ltx.__tag.func.markspaceoff` ... 427
`ltx.__tag.func.markspaceon` ... 427
`ltx.__tag.func.mc_insert_kids` .. 482
`ltx.__tag.func.mc_num_of_kids` .. 319
`ltx.__tag.func.output_num_from` . 270
`ltx.__tag.func.output_parenttree` 716
`ltx.__tag.func.output_tag_from` . 289
`ltx.__tag.func.pdf_object_ref` .. 348
`ltx.__tag.func.space_chars_-`
`shipout` 448
`ltx.__tag.func.store_mc_data` .. 304
`ltx.__tag.func.store_mc_in_page` 526
`ltx.__tag.func.store_mc_kid` ... 313
`ltx.__tag.func.store_mc_label` .. 309
`ltx.__tag.func.store_struct_-`
`mcabs` 514
`ltx.__tag.func.update_mc_-`
`attributes` 534
`ltx.__tag.tables.role_tag_-`
`attribute` 259
`ltx.__tag.trace.log` 173
`ltx.__tag.trace.show_all_mc_data` 230
`ltx.__tag.trace.show_mc_data` .. 215
`ltx.__tag.trace.show_prop` 190
`ltx.__tag.trace.show_seq` 181
`ltx.__tag.trace.show_struct_data` 236

pdfannot commands:

- \pdfannot_dict_put:nnn 128, 486, 509, 527, 532
- \pdfannot_link_ref_last: ... 496, 519

pdfdict commands:

- \pdfdict_gput:nnn 37, 44, 52, 186, 246, 307
- \pdfdict_if_empty:nTF 301
- \pdfdict_new:n 17, 34, 36
- \pdfdict_put:nnn 645, 646
- \pdfdict_use:n 253, 305, 312

\pdffakespace 35, 225

pdffile commands:

- \pdffile_embed_stream:nnN 537, 567, 573
- \pdffile_embed_stream:nnn .. 129, 546

\pdfglyphcode 20

\pdfinterwordspaceon 23, 24

pdfmanagement commands:

- \pdfmanagement_add:nnn 34, 35, 255, 257, 259, 343
- \pdfmanagement_if_active_p: .. 9, 10
- \pdfmanagement_remove:nn 261

prg commands:

- \prg_do_nothing: 79, 248, 471, 472, 473, 474
- \prg_generate_conditional_variant:Nnn 126
- \prg_new_conditional:Nnn 59, 222
- \prg_new_conditional:Npnn 73, 96, 111, 121, 315, 321, 332
- \prg_new_eq_conditional:NNn . 73, 229
- \prg_new_protected_conditional:Npnn 610
- \prg_replicate:nn 121
- \prg_return_false: 69, 74, 91, 102, 118, 128, 226, 318, 330, 336, 625
- \prg_return_true: 70, 88, 101, 105, 115, 125, 225, 319, 329, 335, 610, 626
- \prg_set_conditional:Npnn 77
- \prg_set_protected_conditional:Npnn 612

\ProcessOptions 45

prop commands:

- \prop_clear:N 157
- \prop_count:N 178
- \prop_get:NnN .. 139, 147, 178, 196, 197, 241, 272, 370, 378, 390, 392, 405, 406, 418, 419, 521, 522, 778, 989
- \prop_get:NnNTF 93, 145, 158, 161, 162, 176, 180, 195, 216, 247, 330, 438, 474, 479, 484, 489, 552, 572, 636, 656, 693, 738, 849, 924
- \prop_gpop:NnN 215
- \prop_gput:Nnn 25, 25, 30, 33, 34, 55, 90, 91, 92, 93, 95, 98, 99, 100, 100, 101, 102, 112, 113, 114, 115, 121, 122, 123, 124, 130, 147, 150, 153, 170, 200, 204, 207, 255, 258, 259, 287, 295, 332, 333, 341, 370, 391, 397, 403, 409, 411, 1024, 1098
- \prop_gremove:Nn 210
- \prop_if_exist:NTF 293, 871, 921
- \prop_if_exist_p:N 431
- \prop_if_in:NnTF 69, 132, 133, 141, 240, 308, 698, 1047, 1087, 1091
- \prop_item:Nn 41, 73, 132, 167, 173, 192, 269, 315, 318, 350, 402, 431, 446, 1096, 1103
- \prop_map_inline:Nn 242, 299, 327, 339, 407
- \prop_map_tokens:Nn 317
- \prop_new:N 7, 8, 9, 10, 11, 11, 18, 23, 24, 31, 32, 64, 66, 105, 168, 200, 1018, 1021
- \prop_put:Nnn 131, 164, 472, 473, 545, 546
- \prop_show:N 58, 92, 175, 813, 816, 1065, 1092
- \providecommand 193, 232
- \ProvidesExplFile 3
- \ProvidesExplPackage 3, 3, 3, 3, 3, 3, 3, 3, 7, 26, 37, 1014

Q

\quad 171, 172

quark commands:

- \q_no_value 481, 491, 564, 584
- \quark_if_no_value:NTF 140, 148, 179, 198, 216, 242, 273, 996
- \quark_if_no_value_p:N 421, 422, 495, 496, 525, 526, 588, 589
- \q_stop 232, 265, 301

R

raw_␣(mc-key) 61, 238, 420

ref_␣(struct-key) 91, 422

ref commands:

- \ref_attribute_gset:nnnn 137, 139, 146, 148, 150
- \ref_label:nn 133, 155, 373
- \ref_value:nn 90, 519
- \ref_value:nnn . 6, 82, 82, 84, 161, 166

ref internal commands:

- __ref_value:nnn 87, 90

regex commands:

- \regex_replace_once:nnN 228
- \renewcommand 370, 371

\RenewDocumentCommand	8	shipout commands:	
\RequirePackage	20, 46, 47, 273, 276, 282, 285, 342	\g_shipout_readonly_int	72, 147, 213, 346
\rlap	277	show-spaces _␣ (setup-key)	161, 6
role _␣ (rolemap-key)	140, 666	\ShowTagging	16, 34, 61
role-missing	19, 50	skip commands:	
role-namespace _␣ (rolemap-key)	140, 666	\skip_horizontal:n	72
role-parent-child	53, 55	\c_zero_skip	72
role-tag	19, 57	stash _␣ (mc-key)	61, 120
role-unknown	19, 50	stash _␣ (struct-key)	90, 422
role-unknown-tag	19, 50	\stepcounter	394
root-AF _␣ (setup-key)	92, 653	str commands:	
S			
\selectfont	6	\str_case:nnTF	52, 731
seq commands:		\str_const:Nn	58
\seq_clear:N	240, 282	\str_if_empty:nTF	593
\seq_const_from_clist:Nn	20, 33	\str_if_eq:nnTF	124, 334, 420, 523
\seq_count:N	22, 25, 252, 340, 1055, 1057, 1059, 1083, 1109	\str_if_eq_p:nn	283, 325, 327
\seq_get:NN	614	\str_new:N	104
\seq_get:NNTF	365, 418, 712, 838, 845	\str_set_convert:Nnnn	135, 261, 282, 434, 447, 453, 465, 479, 495, 525
\seq_gpop:NN	831	\str_use:N	272, 295
\seq_gpop:NNTF	104, 832	\string	20, 21, 22, 193, 356
\seq_gpop_left:NN	227	struct-faulty-nesting	19, 32
\seq_gpush:Nn	12, 14, 87, 94, 719, 759	struct-label-unknown	19, 38
\seq_gput_left:Nn	232, 1051	struct-missing-tag	19, 35
\seq_gput_right:Nn	32, 134, 171, 302	struct-no-objnum	19, 24
\seq_gremove_duplicates:N	264	struct-orphan	25
\seq_gset_eq:NN	156, 218, 247	struct-show-closing	19, 40
\seq_if_empty:N	197, 334	struct-stack _␣ (show-key)	34, 184
\seq_item:Nn	113, 115, 122, 126, 133, 137, 172, 271, 325, 327, 334, 447, 448, 688, 689	struct-unknown	22
\seq_log:N	172, 187, 196, 231, 389, 404	struct-used-twice	19, 36
\seq_map_indexed_inline:Nn	368, 381	sys commands:	
\seq_map_inline:Nn	241, 298, 1045, 1085	\c_sys_backend_str	52
\seq_new:N	11, 13, 15, 16, 16, 17, 18, 18, 19, 19, 106, 107, 169, 1019	\c_sys_engine_str	10, 12
\seq_pop_left:NN	377, 379, 380	\sys_if_engine luatex:TF	41, 42, 66, 71, 84, 85, 107, 225, 265
\seq_put_right:Nn	242	\sys_if_engine pdftex:TF	16
\seq_remove_all:Nn	245	\sys_if_output_pdf:TF	11, 18
\seq_set_eq:NN	204, 205	sys-no-interwordspace	19, 64
\seq_set_from_clist:NN	1040, 1076	T	
\seq_set_from_clist:Nn	84, 87, 193, 213, 365, 376	tabsorder _␣ (setup-key)	6, 253
\seq_set_map:Nnn	265	tag _␣ (mc-key)	61, 238, 420
\seq_set_map_x:Nnn	1041, 1077	tag _␣ (rolemap-key)	140, 666
\seq_set_split:Nnn	134, 446, 687	tag _␣ (struct-key)	90, 422
\seq_show:N	51, 154, 155, 174, 188, 243, 244, 246, 312, 762, 814, 817, 827	tag commands:	
\seq_use:Nn	45, 107, 108, 171, 172, 202, 276, 283, 1056	\tag_check_child:nn	140, 610, 612
\l_tmpa_seq	282, 302, 312, 687, 688, 689	\tag_check_child:nnTF	140, 610
		\tag_get:n	16, 62, 89, 90, 104, 71, 87, 90, 373
		\tag_if_active:	73, 77
		\tag_if_active:TF	16, 18, 72, 203, 317
		\tag_if_active_p:	16, 72
		\tag_if_box_tagged:N	16, 96

\tag_if_box_tagged:NTF	16, 95	\l__tag_active_struct_bool	84, 121 , 123, 186, 192, 197, 205, 218
\tag_if_box_tagged_p:N	16, 95	\g__tag_active_struct_dest_bool	115, 216, 239
\tag_mc_artifact_group_begin:n	60, 60 , 60, 63	\g__tag_active_tree_bool	9, 32, 83, 115 , 234, 324, 339
\tag_mc_artifact_group_end:	60, 60 , 61, 70	__tag_add_missing_mcs:Nn	74, 164 , 164, 216
\tag_mc_begin:n	10, 60 , 25, 66, 112, 154 , 154, 265, 276, 297, 318 , 318, 322, 328, 407, 435, 485, 508	__tag_add_missing_mcs_to_-stream:Nn	58, 58, 186 , 186, 348, 352, 359, 361
\tag_mc_begin_pop:n	60, 74, 78 , 79, 100, 416, 446, 499, 522	\g__tag_attr_class_used_seq	264, 265, 1017 , 1051
\tag_mc_end:	60, 31, 73, 91, 216 , 216, 267, 278, 305, 318 , 319, 395, 401, 413, 442, 497, 520	\g__tag_attr_entries_prop	270, 1017 , 1024, 1047, 1087, 1092, 1096
\tag_mc_end_push:	60, 65, 78 , 78, 81, 401, 428, 483, 506	__tag_attr_new_entry:nn	422, 1022 , 1022, 1032
\tag_mc_if_in:	73, 229	\g__tag_attr_objref_prop	1017, 1091, 1098, 1103
\tag_mc_if_in:TF	60, 42, 59, 222	\l__tag_attr_value_tl	1081, 1100, 1105, 1107, 1111, 1115
\tag_mc_if_in_p:	60, 59 , 222	__tag_check_add_tag_role:nn	131, 169 , 169
\tag_mc_reset:N	61	__tag_check_add_tag_role:nnn	173, 188
\tag_mc_reset_box:N	61, 77 , 77, 228 , 228	__tag_check_if_active_mc:	111
\tag_mc_use:n	60, 36, 36 , 36, 38	__tag_check_if_active_mc:TF	83, 102, 110, 156, 188, 218, 324, 330, 397, 403
\tag_start:	6, 183, 195, 225	__tag_check_if_active_struct:	121
\tag_start:n	6, 183 , 213, 227, 412, 441	__tag_check_if_active_struct:TF	40, 110, 683, 684, 828, 829, 869, 919, 1002
\tag_stop:	6, 183 , 190, 224	__tag_check_if_mc_in_galley:	315
\tag_stop:n	6, 183 , 201, 226, 408, 436	__tag_check_if_mc_in_galley:TF	147, 168
\tag_stop_group_begin:	6, 67, 183 , 183	__tag_check_if_mc_tmb_missing:	321
\tag_stop_group_end:	6, 72, 183 , 189	__tag_check_if_mc_tmb_missing:TF	109, 156, 173, 321
\tag_struct_begin:n	89, 48, 289, 295, 434, 484, 507, 676 , 676, 680, 681	__tag_check_if_mc_tmb_missing_-p:	321
\tag_struct_end:	89, 26, 53, 307, 311, 443, 498, 521, 676 , 677, 823, 824, 862	__tag_check_if_mc_tme_missing:	332
\tag_struct_end:n	89, 678, 859	__tag_check_if_mc_tme_missing:TF	152, 160, 177, 332
\tag_struct_gput:nnn	977, 977, 985	__tag_check_if_mc_tme_missing_-p:	332
\tag_struct_insert_annot:nn	89, 117, 496, 519, 999 , 999, 1008	__tag_check_info_closing_-struct:n	146, 146, 154, 834
\tag_struct_object_ref:n	89, 971 , 972, 976	__tag_check_init_mc_used:	245, 245, 248, 254
\tag_struct_parent_int:	89, 117, 489, 496, 512, 519, 999 , 1009	__tag_check_mc_if_nested:	159, 207 , 207, 335
\tag_struct_use:n	89, 90, 58, 865, 865, 867	__tag_check_mc_if_open:	207, 215, 220, 407
\tag_struct_use_num:n	915, 915, 917		
\tag_tool:n	33, 13 , 13, 14, 16, 20		
tag internal commands:			
__tag_activate_mark_space	427		
\g__tag_active_mc_bool	33, 82, 113, 115 , 233		
\l__tag_active_mc_bool	85, 113, 121 , 187, 193, 198, 206, 219		
\g__tag_active_space_bool	9, 48, 54, 115 , 232		
\g__tag_active_struct_bool	81, 115 , 123, 217, 235, 362		

_tag_check_mc_in_galley:TF ..	315	_tag_exclude_struct_headfoot_-	
_tag_check_mc_in_galley_p: ..	315	begin:n	423 , 464 , 465
_tag_check_mc_pushed_popped:nn		_tag_exclude_struct_headfoot_-	
.....	88 , 95 , 108 , 111 , 116 , 222 , 222	end:	439 , 466 , 467
_tag_check_mc_tag:N		_tag_fakespace	363
.....	172 , 234 , 234 , 347	_tag_fakespace:	66 , 68 , 229
_tag_check_mc_used:n		_tag_finish_structure:	
.....	133 , 250 , 250 , 291	13 , 16 , 321 , 322
\g_tag_check_mc_used_intarray ..		_tag_get_data_mc_counter: ...	9 , 9
.....	245 , 255 , 257 , 260	_tag_get_data_mc_tag:	
_tag_check_no_open_struct:	237 , 237 , 316 , 316
.....	155 , 155 , 836 , 843	_tag_get_data_struct_counter: .	
_tag_check_para_begin_show:nn	416 , 417
.....	260 , 296	_tag_get_data_struct_id: .	405 , 405
_tag_check_para_end_show:nn ...		_tag_get_data_struct_num:	410 , 411
.....	271 , 306	_tag_get_data_struct_tag:	397 , 397
_tag_check_parent_child:nnN ...		_tag_get_mathsubtype	251
.....	513 , 519 , 607	_tag_get_mc_abs_cnt: .	14 , 15 , 19 ,
_tag_check_parent_child:nnnnN .	467	20 , 73 , 93 , 103 , 114 , 168 , 210 , 211 ,
_tag_check_parent_child:nnnnN	219 , 238 , 246 , 254 , 272 , 293 , 307 , 317
.....	189 , 363 , 469 ,	_tag_get_mc_cnt_type_tag	245
.....	515 , 528 , 543 , 608 , 619 , 770 , 890 , 946	_tag_get_num_from	270
_tag_check_show_MCID_by_page: .		\l_tag_get_parent_tmpa_tl	
.....	269 , 269	102 , 187 , 190 , 203 ,
_tag_check_struct_used:n	361 , 364 , 377 , 617 , 620 , 768 , 771 , 785
.....	159 , 159 , 874	\l_tag_get_parent_tmpa_tl_uuuu\l_-	
_tag_check_structure_has_tag:n		_tag_get_parent_tmpb_tl_uuuu\l_-	
.....	131 , 131 , 704	_tag_tmpa_str	99
_tag_check_structure_tag:N ...		\l_tag_get_parent_tmpb_tl	
.....	139 , 139 , 449	103 , 188 , 191 , 203 ,
_tag_check_typeout_v:n	362 , 365 , 377 , 618 , 621 , 769 , 772 , 785
.....	66 , 66 , 107 , 108 , 111 ,	_tag_get_tag_from	289
.....	146 , 154 , 161 , 199 , 208 , 246 , 351 , 356	\l_tag_get_tmpe_tl	99 , 145 ,
_tag_debug_mc_begin_ignore:n	150 , 161 , 163 , 164 , 741 , 747 , 992 , 996
.....	350 , 390	_tag_hook_kernel_after_foot: .	
_tag_debug_mc_begin_insert:n	382 , 391 , 460 , 467 , 474
.....	332 , 343	_tag_hook_kernel_after_head: .	
_tag_debug_mc_end_ignore:	364 , 415	380 , 389 , 459 , 466 , 473
_tag_debug_mc_end_insert:	357 , 405	_tag_hook_kernel_before_foot: .	
_tag_debug_struct_begin_-		381 , 390 , 458 , 465 , 472
ignore:n	392 , 821	_tag_hook_kernel_before_head: .	
_tag_debug_struct_begin_-		379 , 388 , 457 , 464 , 471
insert:n	384 , 818	\g_tag_in_mc_bool	
_tag_debug_struct_end_check:n	17 , 18 , 160 , 221 , 224 ,
.....	414 , 861	336 , 404 , 405 , 408 , 419 , 431 , 432 , 449
_tag_debug_struct_end_ignore: .		_tag_insert_bdc_node	341
.....	407 , 856	_tag_insert_bmc_node	334
_tag_debug_struct_end_insert: .		_tag_insert_emc_node	327
.....	399 , 854	_tag_lastpagelabel:	62 , 63 , 81
_tag_exclude_headfoot_begin: ..		_tag_log	173
.....	396 , 457 , 458	\l_tag_loglevel_int	
_tag_exclude_headfoot_end:	114 , 134 , 148 , 170 , 174 ,
.....	410 , 459 , 460	178 , 197 , 225 , 228 , 242 , 245 , 248 ,

249, 250, 252, 345, 352, 359, 366,
 386, 394, 401, 409, 416, 436, 647, 660
 __tag_mark_spaces 368
 __tag_mc_artifact_begin_marks:n
 20, 42, 78, 344
 \l__tag_mc_artifact_bool
 21, 123, 161, 175, 222, 340
 \l__tag_mc_artifact_type_tl
 20, 127, 131, 135,
 139, 143, 147, 151, 155, 330, 342, 344
 __tag_mc_bdc:nn 230, 233, 234, 274, 306
 __tag_mc_bdc_mcid:n .. 120, 235, 278
 __tag_mc_bdc_mcid:nn
 235, 235, 280, 285
 __tag_mc_begin_marks:nn
 20, 20, 41, 77, 351
 __tag_mc_bmc:n 230, 231, 302
 __tag_mc_bmc_artifact: 300, 300, 313
 __tag_mc_bmc_artifact:n 300, 304, 314
 \l__tag_mc_botmarks_seq
 74, 18, 87, 108,
 155, 158, 172, 205, 213, 218, 317, 334
 __tag_mc_disable_marks: 75, 75
 __tag_mc_emc: 155, 230, 232, 410
 __tag_mc_end_marks: . 20, 60, 79, 411
 \l__tag_mc_firstmarks_seq
 74, 18, 84, 107, 154, 171,
 193, 196, 197, 204, 205, 317, 325, 327
 \g__tag_mc_footnote_marks_seq ... 15
 __tag_mc_get_marks: 81, 81, 146, 167
 __tag_mc_handle_artifact:N
 116, 300, 308, 342
 __tag_mc_handle_mc_label:n
 27, 27, 180, 355
 __tag_mc_handle_mcid:nn
 235, 283, 288, 348
 __tag_mc_handle_stash:n 50,
 131, 131, 153, 210, 289, 289, 299, 383
 __tag_mc_if_in: 59, 73, 222, 229
 __tag_mc_if_in:TF 59, 85, 209, 217, 222
 __tag_mc_if_in_p: 59, 222
 __tag_mc_insert_extra_tmb:n ...
 105, 105, 168
 __tag_mc_insert_extra_tme:n ...
 105, 150, 169
 __tag_mc_insert_mcid_kids:n ...
 122, 122, 138, 229
 __tag_mc_insert_mcid_single_-
 kids:n 122, 127, 230
 \l__tag_mc_key_label_tl
 . 23, 177, 180, 302, 351, 352, 355, 456
 \l__tag_mc_key_properties_tl ...
 23, 162, 251, 266, 267,
 287, 288, 350, 430, 439, 440, 452, 453
 \l__tag_mc_key_stash_bool
 21, 28, 37, 122, 183, 357
 \g__tag_mc_key_tag_tl ... 19, 23,
 165, 225, 237, 243, 316, 338, 409, 426
 \l__tag_mc_key_tag_tl 23, 164, 172,
 174, 224, 242, 337, 347, 349, 351, 425
 __tag_mc_lua_set_mc_type_attr:n
 74, 74, 98, 174
 __tag_mc_lua_unset_mc_type_-
 attr: 74, 100, 223
 \g__tag_mc_main_marks_seq 15
 \g__tag_mc_marks 14,
 22, 31, 44, 51, 62, 68, 85, 88, 194, 214
 \g__tag_mc_multicol_marks_seq ... 15
 \g__tag_mc_parenttree_prop
 18, 19, 100, 148, 167, 295
 \l__tag_mc_ref_abspage_tl
 12, 238, 250, 258, 266
 __tag_mc_set_label_used:n 31, 31, 51
 \g__tag_mc_stack_seq
 19, 87, 94, 104, 231
 __tag_mc_store:nnn . 90, 90, 104, 131
 \l__tag_mc_tmpa_tl .. 13, 252, 255, 259
 g__tag_MCID_abs_int 7
 \g__tag_MCID_byabspage_prop
 10, 248, 257, 265
 \g__tag_MCID_tmp_bypage_int
 16, 151, 255, 263, 276
 \g__tag_mode_lua_bool
 41, 42, 43, 82, 220, 238,
 264, 271, 280, 343, 399, 414, 426, 444
 __tag_new_output_prop_handler:n
 67, 77, 87, 689
 __tag_pairs_prop 190
 \g__tag_para_begin_int
 233, 266, 294, 330, 335
 \l__tag_para_bool 233,
 250, 284, 302, 370, 371, 374, 398, 425
 \g__tag_para_end_int
 233, 277, 304, 330, 336
 \l__tag_para_flattened_bool
 233, 258, 286, 308, 375
 \g__tag_para_main_begin_int
 233, 288, 321, 326
 \g__tag_para_main_end_int
 233, 310, 321, 327
 \l__tag_para_main_tag_tl 233, 257, 291
 \l__tag_para_show_bool
 233, 251, 263, 274
 \l__tag_para_tag_default_tl ... 233
 \l__tag_para_tag_tl 233, 252, 256, 295
 \l__tag_parent_child_check_tl ...
 193, 194, 367, 368, 413,
 623, 624, 775, 776, 895, 896, 951, 952

__tag_parenttree_add_objr:nn ...
 [144](#), [144](#), [386](#)
 \l__tag_parenttree_content_tl ...
 [151](#), [170](#), [182](#), [199](#), [207](#), [228](#), [231](#)
 \g__tag_parenttree_objr_tl ...
 [143](#), [146](#), [228](#)
 __tag_pdf_name_e:n [85](#), [85](#)
 __tag_pdf_object_ref [348](#)
 __tag_prop_gput:Nnn
 [9](#), [23](#), [84](#), [90](#), [93](#),
 [95](#), [97](#), [100](#), [105](#), [112](#), [114](#), [132](#), [141](#),
 [147](#), [168](#), [170](#), [177](#), [256](#), [264](#), [269](#),
 [280](#), [288](#), [458](#), [470](#), [484](#), [500](#), [508](#),
 [530](#), [553](#), [580](#), [620](#), [660](#), [694](#), [725](#),
 [743](#), [752](#), [801](#), [880](#), [936](#), [993](#), [1061](#), [1112](#)
 __tag_prop_item:Nn .. [9](#), [43](#), [168](#), [173](#)
 __tag_prop_new:N [9](#),
 [9](#), [10](#), [11](#), [18](#), [86](#), [168](#), [168](#), [179](#), [688](#)
 __tag_prop_show:N [9](#), [56](#), [168](#), [175](#), [182](#)
 __tag_ref_label:nn
 [29](#), [152](#), [152](#), [158](#), [269](#), [708](#)
 __tag_ref_value:nnn ... [42](#), [159](#),
 [159](#), [162](#), [162](#), [163](#), [166](#), [178](#), [179](#),
 [240](#), [294](#), [305](#), [382](#), [872](#), [878](#), [881](#), [887](#)
 __tag_ref_value_lastpage:nn ...
 . [46](#), [141](#), [155](#), [158](#), [164](#), [164](#), [273](#), [287](#)
 \c__tag_refmc_clist [112](#)
 \c__tag_refstruct_clist [112](#)
 g__tag_role/RoleMap_dict [17](#)
 __tag_role_add_tag:nn
 [129](#), [129](#), [157](#), [253](#), [329](#), [710](#)
 __tag_role_add_tag:nnnn
 [171](#), [171](#), [208](#), [285](#), [715](#)
 __tag_role_alloctag:nnn [83](#),
 [87](#), [97](#), [109](#), [119](#), [128](#), [144](#), [183](#), [250](#), [281](#)
 \l__tag_role_debug_prop
 [141](#), [11](#), [472](#), [473](#), [545](#), [546](#)
 __tag_role_get:nnNN
 [158](#), [160](#), [168](#), [209](#), [211](#), [226](#), [720](#)
 __tag_role_get_parent_child_
 rule:nnnN [154](#), [413](#), [414](#), [466](#), [498](#), [591](#)
 \g__tag_role_index_prop . [141](#), [10](#),
 [370](#), [378](#), [390](#), [391](#), [392](#), [397](#), [403](#),
 [405](#), [406](#), [409](#), [411](#), [418](#), [419](#), [474](#), [484](#)
 \g__tag_role_NS<ns>_class_prop [141](#)
 \g__tag_role_NS<ns>_prop [141](#)
 \g__tag_role_NS_mathml_prop ... [407](#)
 __tag_role_NS_new:nnn [143](#),
 [19](#), [21](#), [29](#), [72](#), [73](#), [76](#), [78](#), [79](#), [80](#), [82](#)
 \g__tag_role_NS_prop
 ... [141](#), [9](#), [25](#), [55](#), [145](#), [299](#), [317](#), [698](#)
 \g__tag_role_parent_child_
 intarray [346](#), [349](#), [427](#)

__tag_role_read_namespace:n ...
 [291](#), [291](#),
 [310](#), [311](#), [313](#), [315](#), [316](#), [318](#), [319](#), [320](#)
 __tag_role_read_namespace_
 line:nw [228](#), [232](#), [265](#), [301](#)
 __tag_role_remap:
 [630](#), [630](#), [631](#), [793](#), [900](#), [956](#)
 __tag_role_remap_id: [631](#), [631](#)
 __tag_role_remap_inline:
 [632](#), [634](#), [654](#)
 \l__tag_role_remap_NS_tl ... [628](#),
 [642](#), [658](#), [792](#), [795](#), [899](#), [902](#), [955](#), [958](#)
 \l__tag_role_remap_tag_tl
 [628](#), [636](#), [638](#), [649](#),
 [656](#), [662](#), [791](#), [794](#), [898](#), [901](#), [954](#), [957](#)
 \l__tag_role_role_namespace_
 tmpa_tl [12](#),
 [671](#), [691](#), [696](#), [698](#), [700](#), [704](#), [719](#)
 \l__tag_role_role_tmpa_tl
 [12](#), [670](#), [689](#), [695](#), [712](#), [718](#)
 \g__tag_role_rolemap_prop .. [141](#),
 [17](#), [147](#), [150](#), [153](#), [162](#), [242](#), [479](#), [489](#)
 \c__tag_role_rules_num_prop [347](#), [438](#)
 \c__tag_role_rules_prop [347](#), [350](#), [431](#)
 \l__tag_role_tag_namespace_tmpa_
 tl [12](#), [521](#), [525](#), [529](#), [669](#), [717](#)
 \l__tag_role_tag_namespace_tmpb_
 tl [522](#), [523](#), [526](#), [530](#)
 \l__tag_role_tag_tmpa_tl
 [12](#), [668](#), [688](#), [711](#), [716](#)
 \g__tag_role_tags_class_prop ...
 ... [141](#), [8](#), [92](#), [101](#), [114](#), [123](#), [139](#), [241](#)
 \g__tag_role_tags_NS_prop
 [141](#), [7](#), [90](#), [99](#), [112](#), [121](#), [132](#), [141](#),
 [176](#), [240](#), [341](#), [446](#), [521](#), [522](#), [694](#), [849](#)
 \l__tag_role_tmpa_seq [12](#)
 \l__tag_role_update_bool
 [228](#), [229](#), [237](#), [314](#), [317](#)
 \c__tag_role_userNS_id_str
 [142](#), [58](#), [82](#)
 \g__tag_root_default_tl [195](#)
 \g__tag_saved_in_mc_bool
 [395](#), [404](#), [419](#), [431](#), [449](#)
 __tag_seq_gput_right:Nn [9](#),
 [30](#), [168](#), [171](#), [178](#), [184](#), [189](#), [199](#), [216](#)
 __tag_seq_item:Nn ... [9](#), [38](#), [168](#), [172](#)
 __tag_seq_new:N
 [9](#), [9](#), [16](#), [88](#), [168](#), [169](#), [180](#), [690](#)
 __tag_seq_show:N . [9](#), [49](#), [168](#), [174](#), [181](#)
 __tag_show_spacemark [354](#)
 \l__tag_showspace_bool ... [14](#), [26](#), [35](#)
 __tag_space_chars_shipout [448](#)
 \g__tag_state_prop . [200](#), [207](#), [210](#), [215](#)

```

\__tag_store_parent_child_-
    rule:nnn ..... 347, 347, 384
g__tag_struct_0_prop ..... 86
\__tag_struct_add_AF:nn .....
    ..... 550, 577, 593, 612, 619, 659
\__tag_struct_add_inline_AF:nn ..
    ..... 539, 568, 592, 636, 640, 649
\g__tag_struct_AFobj_int .....
    536, 543, 544, 548, 549, 552, 572, 575
\g__tag_struct_cont_mc_prop ....
    ..... 10, 92, 93, 95, 98, 192
\g__tag_struct_dest_num_prop ... 63
\l__tag_struct_elem_stash_bool ..
    ..... 62, 426, 764
\__tag_struct_exchange_kid_-
    command:N ..... 225, 225, 235, 266
\__tag_struct_fill_kid_key:n ...
    ..... 101, 118, 236, 236, 343
\__tag_struct_format_Ref:n .....
    ..... 324, 324, 325
\__tag_struct_get_dict_content:nN
    ..... 102, 119, 295, 295, 344
\__tag_struct_get_id:n .....
    ..... 59, 64, 77, 78, 117, 117, 351, 407
\__tag_struct_get_parentrole:nNN
    ..... 155,
    155, 171, 185, 359, 615, 766, 886, 942
\__tag_struct_gput_data_ref:nn ..
    ..... 521, 986, 986, 998
\__tag_struct_insert_annot:nn ...
    ..... 359, 359, 1004
\l__tag_struct_key_label_tl ....
    ..... 61, 425, 706, 708
\__tag_struct_kid_mc_gput_-
    right:nn ..... 172, 182, 195, 292
\__tag_struct_kid_OBJR_gput_-
    right:nnn ..... 207, 207, 223, 374
\__tag_struct_kid_struct_gput_-
    right:nn 197, 197, 206, 810, 876, 932
g__tag_struct_kids_0_seq ..... 86
\__tag_struct_mcid_dict:n .....
    ..... 95, 98, 172, 187
\g__tag_struct_objR_seq ..... 8
\__tag_struct_output_prop_aux:nn
    ..... 67, 67, 81
\g__tag_struct_ref_by_dest_prop . 66
\g__tag_struct_roletag_NS_tl ... 57
\l__tag_struct_roletag_NS_tl ...
    ..... 60, 724, 729, 756
\l__tag_struct_roletag_tl .....
    ..... 57, 723, 729, 731, 756, 760
\__tag_struct_set_tag_info:nnn ..
    127, 129, 139, 154, 700, 796, 903, 959

\g__tag_struct_stack_current_tl .
    ..... 15, 26,
    35, 66, 72, 84, 136, 144, 150, 186,
    197, 207, 221, 293, 297, 360, 371,
    380, 402, 407, 413, 761, 808, 812,
    813, 816, 834, 840, 877, 884, 933, 940
\l__tag_struct_stack_parent_-
    tmpa_tl ..... 15, 367,
    376, 391, 436, 698, 710, 714, 739,
    767, 779, 788, 805, 809, 811, 814, 817
\g__tag_struct_stack_seq 11, 22, 25,
    366, 614, 713, 719, 762, 827, 832, 838
\c__tag_struct_StructElem_-
    entries_seq ..... 20
\c__tag_struct_StructTreeRoot_-
    entries_seq ..... 20
\g__tag_struct_tag_NS_tl .....
    ..... 57, 448, 703,
    722, 774, 786, 792, 795, 799, 851,
    892, 899, 902, 906, 948, 955, 958, 962
\g__tag_struct_tag_stack_seq ...
    ..... 13, 45,
    187, 188, 389, 404, 418, 759, 831, 845
\g__tag_struct_tag_tl .....
    ..... 57, 164, 165, 168,
    337, 338, 447, 449, 702, 721, 760,
    773, 786, 791, 794, 798, 847, 849,
    891, 898, 901, 905, 947, 954, 957, 961
\__tag_struct_write_obj:n .....
    ..... 132, 326, 326
\g__tag_tagunmarked_bool ... 125, 251
\l__tag_tmpa_box .....
    ..... 99, 168, 174, 175, 179, 190, 191
\l__tag_tmpa_clist .....
    ..... 99, 1039, 1040, 1073, 1074, 1076
\l__tag_tmpa_int ..... 53,
    56, 61, 64, 68, 77, 99, 352, 364, 366, 436
\l__tag_tmpa_prop 99, 157, 165, 178, 180
\l__tag_tmpa_seq .... 99, 240, 242,
    244, 245, 246, 247, 265, 277, 365,
    368, 376, 377, 379, 380, 381, 446,
    447, 448, 1041, 1045, 1055, 1056,
    1057, 1059, 1077, 1083, 1085, 1109
\l__tag_tmpa_str ..... 41,
    42, 47, 104, 262, 267, 272, 283, 288,
    295, 435, 440, 448, 453, 454, 461,
    466, 473, 480, 487, 496, 503, 526, 533
\l__tag_tmpa_tl 42, 43, 50, 51, 57, 65,
    69, 72, 77, 79, 84, 93, 95, 99, 102,
    104, 106, 107, 111, 112, 115, 116,
    119, 124, 139, 140, 142, 144, 147,
    148, 153, 164, 178, 179, 180, 181,
    181, 183, 184, 186, 197, 198, 199,
    204, 207, 215, 216, 216, 218, 219,

```

227, 231, 232, 241, 242, 244, 248, 250, 263, 271, 272, 273, 274, 275, 279, 279, 281, 281, 287, 344, 350, 373, 377, 378, 379, 380, 380, 390, 391, 392, 397, 403, 405, 409, 418, 418, 421, 422, 426, 428, 438, 440, 449, 474, 476, 479, 481, 495, 499, 515, 518, 521, 549, 555, 557, 558, 560, 564, 576, 579, 588, 592, 614, 616, 636, 640, 644, 656, 781, 788, 831, 832, 838, 840, 845, 848, 849, 851, 888, 893, 927, 944, 949, 1053, 1064		\tagmcbegin 33, 141, 22
\l__tag_tmpb_box		\tagmcend 33, 22
. 99, 169, 176, 177, 181, 183		tagmcid 6, 137
\l__tag_tmpb_seq		\tagmcifin 33
. 99, 1040, 1041, 1076, 1077		\tagmcifinTF 33, 39
\l__tag_tmpb_tl 152, 52, 67, 81, 83, 99, 330, 378, 384, 406, 411, 419, 422, 428, 442, 484, 486, 489, 491, 496, 499, 569, 575, 577, 578, 580, 584, 589, 592, 889, 894, 945, 950		\tagmcuse 33, 22
__tag_tree_fill_parenttree:		\tagpdfparaOff 35, 367
. 152, 153, 225		\tagpdfparaOn 35, 367
__tag_tree_final_checks: 20, 20, 327		\tagpdfsetup 33, 91, 92, 140, 6
\g__tag_tree_id_pad_int . . 41, 45, 122		\tagpdfsuppressmarks 35, 377
__tag_tree_lua_fill_parenttree:		tagstruct 6, 137
. 205, 205, 222		\tagstructbegin 34, 140, 141, 45, 198
__tag_tree_parenttree_rerun_-		\tagstructend 34, 45, 199
msg: 152, 192, 227		tagstructobj 6, 137
__tag_tree_write_classmap:		\tagstructuse 34, 45
. 261, 261, 331		\tagtool 33, 13
__tag_tree_write_idtree: . . . 49, 329		tagunmarked _␣ (setup-key) 6, 251
__tag_tree_write_namespaces:		TeX and L ^A T _E X 2 _ε commands:
. 295, 295, 332		\@M 165
__tag_tree_write_parenttree:		\@auxout 67
. 218, 218, 328		\@bsphack 154
__tag_tree_write_rolemap:		\@cclv 352
. 238, 240, 258, 330		\@esphack 156
__tag_tree_write_structelements:		\@gobble 24, 48
. 128, 128, 333		\@ifpackageloaded 28
__tag_tree_write_structtreeroot:		\@ifundefined 342
. 89, 91, 112, 334		\@kernel@after@foot 391
__tag_whatsits: 36, 54, 55, 58, 318, 319		\@kernel@after@head 389
tag-namespaces _␣ (rolemap-key) 666		\@kernel@before@cclv 349
tag/struct/0 internal commands:		\@kernel@before@foot 390
__tag/struct/0 29		\@kernel@before@footins 345, 347
tag/tree/namespaces internal commands:		\@kernel@before@head 386, 388
__tag/tree/namespaces 294		\@mainaux 193
tag/tree/parenttree internal commands:		\@makecol 351
__tag/tree/parenttree 135		\@maxdepth 178
tag/tree/rolemap internal commands:		\@mult@ptagging@hook 354
__tag/tree/rolemap 234		\@secondoftwo 24, 48
tagabspace 6, 137		\c@page 351
tagmcabs 6, 137		\count@ 359
		\mult@firstbox 357
		\mult@rightbox 361
		\page@sofar 356
		\process@cols 357
		tex commands:
		\tex_botmarks:D 88
		\tex_firstmarks:D 85
		\tex_kern:D 181
		\tex_marks:D 22, 31, 44, 51, 62, 68
		\tex_special:D 58
		\tex_splitbotmarks:D 214
		\tex_splitfirstmarks:D 194
		\the 351
		\tiny 266, 277
		title _␣ (struct-key) 90, 422

